



БУРГАСКИ СВОБОДЕН УНИВЕРСИТЕТ

ЦЕНТЪР ПО ИНФОРМАТИКА И ТЕХНИЧЕСКИ НАУКИ

ДИПЛОМНА РАБОТА

за придобиване на образователно-квалификационна степен
„бакалавър“

на тема

Платформа за споделяне на екрана в уеб
браузъра чрез ASP .NET SignalR

Дипломант

/Георги Димов/

Научен ръководител

/доц. д-р Димитър Минчев/

Факултетен номер: 16321042

Бургас 2022 г.



БУРГАСКИ СВОБОДЕН УНИВЕРСИТЕТ

ЦЕНТЪР ПО ИНФОРМАТИКА И ТЕХНИЧЕСКИ НАУКИ

Декан:.....

/ проф. д-р. Р. Долчинков /

ЗАДАНИЕ

за дипломна работа

на студента: **Георги Димов**

факултетен номер: **16321042**

специалност: **Софтуерно Инженерство**

Образователна-квалификационна степен: **Бакалавър**

Тема: Платформа за споделяне на екрана в уеб браузъра чрез ASP .NET
SignalR

Изходни данни: Microsoft, Visual Studio, .NET, C#, ASP.NET, SignalR.

Съдържание:

Увод, цел и задачи.

Глава 1. Характеристика на платформата

Глава 2. Избор и обосновка на използваните технологии

Глава 3. Разработване на платформата

Заклучение и перспективи за развитие.

Дата на задаване: 01.02.2021 г.

Срок за предаване: 01.05.2021 г.

Ръководител: доц. д-р Минчев

Подпис:.....

Дипломант: Георги Димов

Подпис:.....

Пр. Координатор: доц. д-р Георгиева

Подпис:.....

Увод, цел и задачи

Една от най-бързо развиващите се сфери в страната и глобално е информационните технологии. Този сектор привлича все повече внимание с много високо заплащане и добри условия на труд. Поради тези причини се наблюдава тенденция, в която организации и фирми се обвързват все по-силно със софтуерни продукти. Всички големи и повечето новосъздадени компании искат да предложат на своите служители и клиенти модерен, бърз, ефективен и гъвкав софтуерен продукт.

Целта на тази дипломна работа е да покаже точно такъв тип софтуерен продукт, като ще се разгледа в детайли всичко за неговото пълно реализиране. Основната крайна цел е да се покаже работещо софтуерно решение за споделяне на екран и уеб съдържание в реално време в уеб браузъра, чрез технологията на Майкрософт .NET SignalR¹.

Все повече компании целят техния продукт да бъде изцяло уеб базиран, понеже така крайния продукт е изцяло независим от операционната система на крайния потребител. Всичко от което се нуждае той е уеб браузър. Чрез този метод няма нужда от никакви инсталационни файлове и също така компаниите могат драстично да намалят своите разходи с помощта на уеб базираните приложения поради намалената нужда от поддръжка, по-ниските изисквания към системата на крайния потребител и опростената архитектура.

Тази платформа за споделяне на екрана в уеб браузъра е с отворен код, като тя може да бъде достъпна в хранилище на GitHub², където има инструкции за стартирането, използването и необходимия лиценз за платформата. Софтуерът с отворен код е софтуер с изходен код, който всеки може да провери, модифицира и подобри.

Основна задача на тази дипломна работа е да покаже подробно и стъпка по стъпка как би се изготвил целия проект, като това включва: използвани технологии, създаване на самия проект, използвани версии на всички инструменти, структура и софтуерна архитектура, използвани класове, връзките между компонентите, клиентски код. Тук ще включва основни и малки части от самия код, като те ще бъдат съпроводени с обяснения и каква функция имат за цялостната работа на платформата.

Друга задача е да се демонстрира приложението на тази платформа, както и нейната функция и защо тя има перспектива и по-нататъшно развитие. За по-добро разбиране цялостната идея ще използвам различни блок схеми и изображения, които ще имат за цел по-добро обяснение и визуализиране на цялостната работа на платформата.

¹ .NET SignalR = SignalR е безплатна софтуерна библиотека с отворен код за Microsoft ASP.NET, която позволява на сървърния код да изпраща асинхронни известия до клиентски уеб приложения, вж: <https://dotnet.microsoft.com/apps/aspnet/signalr>

² GitHub = уеб базирана услуга за разполагане на софтуерни проекти и техни съвместни разработки върху отдалечен интернет сървър в т.нар. хранилище (software repository), вж: <https://github.com/about>

Идеята за този проект се роди в главата ми, когато работех като разработчик на софтуер (software developer) за компания, чийто основен програмен продукт е за оторизирани сервиси на коли. Служителите на тези сервиси използват нашия софтуер, който е изцяло уеб базиран и сравнително често ни даваха обратна връзка, че имали затруднения, когато правили оферта на клиенти, които искат да закупят чисто нов автомобил. Проблема бил там, че един нов автомобил може да е с различни двигатели, джанти, интериорни екстри и много други приспособления, които влияят директно на крайната цена, както и, че някои екстри зависят от предварително избрани автомобилни модули. Служителите на тези сервиси смятат, че ще е много удобно, ако можеше клиентите да участват в реално време в изготвянето на оферта за дадено превозно средство. По този начин служител и клиент/и могат заедно да виждат в реално време офертата за даден автомобил и да я изготвят напълно индивидуално, лесно и заедно. Ето така се роди идеята, която предложих и беше одобрена. След това направих доказване на концепция (proof of concept) и накрая реализираното решение. В тази дипломна работа искам да покажа основата на това работещо решение, като тя е пренасяне на данни в реално време по уебсокет (и не само), чрез технологията на Microsoft .NET SignalR.

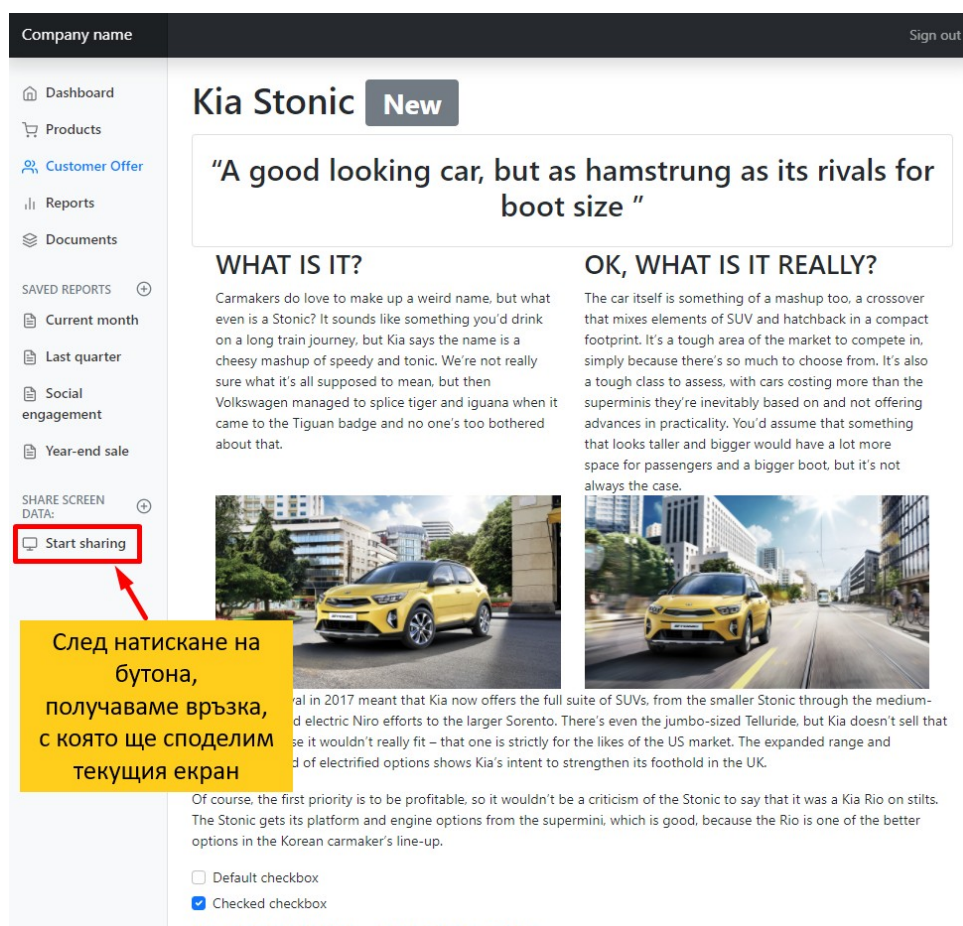
Глава 1. Характеристика на платформата

1.1 Приложение и Функция

В случай, че платформата за споделяне на екран бъде интегрирана правилно, то тя би намерила много широко приложение в различни сфери. Тя е написана максимално абстрактно, като по този начин споделянето на екрана е абсолютно независимо от съдържанието, функционалността и уеб страницата на крайния клиент. Ако даден клиент иска да я използва, това може да стане с няколко реда код и вмъкване на бутон за споделяне на екран в неговата страница.

Примерни приложения на тази платформа биха били в сектори, които споделянето на екран би улеснило значително комуникацията между клиент и купувач. Например при покупко-продажба на апартамент, в случай че дадената агенция използва тази платформа, то тя може много лесно да сподели своята уеб страница съдържаща офертата за покупко-продажба. Така клиента може да види скица, квадрати и как крайната цена би се променила в реално време, според неговите избори на завършеност, включване на газова инсталация или други опции за дадено жилище.

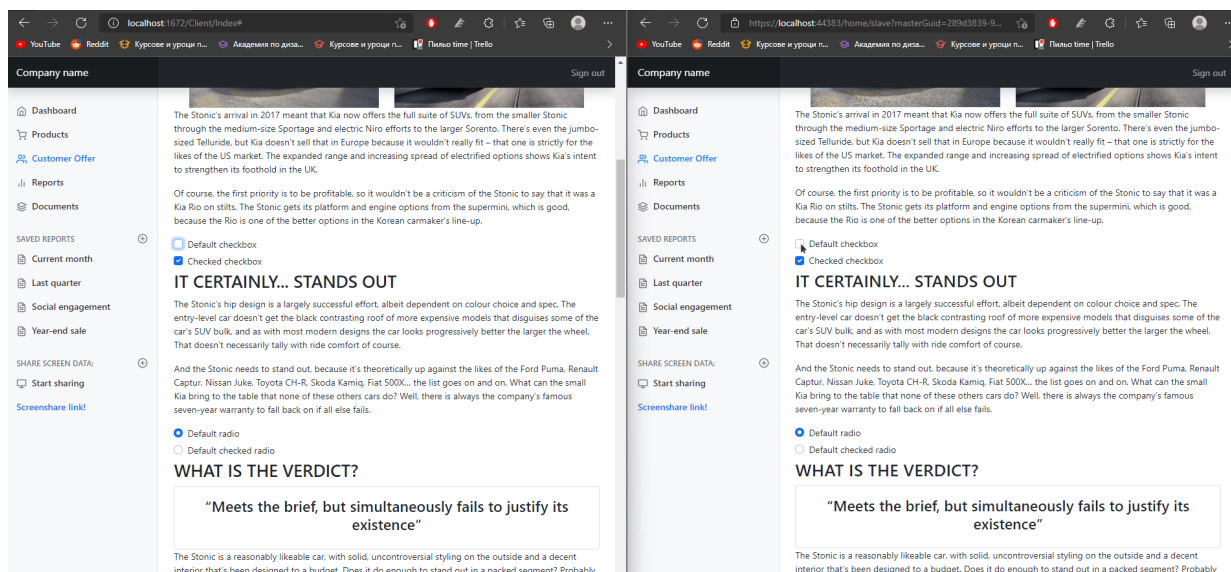
Направил съм един тестови клиент (.NET MVC проект), който много добре описва приложението и различните функции на платформата за споделяне на екрана в реално време в уеб браузъра. Примерния клиент е оторизиран сервиз за продажби на автомобили. Служителите в този сервиз продават нови автомобили, като за тяхната продажба се изготвя напълно индивидуална оферта за всеки клиент. Офертата е индивидуална, понеже самия автомобил може да бъде произведен с газов, дизелов или хибриден двигател. Лети джанти може да са допълнителна опция, както и усъвършенствано шофьорско табло. Всички тези опции биха повлияли на крайната цена на автомобила. За удобство на служителите, те могат да споделят своята уеб страница съдържаща дадената автомобилна оферта и да я изготвят в реално време заедно с клиента, като по този начин клиента може да вижда как се променя цената. Клиента също така може да проверява дали неговите данни, като адрес, име и други са правилно изписани от служителите, намалявайки значително допускането на различни грешки. На Фиг. 1 може да видим как изглежда примерния тестови клиент.



Фиг. 1 Клиент използващ платформата за споделяне на екран

След натискане на бутона (*Start sharing*), показан във Фиг. 1, получаваме връзка, която може да изпратим на един или повече потребители (още наричани *slaves*). Един или повече потребители могат да бъдат закачени за наблюдаване на екрана на дадения клиент (още наричан *master*), като всички промени ще се отразят едновременно при всички потребители. Връзката, която имаме тук е един към много. От тук насетне за удобство ще наричаме **клиент** или още **master**, този който презентира своя екран и **потребител/и** или **slaves**, тези, които са зрители на екрана на клиента. Един master може да споделя своя екран на много slaves. Те от своя страна могат само да наблюдават страницата без да правят каквито и да било промени по нея.

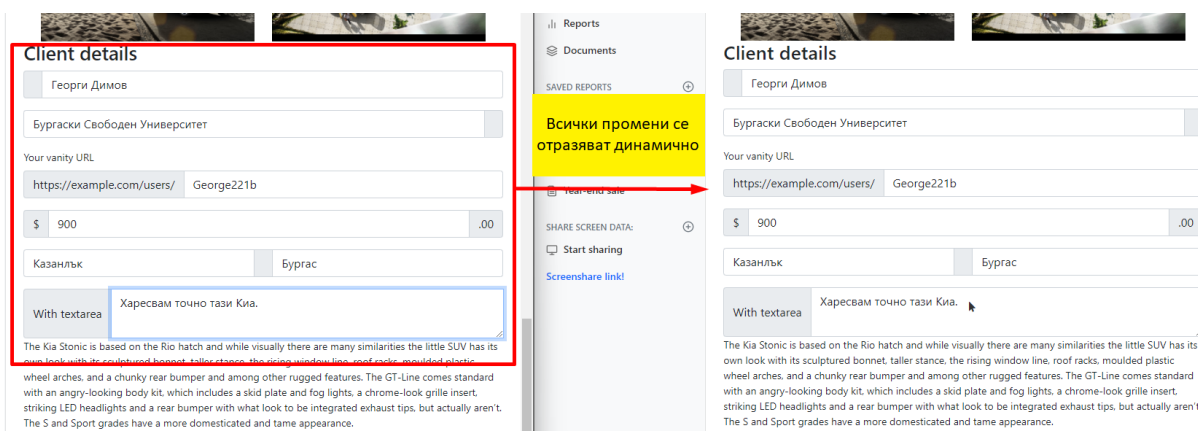
След като връзката за споделяне на екрана бъде отворена от един или повече потребители, съдържанието на клиентската страница се зарежда мигновено при тях. Всички действия и промени на клиента се отразяват динамично при неговите потребители. Като това се случва без да е нужно опресняване на страницата. На Фиг. 2 може да видим как се е заредила страницата на клиента и потребителя.



Фиг. 2 Потребител (дясно) отворил връзка за споделяне на екран на клиент (ляво). Съдържанието се зарежда мигновено.

Функциите, които предлага платформата за споделен екран, са динамичното зареждане на страницата при всеки потребител в зависимост от всички действия, които предприема клиента на неговата страница. Тези действия включват:

1. Промяна съдържанието на текстови полета (HTML³ input, select, textarea⁴ елементи) и всички възможни елементи за промяна на техния статус/избор.

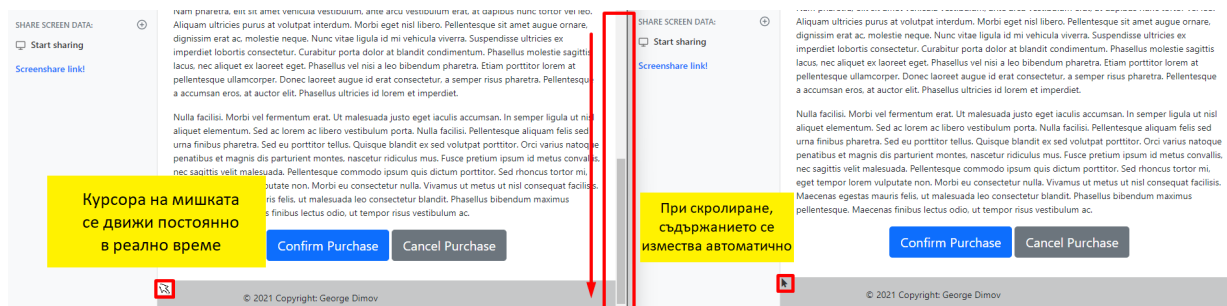


Фиг. 3 Демонстрация на функцията за динамично пренасяне на съдържание въведено в текстови форми.

2. Скролиране на страницата, вертикално или хоризонтално.
3. Местене на курсора в страницата.

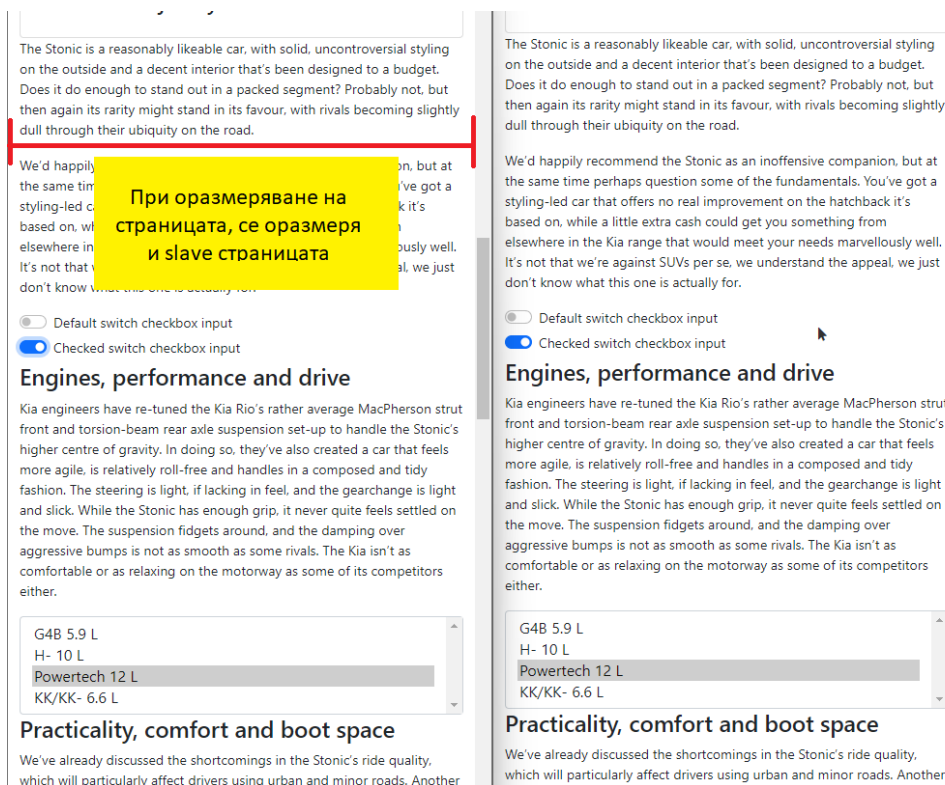
³ HTML = HTML (HyperText Markup Language, произнасяно най-често като „ейч-ти-ем-ел“) е основният маркиращ език за описание и дизайн на уеб страници.

⁴ input, select, textarea = видове HTML елементи, които служат за приемане на данни от клиента.



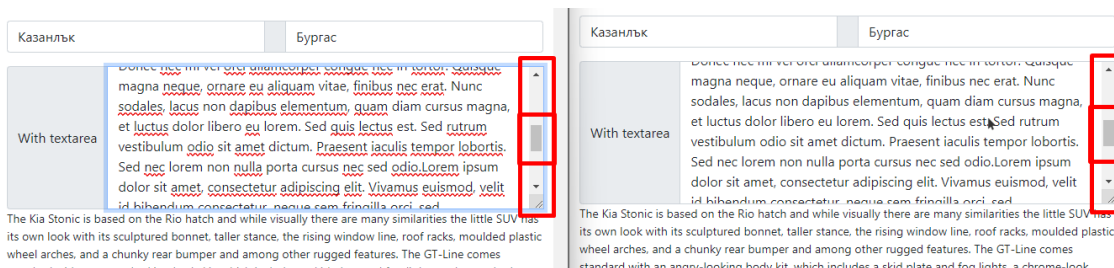
Фиг. 4 Позицията на курсора (точки X, Y) и скролиране (хоризонтално и вертикално).

4. Оразмеряване на страницата.



Фиг. 5 Оразмеряване на страницата. Свиване на страницата хоризонтално. Промените се отразяват съобразено с това.

- В случай, че в даден елемент се появи скрол, като това примерно би се случило при въвеждане на много информация в textarea HTML елемент, то скролирането в този елемент се пренася също.



Фиг. 6 При поява на скрол в даден/и елементи, скролирането в тези елементи се пренасят в съответствие със скрол позицията на клиента.

6. При сложни промени в DOM⁵ дървото на клиента, като това се конфигурира допълнително.

За да бъде по-ясно демонстрирано приложението и функциите на платформата за споделен екран съм използвал Фиг. 2 до Фиг. 6. За още по-добра демонстрация може да се изгледа видео, което показва в рамките на една минута много точно всички функции на платформата. Това видео може да бъде намерено на следната интернет страница:

<https://github.com/George221b/screenshare-in-the-web-signalr>

Тук се намира целия source code⁶ на платформата за споделен екран, както и оказания за нейното използване.

1.2 Схема на изпълнение

Във Фиг. 7, 8 и 9 съдържат три блок схеми, чиято цел е да демонстрират цялостната работа, ход и изпълнение на платформата за споделен екран.

След като клиент отвори уеб страница, в която има интегрирана платформата за споделен екран, то в клиентската част на приложението трябва да има бутон, който ще създаде връзка за споделяне с потребителите. Това може да се види визуално във Фиг. 1.

Фиг. 7 показва, че когато клиент се намира във клиентската част на приложението, то той взаимодейства с нашия backend API⁷. Това се осъществява, чрез изпращане на HTTP⁸ request⁹. След успешна заявка до сървъра (API) се случват няколко неща. Първото е, че се записва дадения клиент/master в зададеното място за хранилище на сървъра. Второто е, че се е генерирал уникален master GUID и линк, който може да бъде препратен на потребителите. Последното нещо, което се случва след успешната заявка е, че се отваря SignalR връзка с нашия сървър. Тя не се осъществява по HTTP протокола, а по WS протокола¹⁰. В по-голяма дълбочина ще разгледаме темите за SignalR и WebSocket в глава 2.3 (SignalR). След успешна връзка със SignalR ние сме запазили уникалното “ConnectionID” на дадения клиент, както и сме записали цялата информация за него. Като това включва

⁵ **DOM** = Document Object Model = Документен обектен модел. Когато се зареди уеб страница, браузърът създава обектен модел на документа на страницата. HTML DOM моделът е конструиран като дърво от обекти.

⁶ **source code** = изходен код. Изходният код е сбор от инструкции (заедно с коментарите), написан на разбираем за човека език за програмиране обикновено като текст. Изходният код позволява модификация на компютърната програма, разглеждане на начина, по който тя работи, откриване на грешки и други действия.

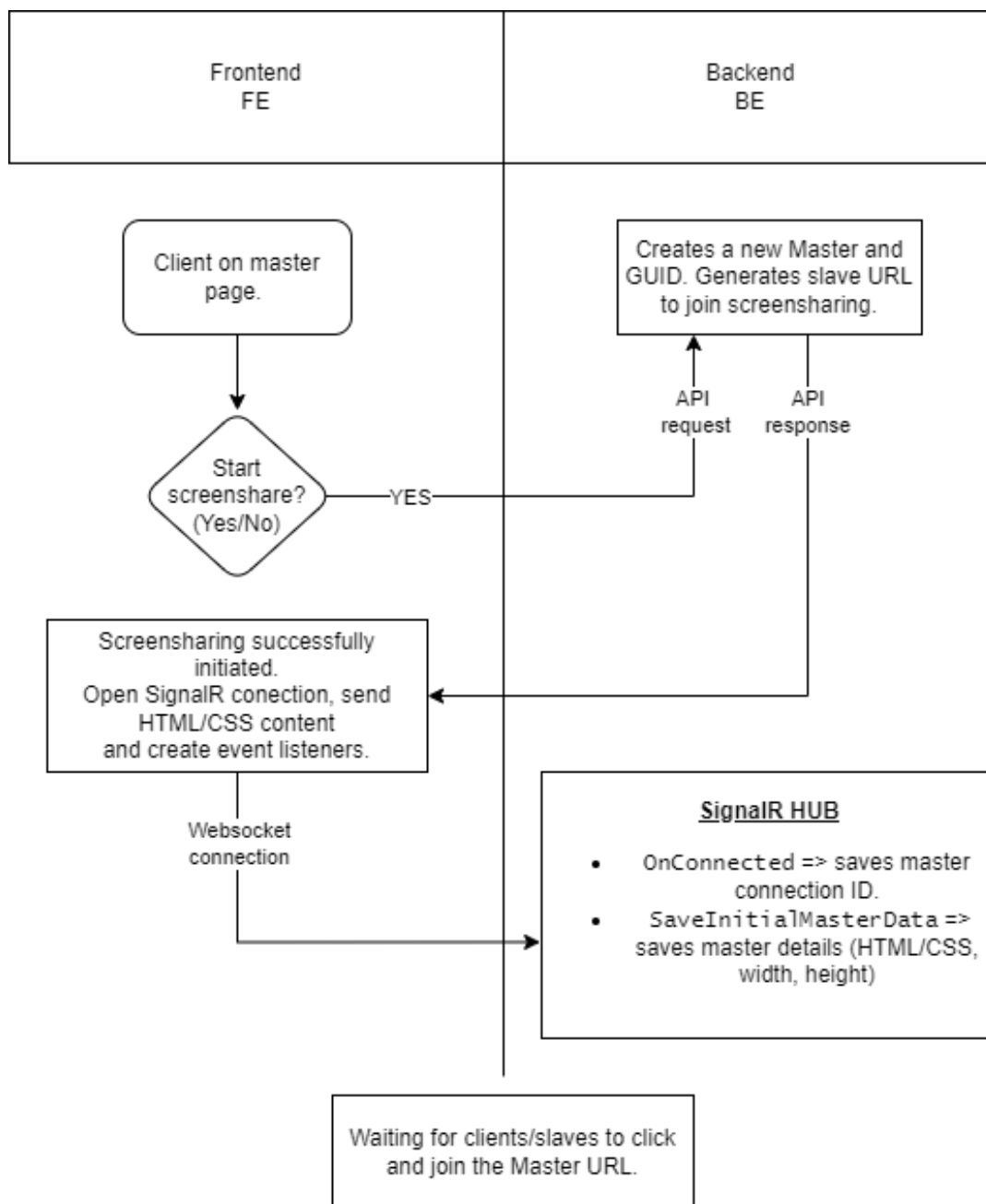
⁷ **API** = Приложно-програмният интерфейс (на английски: Application Programming Interface, API) е интерфейсът на изходния код, който операционната система или нейните библиотеки от ниско ниво предлагат за поддръжката на заявките от приложния софтуер или компютърните програми.

⁸ **HTTP** = Протокол за пренос на хипертекст (на английски: Hypertext Transfer Protocol, HTTP) е мрежов протокол, от приложния слой на OSI модела, за пренос на информация в компютърни мрежи.

⁹ **HTTP request** = HTTP заявка се прави от клиент до наименуван хост, който се намира на сървър. Целта на заявката е достъп или запис на ресурс на сървъра. За да направи заявката, клиентът използва URL (Uniform Resource Locator), който включва информацията, необходима за достъп до ресурса.

¹⁰ **WebSocket протокол** = WebSocket е компютърен комуникационен протокол, осигуряващ двупосочна интерактивна комуникационна сесия между браузъра на потребителя и сървъра през една TCP връзка.

неговия HTML, CSS¹¹, дължина и височина на страницата. След това се закачат всички event listeners¹² за всички клиентски действия. Накрая като имаме цялата информация за дадения master/клиент и сме закачили всички event listeners, остава само да изчакаме свързването на потребители/slaves с нашата платформа. Като това свързване се получава след отваряне на уникалната връзка за споделяне на екрана от страна на потребител.

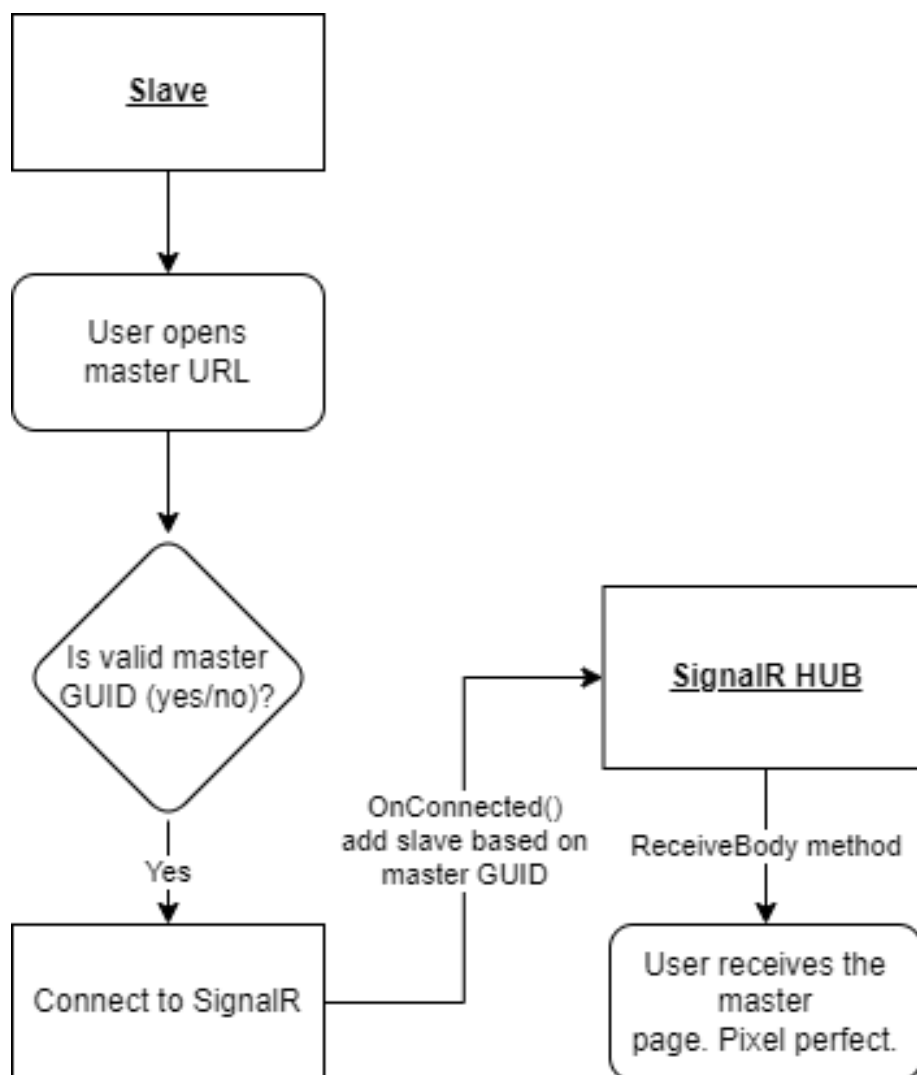


Фиг. 7 Първа блок схема демонстрираща хода на действията в платформата.

¹¹ CSS = (Cascading Style Sheets) е език за програмиране и също описание на уеб дизайн програмни стилове – използва се основно за описание на онлайн представянето на уеббазиран документ, който написан на език за маркиране. Вж. повече: <https://bg.wikipedia.org/wiki/CSS>

¹² event listener = Слушател на събитие е процедура или функция в компютърна програма, която изчаква настъпването на събитие. Примери за събитие са потребителят щракване или преместване на мишката, натискане на клавиш на клавиатурата, оразмеряване на прозорец и други.

Фиг. 8 демонстрира следващия етап на действия в платформата за споделен екран.

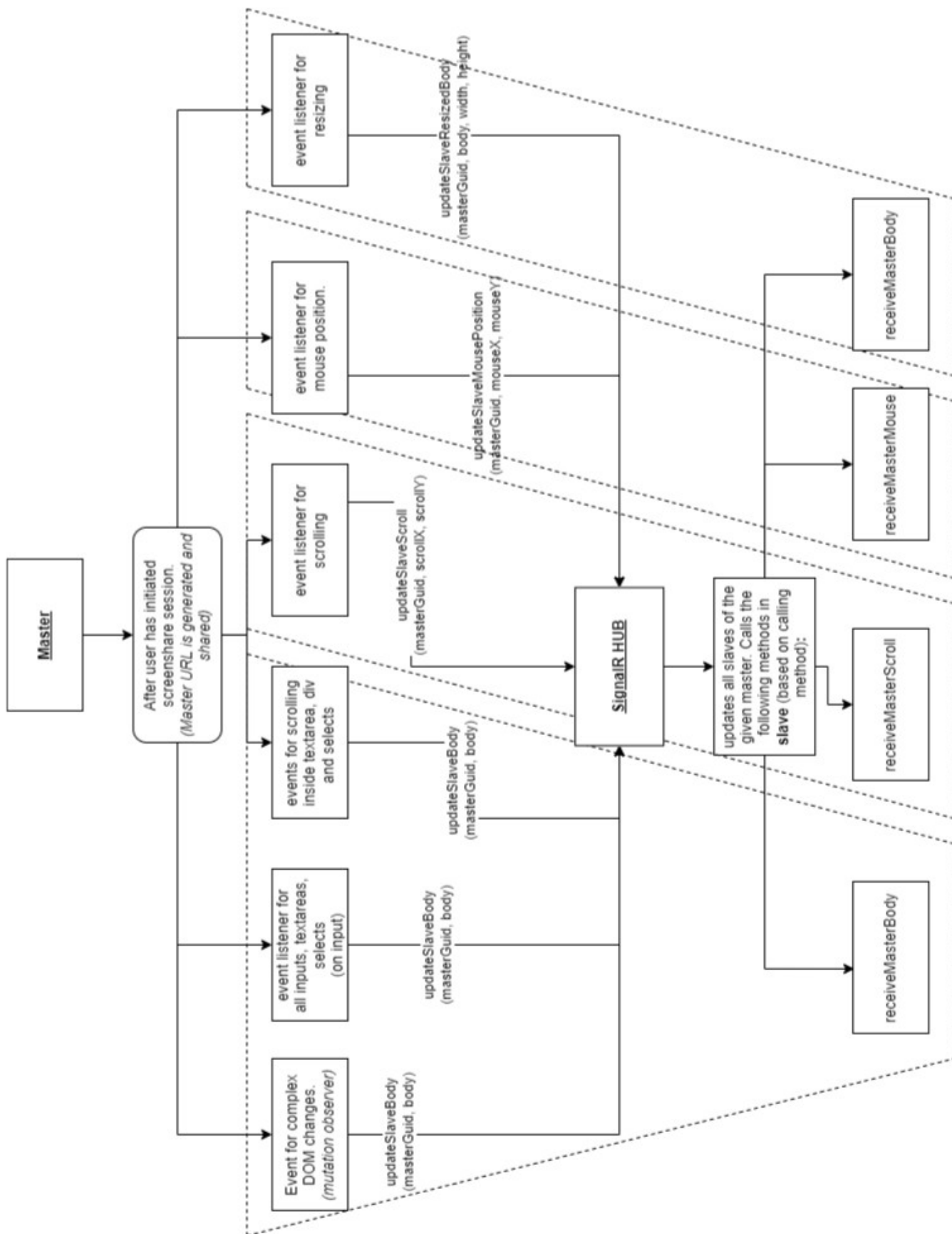


Фиг. 8 Втора блок схема демонстрираща хода на действията след като се присъедини потребител към платформата за споделен екран.

След като даден потребител се опита да се присъедини и да наблюдава екрана на клиента, чрез отваряне на връзката се случват две основни неща. Първо се валидира дадения master GUID. Ако той е валиден, потребителя отваря и от своя страна SignalR връзка със сървъра. След успешна връзка със SignalR и сървъра, потребителя зарежда цялата клиентска страница, като тя се визуализира в `iframe`¹³ HTML елемент. Визуализираната страница е перфектно копие пиксел по пиксел на клиентската страница, като това се постига с вече запазените размери на прозореца на клиента. Всяко едно действие от страна на клиента се отразява при потребителя в реално време, чрез вече отворената двупосочна връзка базирана на SignalR и вебсокети.

¹³ **iframe** = HTML елемент, който се използва за показване на уеб страница в уеб страница.

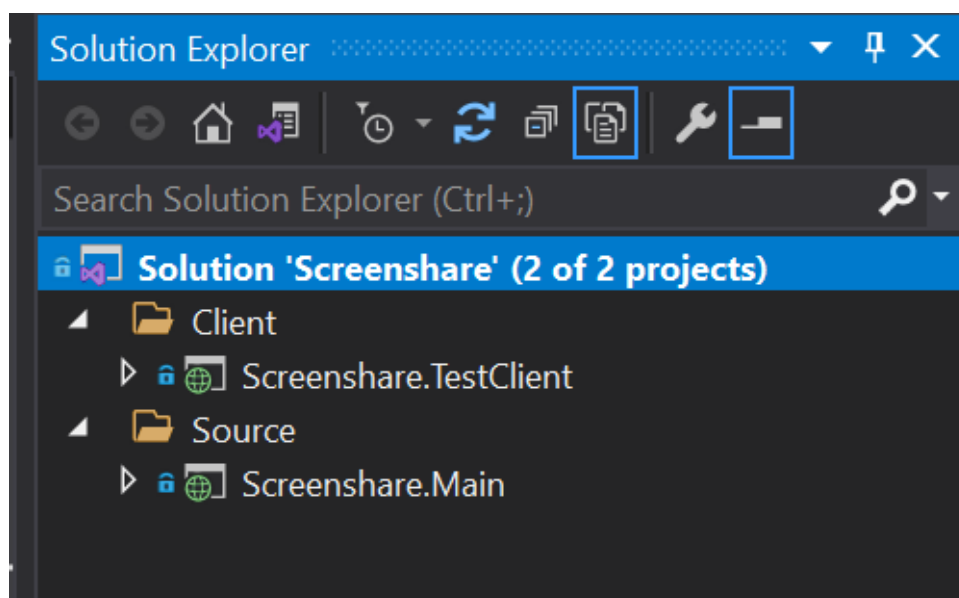
Последната блок схема Фиг. 9 демонстрира множество от методи, които се извикват след като бъде изпълнено дадено събитие от master. Всяко събитие, което се е случило на потребителския интерфейс извиква определен метод в сървърната част на нашата платформа. Пример за това би била следната ситуация - клиент въвежда информация в HTML елемент, като например в `textarea`. След това моментално се извика закаченото събитие за промяна на съдържанието на дадения елемент, който от своя страна прави връзка със SignalR и сървър, където се извиква точно определен метод за реакция на това събитие. Като съответната реакция на това събитие, би била пренасянето на дадените промени от клиента до всички закачени потребители/slaves. Гледайки Фиг. 9, може да стигнем до заключението, че всички методи намиращи се **над** SignalR Hub се изпълняват след събитие направено от клиента, а всички методи **под** SignalR Hub се изпълняват съответно на сървърната част на приложението. Някои събития извикват еднакви методи в backend частта, като това се случва с цел избягване на повтарящ се код и преизползване на вече написаните методи. Така поддръжката става в пъти по-лесна и кода по-четим и изчистен. Различните методи са разделени в групи, като всяка група е заградена с прекъсната линия. Разделяме ги според това, кой метод се изпълнява на сървър.



Фиг. 9 Трета блок схема, която показва с кои методи клиента взаимодейства заедно с SignalR сървърна част. Методите са визуално отделени в групи.

1.3 Архитектурна характеристика

Платформата за споделяне на екран може да бъде разгледана и разграфена по много начини, според различните ѝ характеристики. Ако тя бъде разгледана максимално абстрактно, като софтуерна архитектура, абстрахирайки се от архитектурата на самите проекти, ще забележим, че имаме две основни папки в нашия Solution¹⁴ файл. Solution файлът представлява структура за организиране на различните проекти във Visual Studio¹⁵. Той съдържа информация за състоянието на проектите, техните референции и средата за разработка.



Фиг. 10 Структура на Solution файла във Visual Studio

Папка **Source** съдържа backend частта на платформата. Backend е сървърната страна на един уебсайт. Той съхранява и поддържа данни, а също така гарантира, че всичко от клиентската страна на уебсайта работи добре. Това е частта от уебсайта, която не може да се види, визуализира или да се взаимодейства с нея. Backend не влиза в пряк контакт с потребителите и частите и спецификациите, разработени от тези програмисти, са косвено достъпни от потребителите чрез приложение от клиентския край. Дейности, като писане на API, създаване на библиотеки и работа със системни компоненти без потребителски интерфейси или дори системи за научно програмиране, също са включени в backend.

В случая на нашата платформа имаме един “Screenshare.Main” проект, съдържащ няколко основни компонента:

- Hubs/ScreencastingHub.cs
- Controllers/ScreencastController.cs

¹⁴ **Solution** = .sln файл, текстово базиран, споделен, вж:

<https://docs.microsoft.com/en-us/visualstudio/ide/solutions-and-projects-in-visual-studio?view=vs-2019>

¹⁵ **Visual Studio** = мощна интегрирана среда за разработка на софтуерни приложения за Windows и за платформата .NET Framework, вж: <https://visualstudio.microsoft.com/>

- Views/Home/Slave.cshtml

Първия компонент е “ScreencastingHub” класа, който наследява базов абстрактен клас “Hub” от библиотеката *Microsoft.AspNet.SignalR*, която съдържа API на SignalR Hubs. API на SignalR Hubs ни позволява да осъществяваме RPC ¹⁶ повиквания от сървър към свързани клиенти и от клиенти към сървъра. Методите дефинирани на сървъра могат да бъдат извиквани от клиенти, както и да се извикват функции, които ще се изпълняват при клиентите. В клиентския код дефинирате функции, могат да се извикват от сървъра, както и да се извикват методи, които ще се изпълняват на сървъра.

В Hub класа се изпълнява главната backend работа. Тук са разположени основните методи, които осъществяват споделянето на екрана. Именно тук се изпълнява бизнес логика, според това дали инициатора е *master* или някои от неговите *slaves*, още наричани подчинени/роби, които ще гледат споделяния екран. Този модел се нарича Master/Slave¹⁷.

Другия основен backend компонент в платформата е контролера ¹⁸ “*ScreenCastController*”. Той се извиква първоначално преди да е започнало споделянето на екран в платформата. Отговорен е за създаването на master и генерирането на уникален GUID¹⁹, както и линк/връзка, на която може да бъде осъществено споделянето на екрана и закачането на slaves.

Последния много основен компонент в Main проекта е “*Slave.cshtml*” view²⁰. Тук използваме най-широко разпространения дизайн Модел-Изглед-Контролер²¹. Клиентите, които достъпват този изглед са slaves. Тук се очаква задължително подаден query ²²параметър, който идентифицира точно на кой master да бъде споделян екрана. В

¹⁶ **RPC** = Remote procedure call = Отдалечено извикване на процедура е, когато компютърна програма кара процедура (подпрограма) да се изпълни в различно адресно пространство, което е кодирано, сякаш е нормално (локално) извикване на процедура, без програмистът изрично да кодира подробностите за отдалеченото взаимодействие. Тоест, програмистът пише по същество същия код независимо дали подпрограмата е локална за изпълняващата програма или отдалечена.

¹⁷ **Master/Slave методология** = Master/slave е модел на асиметрична комуникация или контрол, при който едно устройство или процес ("master") контролира едно или повече други устройства или процеси ("slave") и служи като техен комуникационен център.

¹⁸ **Controller** = В уеб API контролер е обект, който обработва HTTP заявки.

¹⁹ **GUID** = Универсално уникален идентификатор (UUID) е 128-битов етикет, използван за информация в компютърните системи. Терминът глобален уникален идентификатор (GUID) е едно и също нещо, но най-често използван в света и технологиите на Microsoft. Когато се генерират съгласно стандартните методи, UUID/GUID са, за практически цели, уникални.

²⁰ **View** = Изгледът (View) се използва за показване на данни с помощта на обект клас модел. Папката Views съдържа всички файлове на изглед (views) в приложение на ASP.NET MVC. Контролерът може да има един или повече методи за действие (actions) и всеки метод за действие може да върне различен изглед. Накратко, контролер може да изобразява един или повече изгледи.

²¹ **Модел-Изглед-Контролер**: Model View Controller (MVC) = MVC е модел за проектиране, използван за отделяне на потребителски интерфейс (изглед), данни (модел) и логика на приложение (контролер). Този модел помага да се постигне разделяне на притесненията. Вж: https://bg.wikipedia.org/wiki/ASP.NET_MVC

²² **Query string** = Query string е част от единен локатор на ресурси (URL, uniform resource locator), който приписва стойности на определени параметри (query parameters). Query string обикновено включва полета, добавени към основен URL адрес от уеб браузър или друго клиентско приложение, например като част от HTML формуляр.

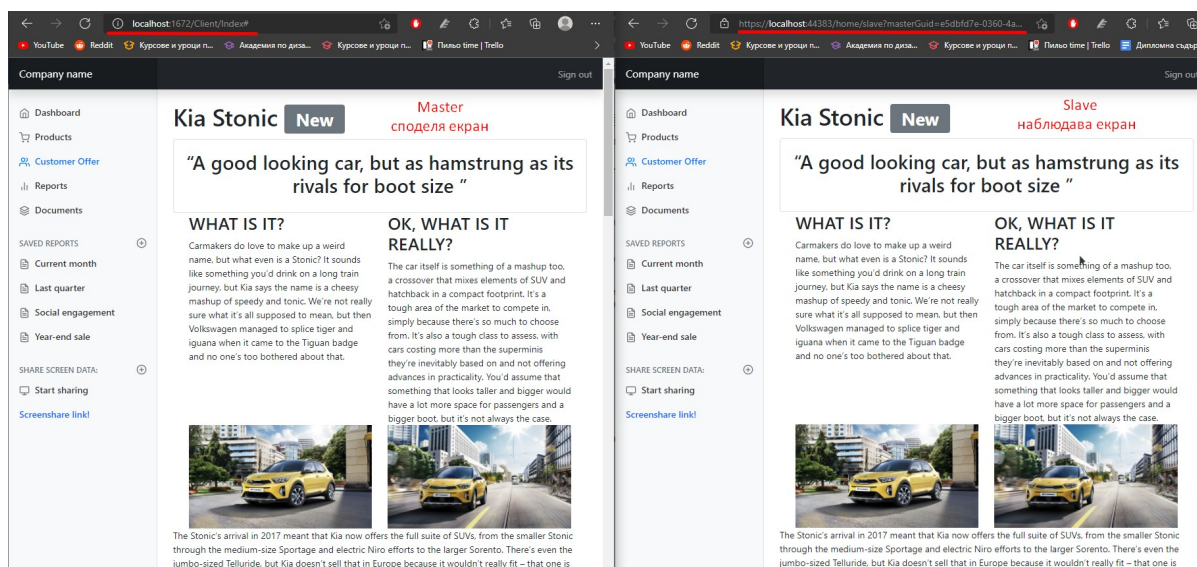
случай, че достъпим този изглед без подадения уникален идентификатор (GUID) за master, ще видим грешка, че въведения URL е грешен.

No GUID sent

Please, check if your provided link is correct!

Фиг. 11 Грешно въведена връзка за споделяне на екран

В случай на правилно въведена мрежова връзка тук веднага ще се зареди съдържанието на страницата в която се намира дадения master и в реално време ще се наблюдава неговите действия в страницата. Тези действия включват скролиране, оразмеряване на прозореца, местене на курсора и въвеждане на данни в различни полета.



Фиг. 12 Успешно установена връзка за споделяне на екран между master и slave

Важно нещо да се отбележи е, че клиентското приложение за оторизирани сервиси (master) се изпълнява на различен domain²³ от този на потребителите, които наблюдават екрана (slaves). В случая на тестовото приложение master се изпълнява на <http://localhost:1672/>, докато slaves се изпълняват на <https://localhost:44383/>. Тези домейни ще са различни и индивидуални според различните клиенти, но в общия

²³ **domain** = Име на домейн е интернет адресът, където хората могат да намерят вашия уебсайт. Всеки уебсайт в интернет се хоства на сървър. Всеки от тези сървъри има адрес за интернет протокол (IP) - набор от 4 числа между 0 и 255, разделени с точки, които казват на компютъра как точно да го достигне. IP адресите, макар и полезни за компютрите, не са точно приятелски настроени и това е мястото, където имената на домейни влизат в игра. Вж: <https://bg.wikipedia.org/wiki/%D0%94%D0%BE%D0%BC%D0%B5%D0%B9%D0%BD>

случай домейна на master се очаква да е frontend ²⁴ частта на едно приложение, докато slave ще се изпълнява, където е API или самия backend²⁵.



Фиг. 13 Споделянето на екран се изпълнява на различни домейни.

1.4 Софтуер с отворен код

Терминът с софтуер с отворен код се отнася до всяка програма, чийто изходен код е предоставен за използване или модификация от потребители или други разработчици. За разлика от собствения софтуер, този с отворен код е компютърен софтуер, който се разработва публично или като част от отворено сътрудничество и е свободно достъпен за обществеността. Отвореният код изиграва голяма роля в общността за разработка на софтуер. Всъщност се разработват поколения от инструменти за работа с отворен код, които се използват днес от разработчиците. Тези инструменти помагат за съхранение, подобряване и отстраняване на проблеми с отворен код в първите дни на разработката на софтуера. Инструмента, който е избран и използван за съхранение на целия код на платформата за споделен екран е GitHub. Това е най-разпространия, лесен за използване и с много интуитивен потребителски интерфейс инструмент.

<https://github.com/George221b/screenshare-in-the-web-signalr>

(връзка до отворения код на платформата за споделен екран)

Има редица други популярни софтуери, които попадат под чадъра с отворен код. Примерите включват:

- Red Hat софтуер. Софтуерна платформа с отворен код и производна на IBM, която предоставя разнообразие от приложения за производителност на ниво предприятие.

²⁴ **frontend** = Предният интерфейс на софтуерна програма или уебсайт е всичко, с което потребителят взаимодейства. От потребителска гледна точка, предния интерфейс е синоним на потребителския интерфейс. Една от основните цели на разработката на интерфейса е да създаде гладко или „безпроблемно“ потребителско изживяване. С други думи, предният край на приложение или уебсайт трябва да бъде интуитивен и лесен за използване.

²⁵ **Backend** = сървърната страна на приложение и всичко, което комуникира между базата данни и браузъра, вж: https://bg.wikipedia.org/wiki/Front-end_%D0%B8_back-end

- LibreOffice. Офис пакет за производителност с отворен код, подобен на програмите на Microsoft Office.
- Програма за манипулиране на изображения на GNU. Инструмент за манипулиране на изображения с отворен код на GNU с подобни компоненти на Adobe Photoshop.
- VLC Media Player. Плейър за аудио и видео файлове с отворен код.
- Linux. Безплатна операционна система с отворен код. Най-големия проект с отворен код в света.

Предимства на софтуера с отворен код:

- По-ниски разходи: Софтуерът с отворен код обикновено не изисква лицензионна такса и често от случаите е напълно безплатен. Това е една от основните причини, поради които малките предприятия избират да приемат и използват този софтуер.
- Гъвкавост: Тъй като целия изходен код е публично достъпен и видим, то програмистът може да вземе целия стандартен софтуерен пакет и да го модифицира по начин, който да отговаря по-добре на него или бизнеса.
- Надеждност и качество: Невъзможно е да се каже, че софтуерът с отворен код е по-добър от собствения софтуер/публично недостъпен или обратното. По отношение на надеждност и качество – и двете имат огромна гама от продукти. Въпреки това, отворен код, който се е утвърдил с времето и е бил използван от множество от разработчици, обикновено се смята, че софтуерът е с добро качество и надеждност, тъй като е бил използван и тестван от много потребители. Пример за утвърдени продукти - Linux, Apache и Sendmail.
- Намалява шанса за проблем наречен „Заклучване на доставчика“: Ако използвате широка гама от публично недостъпен софтуер от даден доставчик, то може да бъдете ограничени до използване само на техните продукти. При евентуална смяна на доставчика това обикновено включва значителни разходи.
- Наличност на външна поддръжка: Широко използваните софтуерни продукти с отворен код биват поддържани от техния създател или създалото се общество около този продукт. Това включва примерни поддръжки за евентуални уязвимости.

Докато отворения код направи разработването на софтуер по-достъпно и е допринесло изключително за растежа на разработката на софтуер, широкото му използване се счита от мнозина за отрицателно. Това се дължи на липсата на регулации, което може да отвори вратата за множество правни въпроси и спорове. Освен това, определянето кое трябва да бъде софтуер с отворен код и какво да бъде софтуер с затворен код остава трудна и горещо дискутирана тема и до днес.

1.5 Лиценз

Когато работим върху творческо произведение (като писане, графика или код), това произведение е под изключителни авторски права по подразбиране. Тоест законът

предполага, че като автори на работата си, ние имаме думата какво е позволено на останалите да правят с нея. Като цяло това означава, че никой друг не може да използва, копира, разпространява или модифицира вашата работа, без да е изложен на риск от съдебни спорове или сваляне в случай на разпрострени на авторската творба.

Софтуерът с отворен код обаче е по-различно обстоятелство, тъй като авторът очаква други хора да използват, променят и споделят неговото творчество. Законът по подразбиране все още предпазва изключително авторското право, поради което има нужда от лиценз, който изрично да посочва и разрешава неговото публично използване и разпространение. Ако не използвате лиценз с отворен код, това означава, че никой не може да използва, копира, разпространява или променя дадения софтуер. Разработчиците може да смятат, че техните публични проекти в GitHub не се нуждаят от лиценз, но нелицензираният код всъщност е по-ограничен с авторски права, отколкото ако беше правилно обозначен като „отворен код“.

Цялостната култура на GitHub е култура на откритост. Публикувайте това, което искате да публикувате, разклонете (fork²⁶) това, което искате да разклоните, и намерете нови възможности за сътрудничество. Звучи направо утопично. Реалността винаги се прокрадва някъде и фактът, че 77% от проектите на GitHub нямат лиценз [6], не е непременно движение към споделяне и свободен поток на идеи. Както обяснява новият сайт на GitHub, ако проектът няма лиценз, той всъщност може да има по-голяма правна защита съгласно федералните закони за авторско право, отколкото код, който е определен със специфични условия за използване. Никой не е задължен да избира лиценз и е в цялото си право да не го включва в кода или проекта си.

Всички платформи, които съхраняват софтуерни продукти с отворен код, като например Github, имат три основни типа софтуерни лицензи:

- Лицензът на MIT²⁷, който е предназначен да бъде изключително ясен и отворен. Позволява на потребителите да правят каквото и да било с даден проект, стига да кредитират разработчика и да не го държат отговорен за използването на проекта. Този лиценз позволява на лицето да получи копие на софтуера и свързаните с него файлове с правата да използва, модифицира, обединява, разпространява, публикува, подлицензира и продава копия на софтуера.
- Лицензът Apache²⁸, който е подобен на лиценза на MIT, но също така изрично предоставя патентни права на потребителите. Този лиценз включва ясно определение относно как се получава лиценз, авторски права, споделяне, принос и различните ограничения за използването на този лиценз.

²⁶ **Fork** = Разклоняването е да вземете изходния код от софтуерна програма с отворен код и да разработите изцяло нова програма. Софтуерните разклонения могат да бъдат противоречиви, но повечето разработчици са съгласни, че правото на разклонение е най-голямата сила на софтуера с отворен код.

²⁷ **MIT** = MIT означава Масачузетски технологичен институт.

²⁸ **Apache** = Фондация Апачи софтуер (на английски: Apache Software Foundation) осигурява „поддръжка на чадъра“ за проекти с отворен код на Апачи. Вж. Повече: <https://www.apache.org/licenses/LICENSE-2.0>



Фиг. 14, Логото на Фондация Апачи софтуер


- Лиценз GPL²⁹, който е по-стар, по-ограничаващ и по-малко популярен от другите два. Това е лиценз, който изисква от потребителите да следят промените си, ако след това разпространяват проекта. Различните версии на този лиценз също ограничават използването на модифициран код в различни класове хардуер.

Други видове лицензи, които са по-рядко и не толкова широко използвани са:

- Affero GPL
- Artistic License 2.0
- BSD 3-клауза лиценз
- BSD 2-клауза лиценз
- Eclipse Public License v1.0
- GPL v3
- LGPL v2.1
- LGPL v3
- Обществен лиценз на Mozilla версия 2.0
- Обществено достояние (без лиценз)

Лицензът, който използвам аз за платформата за споделен екран е Apache License Version 2.0, January 2004.

²⁹ **GPL** = GNU General Public License (на български превеждан като „Общ публичен лиценз на GNU“, е лиценз, издаден от Фондацията за свободен софтуер, с цел той да бъде използван за лицензирането на софтуер като „свободен“.

 George221b/screenshare-in-the-web-signalr is licensed under the **Apache License 2.0**

A permissive license whose main conditions require preservation of copyright and license notices. Contributors provide an express grant of patent rights. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Permissions	Limitations	Conditions
✓ Commercial use	✗ Trademark use	ⓘ License and copyright notice
✓ Modification	✗ Liability	ⓘ State changes
✓ Distribution	✗ Warranty	
✓ Patent use		
✓ Private use		

This is not legal advice. [Learn more about repository licenses.](#)

Фиг. 15 Лиценз на платформата за споделен екран Apache License 2.0

Този лиценз може да бъде публично достъпен, прочетен и обстойно разгледан на следната интернет връзка:

<https://github.com/George221b/screenshare-in-the-web-signalr/blob/main/LICENSE>

Глава 2. Избор и обосновка на използваните технологии

2.1 C# и .NET

C# е съвременен обектно-ориентиран език за програмиране с общо предназначение, създаден и развиван от Microsoft редом с .NET платформата. На езика C# и върху .NET платформата се разработва изключително разнообразен софтуер: офис приложения, уеб приложения и уеб сайтове, настолни приложения, богати на функционалност мултимедийни Интернет приложения (RIA), приложения за мобилни телефони, игри и много други.

C# е език от високо ниво, който прилича на Java и C++ и донякъде на езици като Delphi, VB.NET и C. Всички C# програми са обектно-ориентирани. Те представляват съвкупност от дефиниции на класове, които съдържат в себе си методи, а в методите е разположена програмната логика – инструкциите, които компютърът изпълнява. [5]

През януари 1999 г. Андерс Хейлсберг сформира екип за изграждане на нов език, наричан по това време Cool, което означава "C-подобен обектно-ориентиран език". Microsoft обмисляше да запази името Cool като окончателно име на езика, но избра да не го прави поради съображения за запазена марка. По времето, когато проектът .NET беше публично обявен на конференцията на професионалните разработчици през юли 2000 г., езикът беше преименуван на C#. Името "C sharp" е вдъхновено от музикалната нотация, където символът #, показва, че писмената нота трябва да бъде полутон по-висока. Много хора го тълкуват като C++++, като език надграждащ и базиран на C++, тъй като два плюса в програмирането стоят за инкрементиране на променлива с едно. Хейлсберг е главният дизайнер и водещ архитект на C# в Microsoft, а преди това е участвал в дизайна на Turbo Pascal, Embarcadero Delphi (бивш CodeGear Delphi, Inprise Delphi и Borland Delphi) и Visual J++. В интервюта и технически документи той заявява, че недостатъците в повечето основни езици за програмиране (напр. C++, Java, Delphi и Smalltalk) са довели до основите на Common Language Runtime (CLR), което от своя страна е довело до дизайна на езика C#.



Фиг. 16 Логото на C# и .NET

Езикът C# се разпространява заедно със специална среда, върху която се изпълнява, наречена Common Language Runtime (CLR). Тази среда е част от платформата .NET Framework, която включва CLR, пакет от стандартни библиотеки, предоставящи базова функционалност, компилатори, дебъгери и други средства за разработка. Благодарение на нея CLR програмите са преносими и след като веднъж бъдат написани, могат да работят почти без промени върху различни хардуерни платформи и операционни системи.

Езикът C# не се разпространява самостоятелно, а е част от платформата Microsoft .NET Framework. .NET Framework най-общо представлява среда за разработка и изпълнение на програми, написани на езика C# или друг език, съвместим с .NET (като VB.NET, Managed C++, J# или F#). Тя се състои от .NET езици за програмиране (C#, VB.NET и други), среда за изпълнение на управляван код (CLR), която изпълнява контролирано C# програмите, и от съвкупност от стандартни библиотеки и инструменти за разработка, като например компилаторът csc, който превръща C# програмите в разбираем за CLR междинен код (наречен MSIL) и библиотеката ADO.NET, която осигурява достъп до бази от данни (например MS SQL Server или MySQL).

Като език за програмиране с общо предназначение, можете да използвате C#, за да разработите почти всичко, за което се сетите, от мобилни и настолни приложения до корпоративен софтуер и облачни платформи. Въпреки това, C# блести най-много, когато го използвате за 3 конкретни типа проекти:

- **Уеб разработка** - Като част от платформата .NET, C# е естествено подходящ за изграждане на динамични уебсайтове и приложения. Обектно-ориентираната му природа го прави идеален за разработване на уебсайтове, които се отличават с висока ефективност и лесна мащабируемост.
- **Windows приложения** - Тъй като C# е разработен от Microsoft, съвсем естествено е той да се използва широко за изграждане на настолни приложения, чиято основна цел е да работят на операционна система Windows. Всъщност това е най-добрият програмен език за тази цел - създаване на приложения, пригодени специално за архитектурата на операционната система на Microsoft.
- **Разработване на игри** - C# е широко признат като един от най-добрите езици за разработване на игри, особено игри базирани на междуплатформения игрови двигател Unity. C# се интегрира с Unity двигателя, за да осигури най-добрата среда за разработка на мобилни игри и дори можете да го използвате за разработване на конзолни игри с кросплатформени технологии като Xamarin.

C# е един от най-широко използваните езици за програмиране днес и постоянно се нарежда сред водещите езици в индекса TIOBE³⁰ и проучването за разработчици на Stack Overflow³¹. Причината за това може да се намери в случаите в които се използва C#, а и също така, че е задвижван и подържан от Microsoft и имайки тясна връзка

³⁰ **TIOBE** = Създаден от TIOBE Company. Името му е аббревиатура от „The Importance of Being Earnest“ - името на комедиен спектакъл от Оскар Уайлд. Самият индекс изчислява резултатите от няколко интернет търсачки, като проверява в заявките, които са отправени към тях, за думи свързани с програмни езици. Препратка към него: <https://www.tiobe.com/tiobe-index/>

³¹ **проучването за разработчици на Stack Overflow** = линк към проучването <https://insights.stackoverflow.com/survey/2021#most-popular-technologies>

с .NET, C# продължава да бъде изключително подходящ език за повечето инженери. Много разработчици на софтуер избират да научат C#, защото това може да подобри кариерата им. Универсалността и силата на езика карат много компании по света да търсят C# таланти, поради което толкова много инженери в крайна сметка го учат. Нещо повече, въпреки че C# съществува от много години на пазара, броят на разработчиците, които го използват, изглежда не намалява. Причината за това е, че компаниите, които наемат C# инженери, искат професионалисти, които могат да работят в множество проекти, а гъвкавостта на езика осигурява точно тази способност.

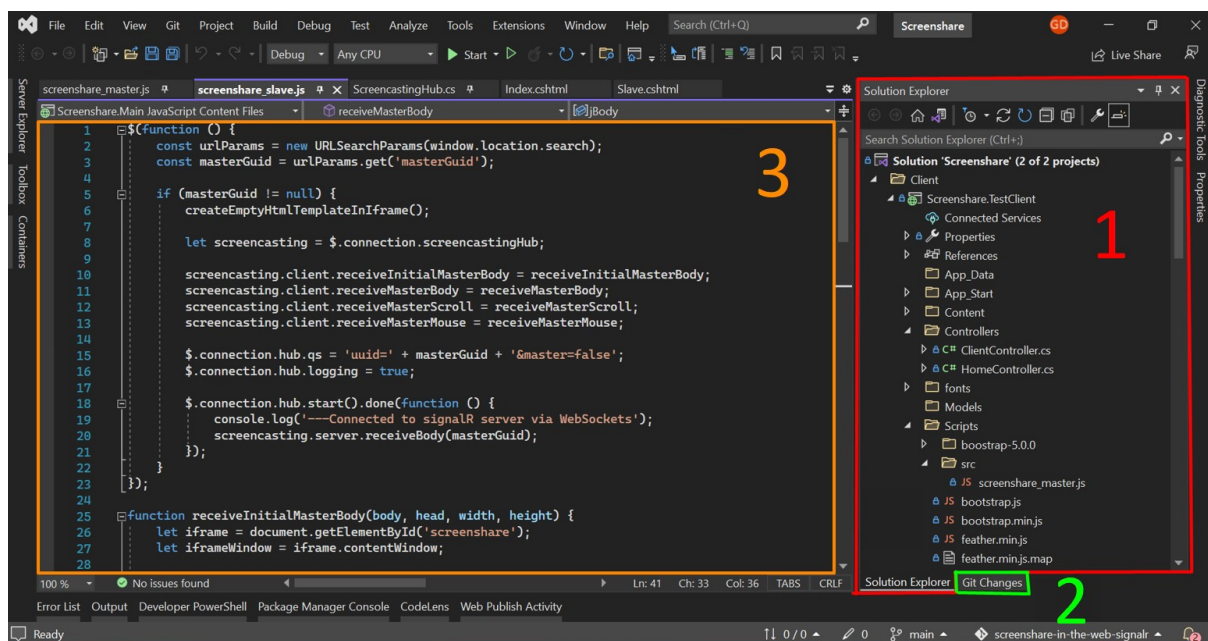
Универсалността може да е най-важната характеристика на C#, но има много други предимства за всеки, който работи с него. Някои от най-важните включват:

- По-бързо време за разработка - C# има няколко функции, които позволяват на разработчиците да пишат по-бързо, отколкото с други езици. Някои от тези функции включват статично въведен и лесен за четене език, познат синтаксис, който изглежда като разширена версия на Java или C++, огромна библиотека, пълна с функционалност на високо ниво и NuGet ³² мениджър.
- Висока мащабируемост - C# превръща софтуера в надеждни продукти, които могат лесно да се настройват и променят, заради неговата типизация и благодарение на архитектурно заложените рамки, които може да се следват. Това означава, че инженерите могат бързо да правят корекции и да надграждат всяка C# програма, за да разширят нейната функционалност и да поддържат повече потребители.
- Обектно-ориентиран - C# възприе обектно-ориентирано програмиране по такъв начин, че може да е езикът, който го използва най-добре. Обектно ориентираното програмиране (ООП) на C# го прави високоефективен и изключително гъвкав, като всичко това прави разработката по-лесна и отнема по-малко ресурси.
- Лесен за научване, труден за усъвършенстване - Като език на високо ниво, C# е много лесен за научаване и разбиране. И това е без да се вземат предвид многото вградени функции, които са много лесни за използване. Нещо повече, всеки инженер, който вече знае C++ или Java, ще се почувства като у дома си, когато за първи път използва C#, тъй като тези езици споделят много от същите функции и цялостен подход към програмирането.
- Голяма общност - C# е един от най-широко използваните езици в света, което означава, че има много C# разработчици, готови да ви помогнат с него. Като продукт на Microsoft, C# има поддръжката на един от най-големите технологични гиганти, което се изразява в експертна помощ, допълнителни ресурси и чести актуализации.

³² **NuGet** =мениджър на пакети, предназначен да даде възможност на разработчиците да споделят код за многократна употреба.

2.2 Microsoft Visual Studio 2019 & 2022

Интегрирана среда за разработка (IDE³³) е програма, богата на функции, която поддържа много различни аспекти за успешна разработка на софтуер. Microsoft Visual Studio IDE е креативна стартова площадка, която можете да използвате за редактиране, отстраняване на грешки (дебъгване) и изграждане на код и след това публикуване на приложение. Освен стандартния редактор и инструмент за отстраняване на грешки, които предоставят повечето интегрирани среди за разработки, Visual Studio включва компилатори, инструменти за завършване/дописване на код, графични дизайнери и много други функции за подобряване на процеса на разработка на софтуер.



Фиг. 17 Отворен проект (платформата за споделен екран) във Visual Studio 2022 и основните му функционалности/менюта.

Фигура 17 показва основните менюта и само една малка част от многобройните различни прозорци, които предлага Visual Studio. За улеснение съм оцветил основните менюта в различни цветове и също така са номерирани:

1. В “Solution Explorer”, горе вдясно, можете да преглеждате, навигирате и управлявате вашите кодови файлове. Solution Explorer може да ви помогне да организирате вашия код, като групира файловете в решения и проекти.

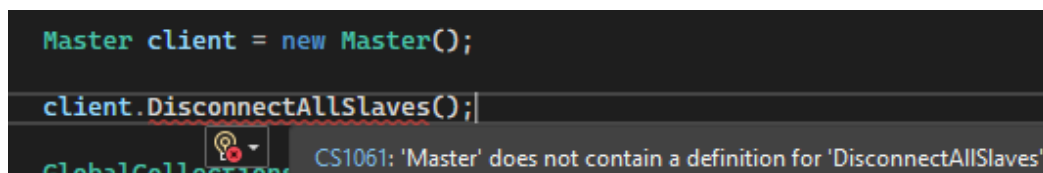
³³ IDE = Integrated development environment = Интегрирана среда за разработка

2. В “Git Changes” долу вдясно можете да проследявате работни елементи и да споделяте код с други, като използвате технологии за контрол на версиите като Git и GitHub.
3. Централният прозорец на редактора, където вероятно ще прекарвате по-голямата част от времето си, показва съдържанието на файла. В прозореца на редактора можете да редактирате код или да проектирате потребителски интерфейс като прозорец с бутони и текстови полета.

Visual Studio се предлага за Windows и Mac. Visual Studio за Mac има много от същите функции като Visual Studio за Windows и е оптимизиран за разработване на междуплатформени и мобилни приложения. Има три издания на Visual Studio: Community, Professional и Enterprise. За разработката на платформата за споделен екран съм използвал Visual Studio 2019 & 2022 Community версиите.

Някои популярни функции във Visual Studio, които подобряват вашата производителност при разработване на софтуер, включват:

- Извивки и бързи действия:



Фиг. 18 Функциите “Squiggles” (извивки) и бързи действия във Visual Studio

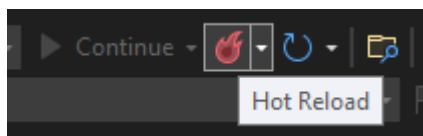
Извивките са вълнообразни подчертавания, които ви предупреждават за грешки или потенциални проблеми във вашия код, докато пишете. Тези визуални следи ви помагат да отстраните проблемите незабавно, без да чакате да откриете грешки по време на изграждане или изпълнение. Ако задържите курсора на мишката върху извивка, ще видите повече информация за грешката. В лявото поле може да се появи и крушка, показваща бързи действия, които можете да предприемете, за да коригирате грешката.

- Почистване на кода - с натискането на един бутон можете да форматирате кода си и да приложите всякакви корекции на кода, предложени от настройките на стила на кода, конвенциите и анализаторите на “Roslyn³⁴”. Почистването на кода, което понастоящем е достъпно само за C# код, ви помага да разрешите проблеми във вашия код.
- Рефакторинг - Рефакторингът включва операции като интелигентно преименуване на променливи, извличане на един или повече реда код в нов метод и промяна на реда на параметрите на метода.

³⁴ **Roslyn** = .NET Compiler Platform, известна още с кодовото си име Roslyn, е набор от компилатори с отворен код и API за анализ на код за езици C# и Visual Basic (VB.NET) от Microsoft.

- “IntelliSense³⁵” е набор от функции, които показват информация за вашия код директно в редактора и в някои случаи пишат малки битове код вместо вас. Това е като да имате вградена основна документация в редактора, така че не е нужно да търсите информация за типа другаде.
- Търсене в Visual Studio - Менютата, опциите и свойствата на Visual Studio понякога могат да изглеждат непосилни. Търсенето в Visual Studio или Ctrl+Q е чудесен начин за бързо намиране на функции и код в IDE на едно място.
- Споделяне на живо - редактирайте съвместно и отстранявайте грешки с други хора в реално време, независимо от типа на приложението или езика за програмиране. Можете незабавно и сигурно да споделите своя проект. Можете също да споделяте сесии за отстраняване на грешки, терминални екземпляри, локални уеб приложения, гласови повиквания и др. Доста тематична функционалност подобна на платформата за споделен екран.

Както споменах малко по-горе за написването на платформата за споделен екран съм използвал две версии за Visual Studio 2019 и 2022, като причината за това е, че по време на разработката на софтуерната част излезе версия за проба (“preview”) на Visual Studio 2022. Останах много силно впечатлен и доволен от най-новата версия на Visual Studio и специално една нейна функция, която използвах много по време на разработката, наречена топло зареждане (“Hot Reload”).



Фиг. 19 Функцията топло зареждане във Visual Studio 2022 (“Hot Reload”)

Започвайки от Visual Studio 2022, независимо от типа приложение, върху което работите, намерението на Hot Reload е да ви спести възможно най-много рестартирания на приложения между редакциите, което ви прави по-продуктивни чрез намаляване на времето, което прекарвате в чакане на приложенията да се компилира, рестартира, стартира и след това да навигирате до предишното местоположение, където сте били в самото приложение.

2.3 SignalR

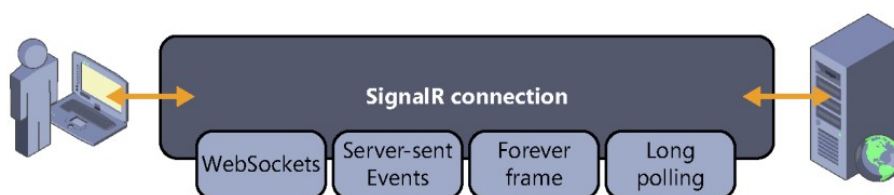
SignalR е рамка, която улеснява изграждането на интерактивни, многопотребителски и уеб приложения в реално време, използвайки набор от много асинхронни техники за постигане на оптимална скорост и максимална ефективност. Първоначално това е бил личен проект на Дейвид Фаулър и Дамян Едуардс, членове на ASP.NET екипа на Microsoft, но вече е официално интегриран продукт в стека от уеб технологии.

³⁵ **IntelliSense** = Интелигентното завършване на код(IntelliSense) е функция за завършване на код, в някои програмни среди, като е съобразена с контекста. Ускорява процеса на кодиране на приложения чрез намаляване на печатните грешки и други често срещани грешки.

Както в случая с много от тези технологии, продуктът е с напълно отворен код (Apache 2.0 лиценз), но с предимствата на пълната подкрепа и поддръжка на базирания в Редмънд гигант. Развитието му може да се проследи и дори да се допринесе за него в GitHub, където може да се намери първичния код на рамката и всички свързани проекти с него.

SignalR ни изолира от всички детайли на ниско ниво, оставяйки ни с впечатлението, че работим върху постоянно отворена връзка между клиента и сървър, което най-често от случаите е вярно. За да постигне това, SignalR включва компоненти, специфични и налични за двата края на комуникацията, което ще улесни доставката и приемането на съобщения в реално време. По начин, който остава скрит за разработчика, SignalR отговаря за определянето на коя е най-добрата техника, налична както на клиента, така и на сървър, за създаване на постоянно отворена връзка между клиента и сървър. Различните видове техники:

- WebSocket (уебсокет): Това е единственият транспорт от изброените, който установява истински постоянна, двупосочна връзка между клиент и сървър. Базирана на различен протокол от стандартния HTTP, наречен WS (WebSocket).
- Изпратени от сървър събития/Източник на събитие (Server Sent Events): Този транспортен тип получава изпратени от сървър събития. Той се свързва със сървър чрез HTTP протокола и получава събития в определен MIME³⁶ формат text/event-stream.
- Ajax long polling: Не се създава постоянна връзка. Този транспортен тип прави заявка до сървър, която остава отворена докато сървърът не отговори, след което връзката се нулира.
- Forever Frame: Този вид транспорт може да се използва само когато клиентът е интернет Explorer. Той създава скрит iframe, който от своя страна прави заявки до сървър.



Фиг. 20 Различните видове свързвания, които SignalR прави според наличното на сървър и клиент.

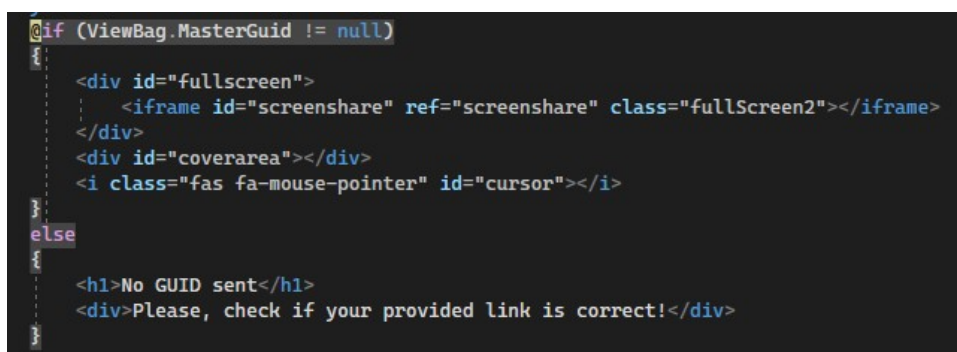
Освен че, SignalR се грижи да създаде основната връзка и да я поддържа непрекъснато отворена, то той също автоматично управлява прекъсванията и опитите за повторно свързване. Както е показано на фигура 20, ще видим, че се използва само една постоянно отворена връзка, а SignalR ще се погрижи за останалото, правейки всичко да функционира.

³⁶ **MIME** = Multipurpose Internet Mail Extensions (MIME) Типът медия показва естеството и формата на документ, файл или набор от байтове. Типовете MIME са дефинирани и стандартизирани в RFC 6838 на IETF

2.4 Razor View Engine

Един от трите компонента в приложението MVC е изгледът (View). Той е отговорен за предоставянето на потребителския интерфейс (UI) към потребителя. Изгледът показва данни от модела на потребителя и също така позволява на тези данни да бъдат променяни. За да се визуализира успешно съдържанието на дадена страница, то браузъра трябва да получи HTML. View Engine обикновено съчетава език за програмиране и HTML, като по-късно цялото съдържание се преобразува до чист HTML.

Razor изглежда съчетава C# код и HTML, като дадения файл, който съдържа изгледа е с разширение .cshtml. Цялото съдържание от този тип файл се преобразува, като това се случва на сървъра. C# кода може да е почти всякакъв с малко лимитации, което позволява огромна гъвкавост при изграждане на съдържание. Също така се спестява много време, когато имаме повторяеми парчета от HTML код. Платформата за споделен екран използва Razor View Engine, като на фигура 21 демонстрира как точно изглежда.



```
@if (ViewBag.MasterGuid != null)
{
    <div id="fullscreen">
        <iframe id="screenshare" ref="screenshare" class="fullScreen2"></iframe>
    </div>
    <div id="coverarea"></div>
    <i class="fas fa-mouse-pointer" id="cursor"></i>
}
else
{
    <h1>No GUID sent</h1>
    <div>Please, check if your provided link is correct!</div>
}
```

Фиг. 21 Пример за Razor View от платформата за споделен екран

За превключване от HTML в C# код се използва символа @, след който вече може да пишем нашия код на C#.

2.5 JavaScript & jQuery

JavaScript е динамичен език за компютърно програмиране. Той е лек и най-обикновено се използва като част от уеб страници, чиито имплементации позволяват клиентската страна да взаимодейства с потребителя и така да се създават динамични страници. Този език е скриптов с обектно-ориентирани възможности. JavaScript първо е бил известен като LiveScript, но Netscape³⁷ променя името му на JavaScript, вероятно поради вълнението, генерирано от широк кръг Java ентусиасти. JavaScript се появи за първи път в Netscape 2.0 през 1995 г. с името LiveScript. Ядрото на езика с общо

³⁷ **Netscape** = Американска частна фирма занимаваща се в секторите: интернет, софтуерни продукти и телекомуникации.

предназначение е вградено в Netscape, Internet Explorer и други браузъри. Спецификацията ECMA-262 дефинира стандартна версия на основния JavaScript език.

- JavaScript е лек, скриптов език за програмиране.
- Проектиран за създаване на приложения, които се изпълняват в брауъра.
- Допълващ и интегриран с почти всички програмни езици.
- Допълващ и интегриран с HTML.
- Отворен и междуплатформен.

JavaScript, който се изпълнява от страна на клиента, като например в брауър, е най-често срещаната форма на езика. Скриптовия файл съдържащ езика трябва да бъде включен или рефериран от HTML документ, за да може кодът да се интерпретира от брауъра. Това означава, че една HTML базирана уеб страница не е нужно вече да бъде статична, тъй като вече може да включва програми, които да взаимодействат с потребителя, контролират брауъра или динамично да създават HTML съдържание. Например, можете да използвате JavaScript, за да проверите дали потребителят е въвел валиден имейл адрес в полето на HTML форма. JavaScript може да се изпълни, когато потребителят изпрати формуляра и само ако всички записи в дадената форма са валидни, то само тогава ще бъдат изпратени до уеб сървъра. JavaScript може да се използва за улавяне на събития, инициирани от потребителя, като щраквания върху бутони, оразмеряване на страницата и други действия, които потребителят инициира изрично или косвено. В глава трета, ще демонстрирам как реално с помощта на този език се закачае за почти всички потребителски събития. Предимствата на използването на JavaScript са:

- По-малко взаимодействие със сървъра: Можете да потвърдите въведеното от потребителя, преди да изпратите страницата към сървъра. Това спестява трафик на сървъра, което означава по-малко натоварване на вашия сървър. Незабавна обратна връзка с посетителите: Не е нужно потребителите да изчакват страницата да презареди, за да видят дали не са забравили да въведат нещо.
- Повишена интерактивност: Можете да създавате интерфейси, които реагират, когато потребителят задържа над тях с мишката или ги активира чрез клавиатурата.
- По-богати интерфейси: Можете да използвате JavaScript, за да включите елементи като компоненти с плъзгане, пускане и плъзгачи, за да предоставите богат потребителски интерфейс на посетителите на вашия сайт.

Едно от основните предимства на JavaScript е, че не изисква скъпи инструменти за разработка. Можете да започнете с прост текстов редактор като Notepad³⁸. Тъй като това е интерпретиран език в контекста на уеб брауъра, като дори не се нуждаете от покупката на компилатор.

jQuery е бърза и малка JavaScript библиотека, създадена от Джон Резиг през 2006 г. с хубаво мото: Пишете по-малко, правете повече. jQuery опростява обхождането на HTML елементи и тяхното намиране в документа, обработката на събития, анимации и

³⁸ **Notepad** = представлява прост текстов редактор за Microsoft Windows. Включен е във всички версии на Microsoft Windows (от Windows 1.0 от 1985 г.).

Ajax³⁹ взаимодействия за бързо уеб развитие. jQuery е помощна библиотека за JavaScript, предназначена да опрости различни видове често срещани задачи чрез писане на по-малко код. Ето списък с важните функции, поддържани от jQuery:

- Манипулиране на DOM: jQuery улеснява избирането на DOM елементи, търсенето им и модифицирането на съдържанието им чрез използване на мултиплатформена технология с отворен код наречена Sizzle.
- Обработка на събития: jQuery предлага елегантен начин за улавяне на голямо разнообразие от събития, като например кликване на върху връзка, без да е необходимо да претрупвате самия HTML кода с манипулатори на събития.
- Поддръжка на AJAX: jQuery ви помага много да разработите само оразмеряващ се и богат на функции сайт, използващ технологията AJAX.
- Анимации: jQuery идва с много вградени анимационни ефекти, които можете да използвате във вашите уебсайтове.
- Лека библиотека: jQuery е много лека библиотека – с размер около 19 KB (минимизирана).
- Поддръжка на различни браузъри: jQuery има поддръжка на различни браузъри и работи добре с IE 6.0+, FF 2.0+, Safari 3.0+, Chrome и Opera 9.0+
- Утвърдена и развиваща се технология: jQuery поддържа CSS3 селектори.

2.6 Github & Sourcetree – GIT GUI

Том Престън-Вернър, Крис Уонстрат и Пи Джей Хайет се събраха през 2008 г., за да си сътрудничат по проект, който се превръща в GitHub. Само за 10 години техния съвместен продукт променя начина, по който хората пишат код. Той не просто прави програмирането по-лесно, но и променя начина, по който разработчиците на софтуер мислят за програмирането. GitHub постига невероятен растеж и успех, като идентифицира основен проблем, с който се борят милиони хора по целия свят - как хората да пишат съвместно и едновременно код, също така дава елегантно решение, от което пазарът отчаяно имаше нужда. Чрез изграждането на SaaS⁴⁰ услуга около Git, GitHub успява да достави стойност и да монетизира тази екосистема с отворен код. Това е основната причина GitHub да стане привлекателна придобивка за Microsoft в началото на юни 2018 г., въпреки проблемната история на Microsoft в общността с отворен код. Сутринта на 4 юни 2018 г. технологичният свят се събуди с невероятната новина, че Microsoft е придобила GitHub за \$7,5 млрд.

GitHub в основата си е хост. Това е пространство за качване на код от всякакви разработчици, за да бъде достъпен всички. Най-често се използва от програмистите за

³⁹ **Ajax** = съкращение на Asynchronous JavaScript and XML. Това е похват в уеб разработките за създаване на интерактивни уеб приложения.

⁴⁰ **SaaS** = Софтуер като услуга (Software as a Service) е модел на доставка на софтуер, при който софтуерът и асоциираните данни са хоствани централно, предимно в (интернет) облак, и са обикновено достъпни за потребителите чрез клиентска програма, например с използване на уеб браузър през интернет.

да работят заедно един с друг върху конкретен софтуер и версия на контрола. GitHub има два ключови принципа:

- Git
- Version control

Git е система за контрол на версиите с отворен код. Позволява на разработчиците да контролират различни версии и да извършват промени с помощта на командния ред. Новодошлите специалисти в сферата свикват трудно с контрол на версиите, което е основната причина защо повечето компании предпочитат да използват GitHub вместо това.

Контролът на версиите (Version Control) позволява на хората да проследяват историята на дадена програма/софтуер. Това работи чрез два метода: branching (разклоняване) и merging (сливане).

- Branching - разклоняването позволява на разработчика да клонира програма, така че той да може да редактира и да разработва кода за себе си, без да засяга или модифицира оригинала.
- Merging - обединяването позволява на разработчика да обедини оригиналния код и техния клон от който са разклонили. Този процес може да бъде върнат, ако е необходимо.

След като уточнихме тези два ключови принципа, можем да кажем, че GitHub предлага удобен потребителски интерфейс, който позволява на разработчиците да използват Git, без да е нужно да научават командите за командния ред. Той позволява софтуера/кода, който е хостнат там да бъде разклоняван и обединяван. Монетизацията на GitHub става, чрез предлагане на платени публично недостъпни пространства за съхранение на проекти.

Най-важните термини, които трябва да знаем, когато използваме GitHub са:

Команден ред - текстов интерфейс, използван за въвеждане на всички видове команди.

- Branching (разклоняване) - създаване на копия на програма за паралелна работа.
- Merging (обединяване) - обединяване на две различни копия на програма в едно.
- Repository (хранилище) - име и местоположение на всички файлове на даден проект. Често съкратено като гиро и също така съдържа цялата история на промените.
- Commit - приготвяне на променени файлове за качване до даден клон.
- Pull request (заявка за изтегляне): заявка за редактиране на клон, управляван от друго лице.
- Push - качване на всички подготвени commit-и до даден клон.
- Fork- клонира хранилище във вашия профил, където може да редактирате всичко.

Sourcetree е безплатен десктоп клиент с графичен потребителски интерфейс (GUI), който опростява начина, по който взаимодействате с хранилищата на Git, така че да можете напълно да се концентрирате върху писането на код. Този GUI улеснява визуализирането и управлението на вашите хранилища. Той също така се интегрира с много лесно с GitHub, както е случая на платформата за споделен екран. В глава 3 ще

демонстрираме как използваме Sourcetree, за да може нашите промени да отидат в хранилището на GitHub.

Продуктът е подходящ както за начинаещи, така и за напреднали потребители, богатият набор от функции на Sourcetree помага на продуктивността. Ето някои негови предимства:

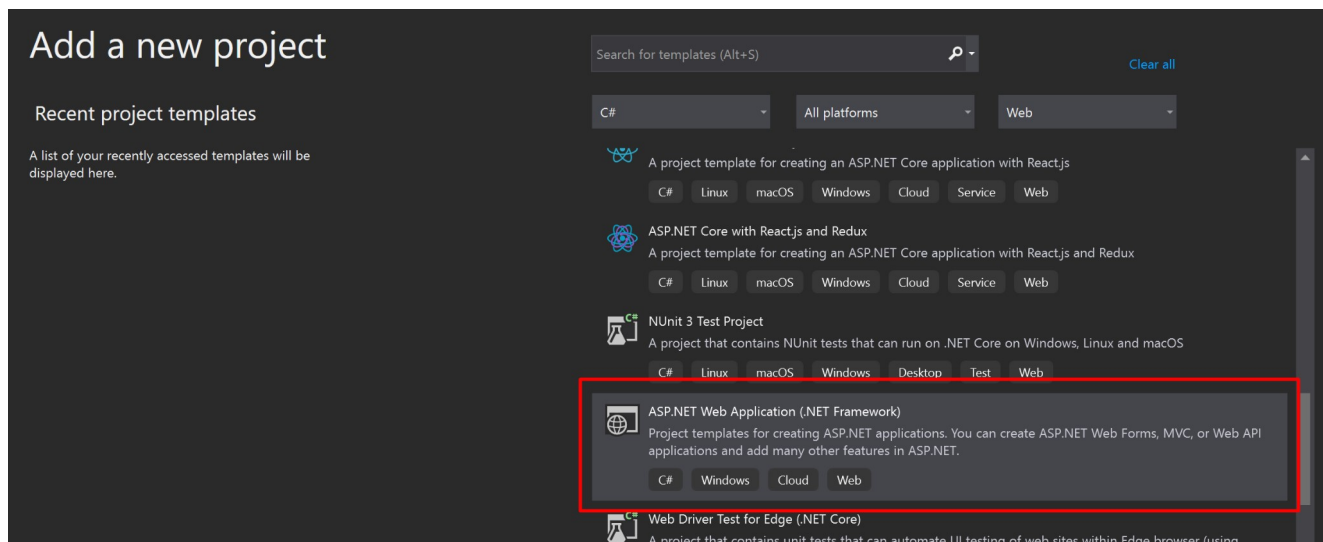
- Използвайте удобен потребителски интерфейс с основните Git команди.
- Управлявайте своите Git хранилища от един клиент (независимо дали хостван или локален)
- Commit, Push, Pull или Merge - всички команди са на едно кликане.
- Свържете своите хранилища с Bitbucket, Stash, Microsoft TFS или GitHub.

Глава 3. Разработване на платформата

В глава трета се разработва платформата за споделен екран, стъпка по стъпка, от начало до край, като за постигането на тази цел още в самия процес на разработката са направени екранни снимки (фигурите приложени в тази глава), за да може целия процес да бъде лесно проследим или повторен. Освен фигурите са приложени и много парчета код, като е важно да се отбележи, че се показват само най-важните части от кода, чрез които цялата платформа се задвижва. Целия проект е няколко хиляди реда код, който е невъзможно да бъде побран в тази дипломна работа, но също така не е и за цел да бъде показано абсолютно всичко. Целта е да се покажат всички задвижващи елементи, благодарение на които споделянето на екран в реално време е възможно. Не са приложени методи, функции и класове, чиято цел е спомагателна, а не основна за самата платформа.

3.1 Структура на платформата, създаване на проектите

Започваме със създаването на основната структура, която показахме в глава 1.3, фигура 10. За тази цел създаваме един Blank Solution, който кръщаваме “Screenshare”. След това създаваме един нов проект от тип ASP.NET Web Application (.NET Framework).



Фиг. 22 Създаване на първия проект в платформата за споделен екран.

На следващия екран даваме правилното име на проекта, в нашия случай “Screenshare.Main”.

Configure your new project

ASP.NET Web Application (.NET Framework) C# Windows Cloud Web

Project name

Screenshare.Main

Location

C:\Users\Georgi\Documents\screenshare-in-the-web-signal\Screenshare\Source\ ...

Framework

.NET Framework 4.8

Фиг. 23 Избор на име на проект и избиране на .NET версия

Като последна стъпка, преди да сме създали проекта е да изберем от какъв подтип да е ASP.NET Web Application. Като тук ние избираме Web API.

Create a new ASP.NET Web Application

Empty
An empty project template for creating ASP.NET applications. This template does not have any content in it.

Web Forms
A project template for creating ASP.NET Web Forms applications. ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

MVC
A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Web API
A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Single Page Application
A project template for creating rich client side JavaScript driven HTML5 applications using ASP.NET Web API. Single Page Applications provide a rich user experience which includes client-side interactions using HTML5, CSS3, and JavaScript.

Authentication
No Authentication
[Change](#)

Add folders & core references

- ☐ Web Forms
- ☒ MVC
- ☒ Web API

Advanced

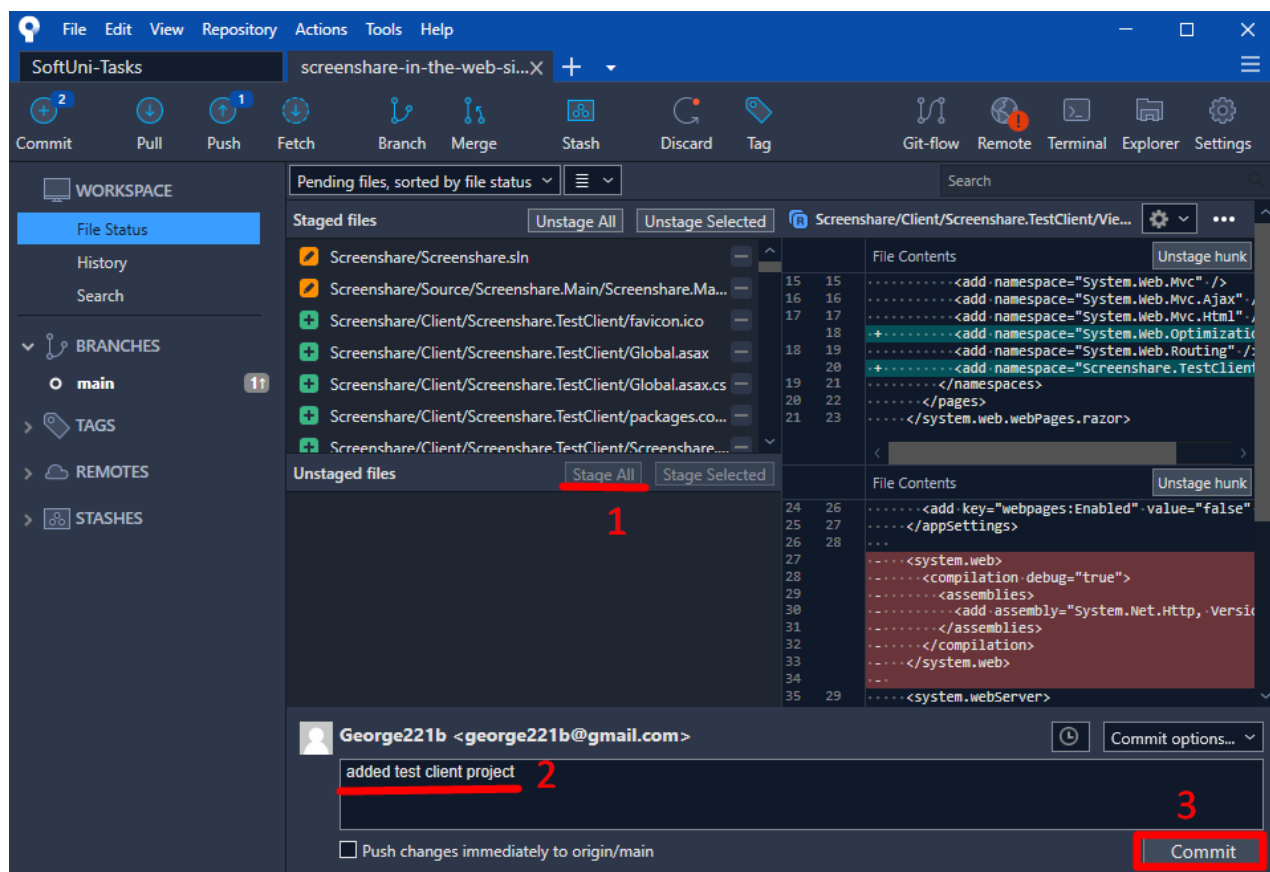
- ☒ Configure for HTTPS
- ☐ Docker support
(Requires [Docker Desktop](#))
- ☐ Also create a project for unit tests
Screenshare.Main.Tests

Back Create

Фиг. 24 Избор на подтип на проекта - Web API

След това може да повтаряме същото упражнение, но този път създаваме проект на име “Screenshare.TestClient” в новосъздадена папка “Client”. По този начин вече трябва да имаме същата структура от проекти, която сме демонстрирали във фигура 10.

В програмирането е важно всичко да се прави на стъпки и етапи, за да имаме винаги работеща версия и контрол между всяка стъпка. В момента сме създали празни проекти, които ще са основата на платформата за споделен екран. Това, което може да се направи е да се пуснат и двата проекта, за да се уверим, че всичко работи до тук и след това да направим първия си commit. По този начин работата ни може много лесно да бъде проследена от други програмисти и също така създаваме една история от промени, по които може да се ориентираме в кой момент какво се е случило. В случай на евентуални промени, които биха “счупили” целия проект, може да се върнем на предна работеща версия на даден commit. Както споменахме в глава 2.6 за версия на контролите използваме Sourcetree и GitHub.



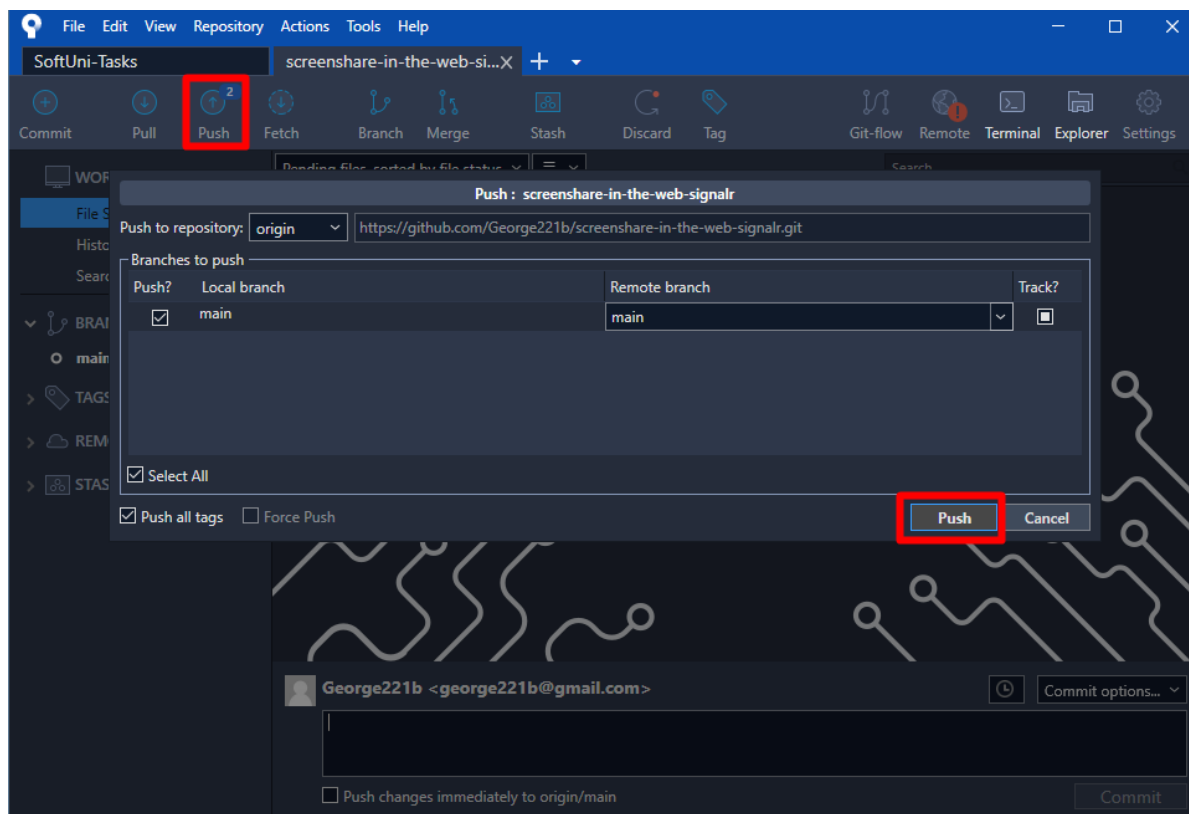
Фиг. 25 Sourcetree и качване на промените до GitHub.

Фигура 25 демонстрира работата ни със Sourcetree, която следва три основни стъпки и принципа на работа с този инструмент остава същия до края. Ето защо ще го демонстрирам само веднъж, а през останалото време правя commit след добавяне на основни компоненти за самата платформа. Трите основни стъпки:

- Stage All - тук избираме кои промени искаме да бъдат част от дадения commit. В общия случай са всички налични, затова избираме Stage All.

- Добавяне на заглавие за текущия commit. Тук избираме подходящо име на нашия commit, който по-късно може да видим и във GitHub.
- Commit - създаваме дадения Commit.

Последната част от работата със SourceTree след направата на commit е, промените да се качат до GitHub профила ни. За да се случи това, трябва да направим командата push, която за наше удобство е на два бутона разстояние. Докато не направим push, промените, които се съдържат в дадения commit, остават локално при нас.



Фиг. 26 Качване на всички Commits до нашето хранилище (Repository).

На следната връзка може да намерим цялата история от промени, които съм направил до пълното изграждане на платформата. (<https://github.com/George221b/screenshare-in-the-web-signalr/commits/main>).

3.2 Основни модели на платформата

В тази под глава ще разгледаме основните модели на платформата за споделен екран, които ще използваме навсякъде, също така ще разгледаме подробно най-важните полета, които по-късно ще се влязат в употреба.

```

1. public abstract class BaseScreenshareUser
2. {
3.     public BaseScreenshareUser()
4.     {
5.         this.GUID = Guid.NewGuid().ToString();
6.     }

```

```

7.
8.     public string ConnectionID { get; set; }
9.     public string GUID { get; set; }
10.    public string Title { get; set; }
11.    public string Head { get; set; }
12.    public string Body { get; set; }
13.    public int Width { get; set; }
14.    public int Height { get; set; }
15.    public int CursorTop { get; set; }
16.    public int CursorLeft { get; set; }
17.    public int ScrollTop { get; set; }
18.    public int ScrollLeft { get; set; }
19. }

```

“BaseScreenshareUser” класът има следните полета:

- “**ConnectionID**”: Много важно поле, чиято цел е, при установяване на връзка със SignalR да запазим стойност в него, която е уникална и автоматично генерирана от SignalR. Запазването на това поле се случва в метода *OnConnected()*, който ще разгледаме в глава 3.4.3. При евентуална прекъсната връзка от SignalR, това поле бива изтрито. Така знаем колко отворени връзки имаме по всяко време, както и на кого принадлежат.
- “**GUID**”: уникална генерирана стойност, използване в backend API за разграничаване на потребителите.
- “**Head**”, “**Body**”: това е чистия HTML на даден клиент (master). Като Body, съдържа стойността на *<body>* елемента, а Head на *<head>* елемента.
- “**Width**”, “**Height**”: Стойности на размера на страницата на даден клиент. Съответно дължина и ширина. Тъй като платформата за споделен екран е pixel perfect, са ни нужните точните размери.
- “**CursorTop**”, “**CursorLeft**”: Къде е текущата локация на курсора на даден клиент.
- “**ScrollTop**”, “**ScrollLeft**”: Колко надалеч в пиксели е скролирал даден клиент, като *ScrollTop* е стойността на вертикалния скрол, а *ScrollLeft* на хоризонталния.

Това е основния модел, от който ще **наследят** останалите два важни класа. В глава 1.1 обяснихме функцията и разликата между клиент/master и потребител/slave. Тук ще създадем съответните класове за това.

```

1. public class Slave : BaseScreenshareUser
2. {
3. }

```

```

1. public class Master : BaseScreenshareUser
2. {
3.     public Master()
4.     {
5.         this.Slaves = new List<Slave>();
6.     }
7.
8.     public ICollection<Slave> Slaves { get; set; }
9. }

```

Важното тук е, че имаме модел Master, който съдържа колекция от Slave модели. Тази връзка, още наричано релация, между двата модела е известна като един към много или, че един клиент, може да има много потребители, но един потребител е част само от един клиент.

3.3 Основен API контролер

Основната цел на един API контролер е да направи връзка между клиент (обикновено уеб браузър) и сървър (уеб сървър). Тази връзка се осъществява благодарение на HTTP протокола (протокол за пренос на хипертекст).

В случая на платформата за споделен екран, ще направим HTTP заявка от frontend частта на приложението (проектът от папка Client) към backend частта (текущия проект, който разработваме - Source/Screenshare.Main). В момента ще разгледаме как се имплементира контролера от сървърна страна, който ще отговаря за успешно получаване на заявката, като мястото от което ще бъде изпратена/изградена ще разгледаме в глава 3.6.

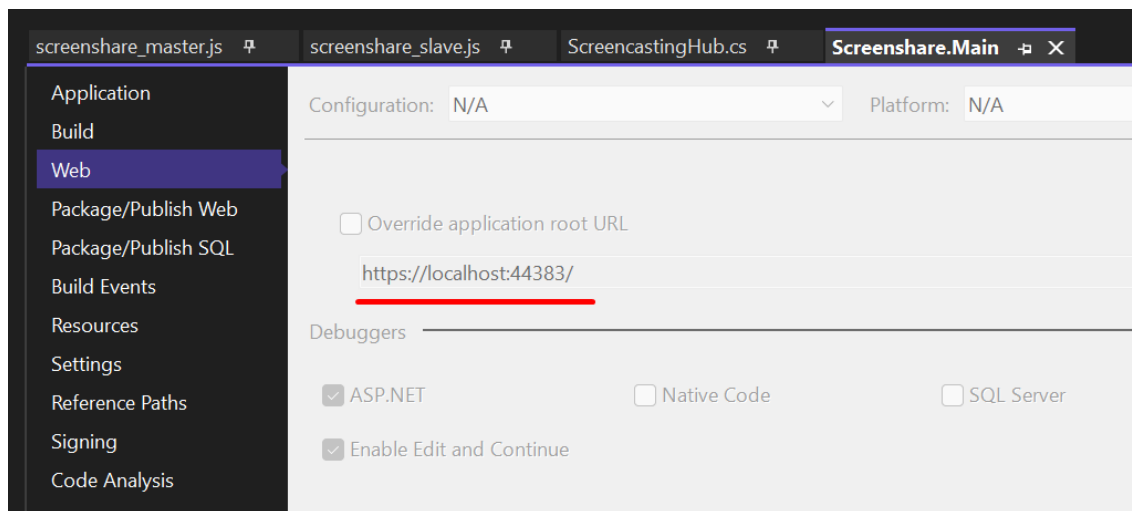
```
1. [RoutePrefix("api/Screencast")]
2. public class ScreencastController : ApiController
3. {
4.     [EnableCors(origins: "*", headers: "*", methods: "*")]
5.     [Route("RequestConnection")]
6.     [HttpPost]
7.     public IHttpActionResult RequestConnection(MasterDataInputModel inputModel)
8.     {
9.         Master client = new Master();
10.
11.         GlobalCollections.users[client.GUID] = client;
12.         string slaveUrl =
13.             HttpContext.Current.Request.Url.AbsoluteUri.Replace(HttpContext.Current.Request.U
14.                 rl.AbsolutePath, string.Empty) + $"/home/slave?masterGuid={client.GUID}";
15.         return Ok(new ScreenshareDetailsOutputModel { GUID = client.GUID, URL =
16.             slaveUrl });
17.     }
18. }
```

Ред едно и ред пет отговарят за изграждането на пълния URL, чрез които се позволява на клиента да изпрати данни на сървъра. Като в нашия случай, пълния URL ще изглежда по следния начин:

HTTP POST {baseUrl}/api/Screencast/RequestConnection

(пълен URL)

Стойността на `baseUrl` е домейна на който тръгва backend приложението, като това може да бъде проверено, чрез `Properties` → `Web`.



Фиг. 27 Проверка на `baseUrl`: Right click backend project -> `Properties` -> `Web`

Ред 4 от фрагмента с код по-горе е важен, понеже за да бъде достъпен ресурс, чрез HTTP заявка, която е изпратена през JavaScript код от различен произход (домейн) от този на дадения ресурс, то това би резултирало в грешка. **CORS**⁴¹ е механизъм за защита, който ограничава как даден скрипт или документ, идващ от даден произход, взаимодейства с ресурс от друг произход. Това е с цел потенциално злонамерни скриптове да бъдат изолирани и така да се намалят възможностите за евентуална атака.

За да бъде позволена заявка от клиентското приложение до нашия сървър се нуждаем от въпросния ред 4. *Enable Cors* със стойност “*origins*” равно на звезда (*), означава, че даваме пълен достъп до нашия контролер, независимо от произхода на клиента. Стойността звезда се счита за лоша практика, понеже все едно изгасяме защитата CORS. Правилния подход за имплементация е да впишем всички произходи, които очакваме да искат достъп до нашия ресурси, тъй като това е тестово приложение, оставям стойността звездичка за по-лесна работа.

Ако разгледаме фрагмента код от ред девет до четиринадесет ще забележим, че това, което се случва е създаване на нов master/клиент, след което той се записва в нашата база (в този случай *in-memory*⁴²), след което връщаме модел попълнен с информация.

3.4 Добавяне на SignalR. Сървърна SignalR част.

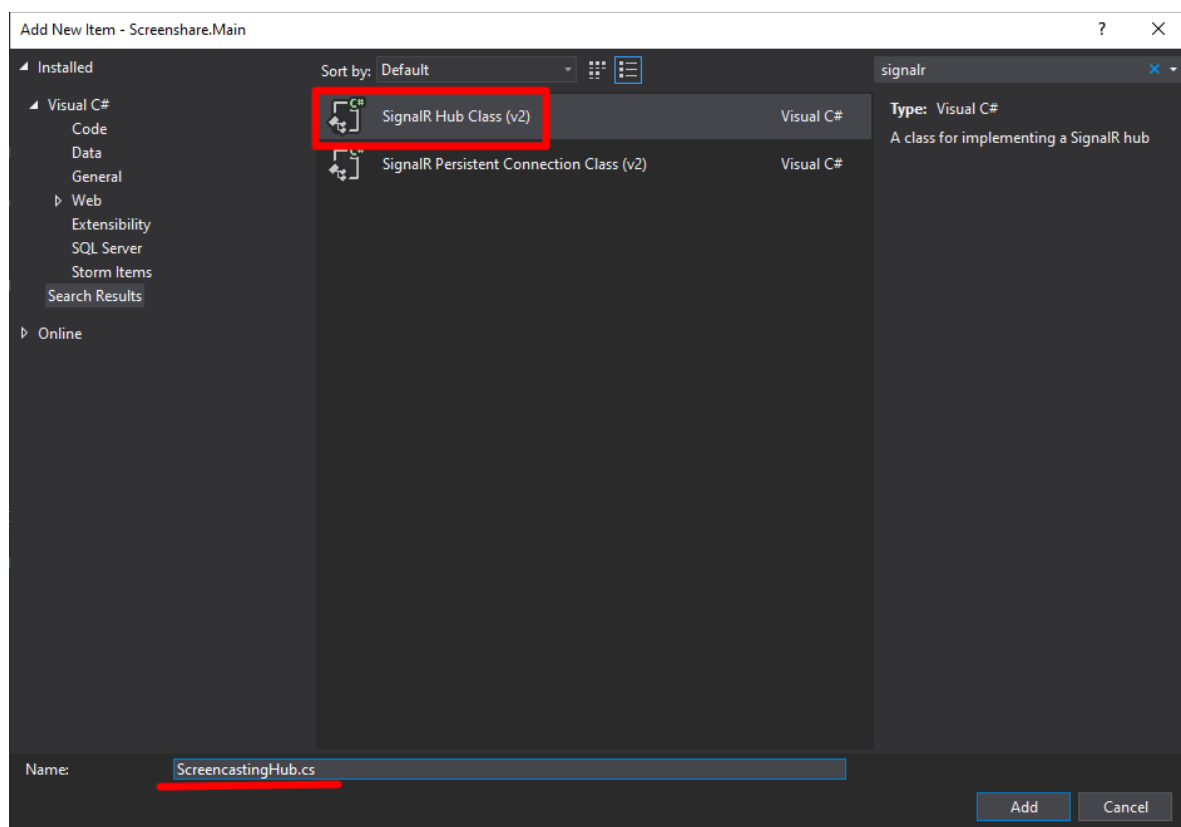
В глава 2.3 разгледахме подробно технологията SignalR как тя работи, функционира и успява да осъществи двупосочна постоянна комуникация между сървър и клиент. В тази глава ще разгледаме как се имплементира .NET SignalR в платформата за споделен екран.

⁴¹ **CORS** = Cross-Origin Resource Sharing - политика със същия произход

⁴² **in-memory** = база данни, съхранявана в оперативната памет.

Има няколко начина SignalR да бъде добавен в нашия проект, но може би най-лесния е следния:

- Създаваме една празна папка в нашия backend проект (*Screenshare.Main*)
- Десен бутон върху новосъздадената папка и следваме следните команди: Add > New Item > Installed > Visual C# > Web > SignalR Hub Class (v2), след това даваме име на нашия Hub клас и добавяме.



Фиг. 28 Добавяне на .NET SignalR Hub Class (v2)

Чрез този процес се добавят всички нужни референции и пакети за работа със SignalR, като обикновено се инсталират последните налични версии, като това може да бъде проверено в NuGet мениджъра.

Последна стъпка преди да разгледаме новосъздадения Hub клас и логиката в него, трябва да добавим OWIN⁴³ Startup клас.

- Add New Item > Installed > Visual C# > Web > OWIN Startup Class > кръщаваме файла Startup > add

```
1. public class Startup
2. {
3.     public void Configuration(IApplicationBuilder app)
4.     {
```

⁴³ **OWIN** = Open Web Interface for .NET, OWIN позволява на уеб приложенията да бъдат отделени от уеб сървърите. Той дефинира стандартен начин за използване на междинен софтуер в конвейер за обработка на заявки и сървърни отговори (request, response).

```

5.         app.Map("/signalr", map =>
6.             {
7.                 map.UseCors(CorsOptions.AllowAll);
8.                 var hubConfiguration = new HubConfiguration
9.                 {
10.
11.                 };
12.
13.                 map.RunSignalR(hubConfiguration);
14.             });
15.
16.             // Uncomment in case of large screenshare body
17.             //GlobalHost.Configuration.MaxIncomingWebSocketMessageSize = null;
18.         }
19.     }

```

Това трябва да е крайния вид на “Startup.cs” файла. Накратко тук се случва самото добавяне на SignalR в нашия проект, като това е мястото на всички конфигурации и настройки. Искам да обърна внимание на няколко важни настройки:

- Ред пети оказва пътя, който трябва да бъде достъпен, чрез WS протокола за да установим връзка, чрез SignalR.
- Ред седми е добавяне на CORS, за които говорехме подробно в частта с Web API, тук имаме пълно сходство.
- Ред десети е мястото на всички настройки и конфигурации, които искаме да направим на SignalR за да отговаря на нашите нужди или сървърни спецификации.
- Ред седемнадесети представлява коментар, който съм добавил. Нека той бъде премахнат в случай, че платформата за споделен екран се използва в страници с голямо количество HTML. Този ред премахва оказва каква големина съдържание може да пренасяме, чрез уебсокет. Като по подразбиране тази стойност е 64 килобайта.

SignalR Hub класът отговаря за всички методи, които могат да бъдат извикани от клиентската част на приложението. Тук имаме връзка, която е постоянно отворена, точно поради тази причина има няколко много важни метода, които ще имплементираме. Тези методи се извикват автоматично, като първия, който ще разгледаме се изпълнява, когато направим успешна връзка между клиент и сървър.

```

1. public class ScreencastingHub : Hub
2. {
3.     public override Task OnConnected()
4.     {
5.         var masterUuid = Context.QueryString["uuid"];
6.         var isMaster = Context.QueryString["master"];
7.
8.         if (GlobalCollections.users.ContainsKey(masterUuid))
9.         {
10.             if (isMaster == "true")
11.             {
12.                 GlobalCollections.users[masterUuid].ConnectionID =
Context.ConnectionId;
13.             }

```

```

14.         else
15.         {
16.             GlobalCollections.users[masterUuid].Slaves.Add(new Slave
17.             {
18.                 ConnectionID = Context.ConnectionId
19.             });
20.         }
21.     }
22.
23.     return base.OnConnected();
24. }
25. }

```

Щом се изпълни метода “**OnConnected**” това означава, че клиентската част (JavaScript) успешно е направила връзката с нашия SignalR Hub. В API Контролера в глава 3.3 се създава master/клиент, а в този метод правим проверка дали текущия потребител, който се е свързал с нашия Hub е master или slave. В случай, че е master ние запазваме “*ConnectionID*”, а ако е slave намираме към кой master иска да се свърже дадения slave и го добавяме в колекцията от потребители на търсения master.

Аналогично на този метод имаме и “**OnDisconnected**” метод, който се изпълнява в случай, че вече имаме установена двупосочна постоянна връзка и тя бива прекъсната по някаква причина. Това може да е при загуба на интернет, затваряне на целия браузър или просто връзката е принудително затворена.

```

1. public override Task OnDisconnected(bool stopCalled)
2. {
3.     var masterUuid = Context.QueryString["uuid"];
4.     var isMaster = Context.QueryString["master"];
5.     if (isMaster == "false")
6.     {
7.         if (GlobalCollections.users.ContainsKey(masterUuid))
8.         {
9.             Master master = GlobalCollections.users[masterUuid];
10.            Slave slaveClient = master.Slaves.Where(c => c.ConnectionID ==
Context.ConnectionId).FirstOrDefault();
11.            if (slaveClient != null)
12.                master.Slaves.Remove(slaveClient);
13.        }
14.    }
15.
16.    return base.OnDisconnected(stopCalled);
17. }

```

Освен тези два метода имаме още един, който се изпълнява автоматично и това е “**OnReconnected**”.

След като сме създали клиент/master в API контролера и след това сме установили връзка, чрез “**OnConnected**” метода, следва да запазим цялата нужна информация от клиента за да може потребителите да виждат неговия екран, без пиксел разлика. За тази цел създаваме метод, който ще приеме всички нужни параметри на даден клиент, а това са неговото HTML тяло, глава, дължина и ширина на екрана.

```

1. public async Task ReceiveBody(string masterGuid)
2. {
3.     if (GlobalCollections.users.ContainsKey(masterGuid))
4.     {
5.         Master client = GlobalCollections.users[masterGuid];
6.
7.         await Clients.Caller.receiveInitialMasterBody(client.Body, client.Head,
            client.Width, client.Height);
8.     }
9. }

```

3.5 Имплементация на Slave/потребител функционалност. SignalR JavaScript част.

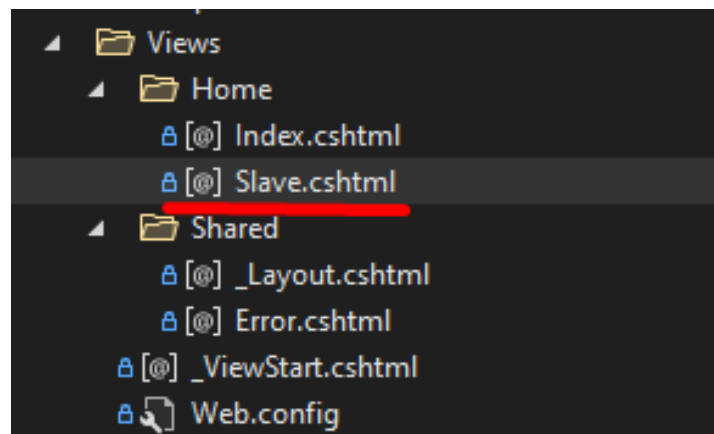
За да имплементирам Slave JavaScript функционалността ще следвам MVC архитектурата, като за тази цел в “HomeController” ще добавя една нова крайна точка наречена Slave.

```

1. public ActionResult Slave(string masterGuid)
2. {
3.     ViewBag.MasterGuid = masterGuid;
4.
5.     return View();
6. }

```

За успешното създаване на тази крайна точка създаваме изглед.



Фиг. 29 Добавяне на Slave изглед в Screenshare.Main

Новосъздадения изглед ще е мястото, където всички потребители/slaves ще виждат екрана на дадения клиент/master. За да пресъздадем екрана едно към едно, цялото тяло на клиента ще бъде пренесено в тази страница, като цялото съдържание ще бъде в iframe HTML елемент. HTML iframe се използва за показване на уеб страница в рамките на друга уеб страница. Точно това ще се случва в платформата за споделен екран. Ще зареждаме уеб страницата на клиента в страницата на потребителите. За да няма никакви разлики пренасяме цялата страница в този HTML елемент.

```

1. <body>
2.     @if (ViewBag.MasterGuid != null)
3.     {
4.         <div id="fullscreen">
5.             <iframe id="screenshare" ref="screenshare"
6.             class="fullScreen2"></iframe>
7.         </div>
8.         <div id="coverarea"></div>
9.         <i class="fas fa-mouse-pointer" id="cursor"></i>
10.    }
11.    else
12.    {
13.        <h1>No GUID sent</h1>
14.        <div>Please, check if your provided link is correct!</div>
15.    }
16.    @Scripts.Render("~/bundles/jquery")
17.    @Scripts.Render("~/bundles/bootstrap")
18.    <script src="~/Scripts/jquery.signalR-2.4.1.js"></script>
19.    <script src="~/signalr/hubs"></script>
20.    <script src="~/Scripts/src/screenshare_slave.js"></script>
21. </body>

```

В този екстракт от код използваме Razor изглед (глава 2.4), в които проверяваме дали имаме наличен “MasterGuid” и ако имаме създаваме iframe HTML елемента, който по-късно ще напълним със съдържанието на дадения клиент. Ред осемнадесет и деветнадесет реферират скриптове, които се използват за осъществяване на връзка със SignalR. Ред двадесети съдържа JavaScript код, който ще използвам за Slave JavaScript функционалността.

```

1. $(function () {
2.     const urlParams = new URLSearchParams(window.location.search);
3.     const masterGuid = urlParams.get('masterGuid');
4.
5.     if (masterGuid != null) {
6.         createEmptyHtmlTemplateInIframe();
7.     }
8. });

```

Функцията в този код се нарича IIFE, (Immediately Invoked Function Expression) което е JavaScript функция, която се изпълнява веднага щом бъде дефинирана. Това е дизайн модел, който е известен също като самоизпълняваща се анонимна функция. При наличието на “masterGuid” се изпълнява следваща функция наречена “createEmptyHtmlTemplateInIframe”.

```

1. function createEmptyHtmlTemplateInIframe() {
2.     let iframeElement = document.getElementsByTagName("iframe")[0];
3.     iframeElement.id = 'screenshare';
4.     iframeElement.setAttribute('scrolling', 'no');
5.     iframeElement.setAttribute('scrolling', 'no');
6.     iframeElement.setAttribute('frameborder', '0');
7.
8.     var ifrm = document.getElementById('screenshare');
9.     ifrm = ifrm.contentWindow || ifrm.contentDocument.document ||

```

```

    ifrm.contentDocument;
10.    ifrm.document.open();
11.    ifrm.document.write('<!DOCTYPE html>');
12.    ifrm.document.write('<html>');
13.    ifrm.document.write('<head></head>');
14.    ifrm.document.write('<body><div></div></body>');
15.    ifrm.document.write('</html>');
16.    ifrm.document.close();
17. }

```

Тази функция създава динамично `iframe` елемента, който ще съдържа клиентския HTML. За тази цел забраняваме скролиране и премахваме границите за този елемент, като така ще пресъздадем еднаква страница и на двете места. Следващата стъпка е да се свържем със SignalR.

```

1. let screencasting = $.connection.screencastingHub;
2.
3. screencasting.client.receiveInitialMasterBody = receiveInitialMasterBody;
4. screencasting.client.receiveMasterBody = receiveMasterBody;
5. screencasting.client.receiveMasterScroll = receiveMasterScroll;
6. screencasting.client.receiveMasterMouse = receiveMasterMouse;
7.
8. $.connection.hub.qs = 'uuid=' + masterGuid + '&master=false';
9. $.connection.hub.logging = true;
10.
11. $.connection.hub.start().done(function () {
12.     screencasting.server.receiveBody(masterGuid);
13. });

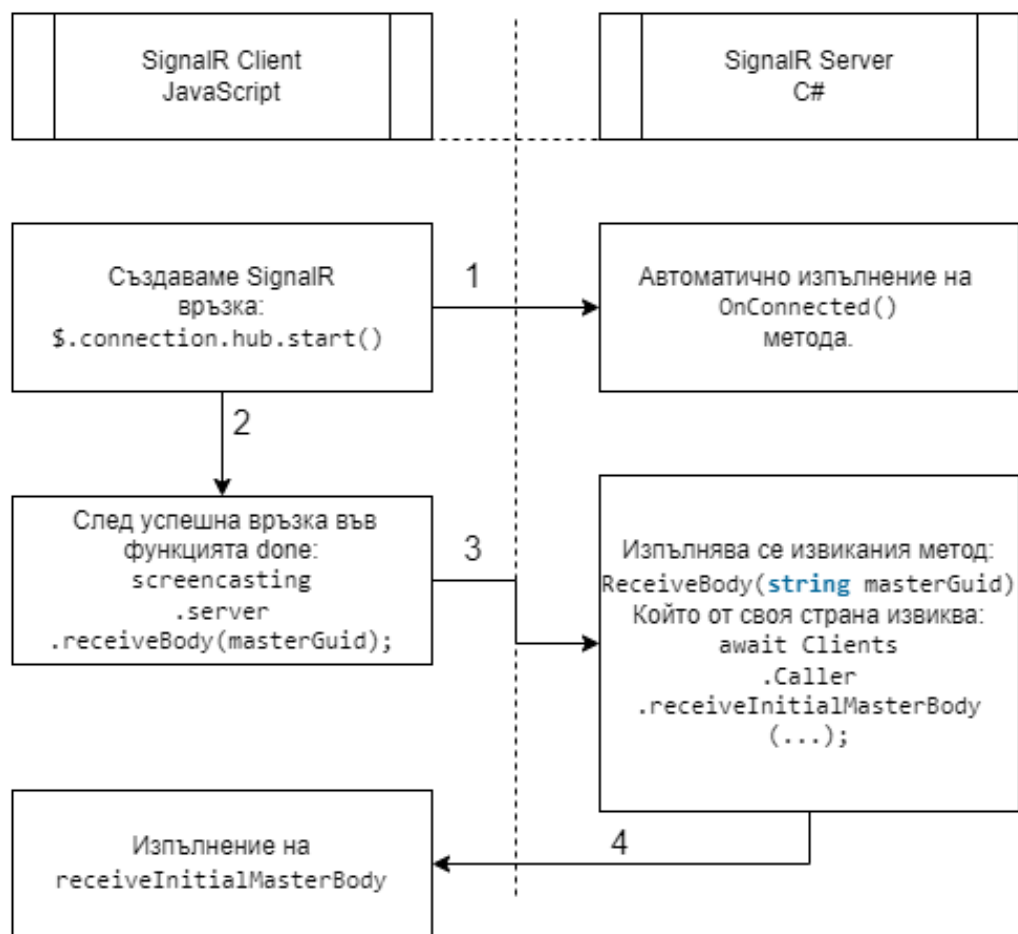
```

Ред първи оказва, че искаме да се свържем със SignalR Hub. По подразбиране се използва релативния път `“/signalR”`, който използваме и сме задали в OWIN Startup класа. От ред трети до шести описваме кои функции да бъдат изпълнени в JavaScript кода, когато сървърът повика и потърси такава със съвпадащо име. В следващата глава ще разгледаме подробно имплементацията на тези функции, тяхното повикване и ролята им в платформата за споделен екран.

Ред единадесети отваря връзка със SignalR, като след успешна установена връзка на сървър се изпълнява `“OnConnected”` метода, който разгледахме в глава 3.4. След успешно установената връзка ние влизаме в JavaScript функцията `“done”`, която изпълнява следната команда:

`screencasting.server.receiveBody(masterGuid)`

Този ред, реално извиква SignalR Hub метод с името `“ReceiveBody”`, които от своя страна извиква JavaScript функция `“receiveInitialMasterBody”`. За по-лесно разбиране на последователността от действия може да разгледаме следната фигура:



Фиг. 30 Ред на изпълнение на функции и методи между клиентска и сървърна част на SignalR

3.6 Имплементация на Master/клиент функционалност. Примерен TestClient проект.

В тази глава ще създадем примерен тестов проект, чието съдържание е без значение, тъй като самата платформа за споделен екран е абстрактна и независима от самото съдържание на страницата. За демонстрация съм избрал да направя тестово уеб приложение, чиято цел е избор на автомобил. Това е съвсем реално приложение, чиято имплементация може да се използва за по-лесната комуникация при покупко-продажба на автомобил.

Аналогично на “Screenshare.Main” проекта, тук отново следваме типичната за .NET MVC архитектура. Създаваме контролер и след това специфичен за него изглед.

```

1. <!doctype html>
2. <html lang="en">
3. <head>
4.     <meta charset="utf-8">
5.     <meta name="viewport" content="width=device-width, initial-scale=1">
6.     <meta name="description" content="">
7.     <meta name="author" content="George Dimov">
8.     <title>Screenshare Car Offer</title>
9.
10.    <!-- Bootstrap core CSS -->
11.    <link href="http://localhost:1672/Content/bootstrap-5.0.0/bootstrap.min.css"
    rel="stylesheet">
12.    <!-- Custom styles for this template -->
13.    <link href="http://localhost:1672/Content/dashboard.css" rel="stylesheet">
14. </head>
15. <body>
16.     <!-- HTML Content here, build with bootstrap 5.0 -->
17.
18.     <script src="~/Scripts/bootstrap-5.0.0/bootstrap.bundle.min.js"></script>
19.     <script src="~/Scripts/feather.min.js"></script>
20.     <script src="~/Scripts/bootstrap-5.0.0/dashboard.js"></script>
21.     @Scripts.Render("~/bundles/jquery")
22.
23.     <script src="~/Scripts/jquery.signalR-2.4.1.js"></script>
24.     <script src="https://localhost:44383/signalr/hubs"></script>
25.
26.     <!-- Main Master JavaScript code for screensharing --->
27.     <script src="/Scripts/src/screenshare_master.js"></script>
28. </body>
29. </html>

```

За по-бързото построяване на страницата използваме bootstrap 5.0. Bootstrap е най-популярната HTML, CSS и JavaScript рамка за разработване на адаптивни, ориентирани към мобилните устройства уебсайтове. Bootstrap е напълно безплатен за изтегляне и използване.

Всичко, което е нужно от страна на едно клиентско приложение, за да се интегрира платформата за споделен екран е:

- HTML съдържанието да съдържа бутон или друг елемент, който след натискане да се свърже с нашия backend. Важното е, този HTML елемент да съдържа атрибут ID (идентификатор) с име “Connect”.
- Да реферира ред двадесети и седми, който съдържа основния JavaScript код за клиента (“screenshare_master.js”).

Аналогично на “screenshare_slave.js”, така и тук в “screenshare_master.js” започваме с IIFE JavaScript функция.

```

1. $(function () {
2.     window.baseUrl = 'https://localhost:44383';
3.     window.areEventsAttached = false;
4.
5.     // Create master and send body to server
6.     $('#connect').click(onClickedConnect);
7. });

```


Обектът “window” се поддържа от всички браузъри. Той представлява прозореца на браузъра. Всички глобални JavaScript обекти, функции и променливи автоматично стават членове на този обект. Ред втори и трети създава глобални променливи, до които ще имаме достъп във всички останали функции. Променливата “baseUrl” трябва да има стойността на мрежовия адрес на backend проекта. След натискане на бутона, за които сме закачили слушател на събитие се изпълнява функция с име “onClickedConnect”.

```
1. function onClickedConnect() {
2.     if (sessionStorage.getItem('masterGuid') == null) {
3.         $.ajax({
4.             type: 'POST',
5.             url: window.baseUrl + '/api/Screencast/RequestConnection',
6.             dataType: 'json',
7.             data: {
8.             },
9.             success: onMasterGuidGenerated
10.        });
11.    }
12. }
```

В тази функция правим заявка до backend частта и контролера, които описах подробно в глава 3.3. В случай на успешна заявка и отговор от сървъра се изпълнява функцията “onMasterGuidGenerated”, която е една от най-основополагащите за платформата.

```
1. function onMasterGuidGenerated(apiResponse) {
2.     sessionStorage.setItem('masterGuid', apiResponse.GUID);
3.     $('#slaveData').append(`<a href="${apiResponse.URL}" class="text-decoration-
none">Screenshare link!</a>`);
4.     let masterGuid = apiResponse.GUID;
5.
6.     // Get master data
7.     let screenshot = document.documentElement;
8.     serializeInputs(screenshot);
9.     serializeScrollable(screenshot);
10.    let body = screenshot.children[1].innerHTML;
11.
12.    var headContent = document.getElementsByTagName('head')[0];
13.    let head = headContent.innerHTML;
14.
15.    let height = $(window).height();
16.    let width = $(window).width();
17.
18.    // Open SignalR connection and send master data
19.    $.connection.hub.url = window.baseUrl + '/signalr';
20.    let screencasting = $.connection.screencastingHub;
21.
22.    $.connection.hub.qs = 'uuid=' + masterGuid + '&master=true';
23.    $.connection.hub.logging = true;
24.
25.    if ($.connection.hub && $.connection.hub.state ===
$.signalR.connectionState.disconnected) {
26.        $.connection.hub.start().done(function () {
27.            screencasting.server.saveInitialMasterData(masterGuid, body, head,
```

```

        width, height);
28.     });
29. }
30.
31. if (window.areEventsAttached === false) {
32.     window.areEventsAttached = true;
33.     // To be shown and discussed below
34. }
35. }

```

Както споменах това е най-ключовата функция и в нея се случват в последователност няколко неща:

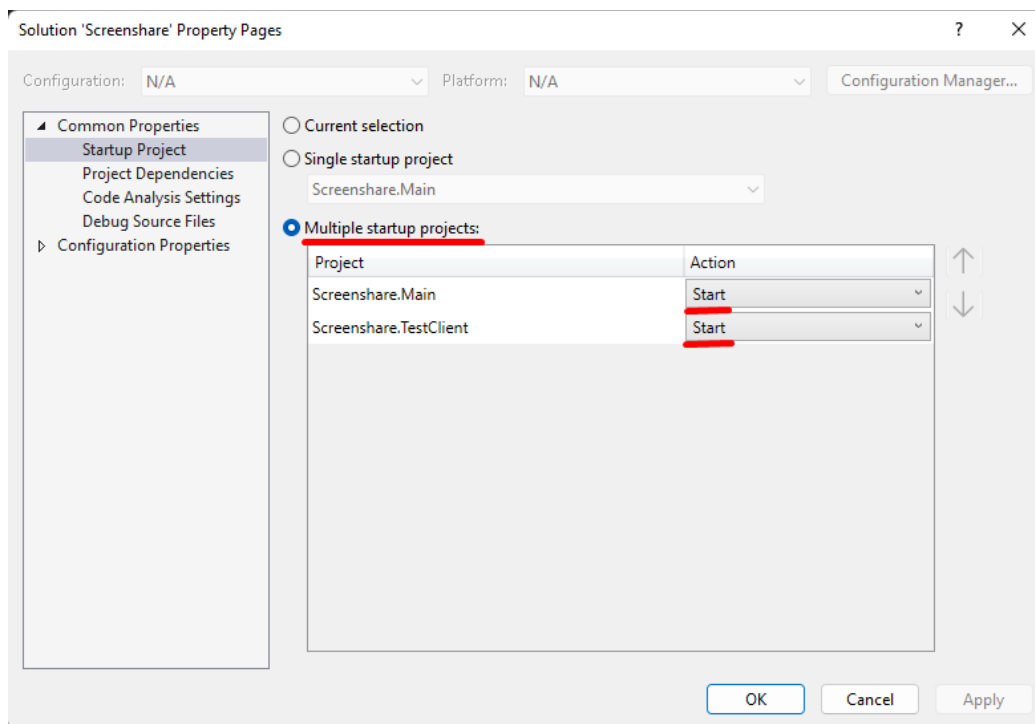
- След успешно направената HTTP заявка до нашия сървър ние получаваме отговор и този отговор се обработва тук. Като резултат от заявката се връща мрежови адрес, които трябва да бъде предоставен до един или много потребители/slaves. След натискане на този URL потребителите автоматично ще виждат екрана на клиента в реално време. За по-лесно копиране на създадената връзка в ред трети я визуализирам в HTML съдържанието.
- Следващите редове събират цялата информация за дадения клиент/master, като това включва неговото HTML Body, Head, дължина и ширина на прозореца.
- В следващите редове установяваме връзка със SignalR Hub, които се намира в нашия backend проект, като ключовите моменти тук са:
 - Използваме променливата “baseUrl”, която сме запазили в глобалния обект прозорец, тъй като самия backend проект е с различен мрежови идентификатор от този на текущото приложение.
 - Използваме допълнителни параметри показани на ред двадесети и втори, като “master=true” дава указания на сървъра, че текущия клиент е master. Ако направим паралел с “screenshare_slave.js” файла, то там този параметър имаше обратната стойност.
- След успешно свързване със SignalR на ред двадесети и седми правим директно повикване на метод с името “saveInitialMasterData”, които се намира на сървъра. Дадения метод запазва всички данни за текущия клиент.

```

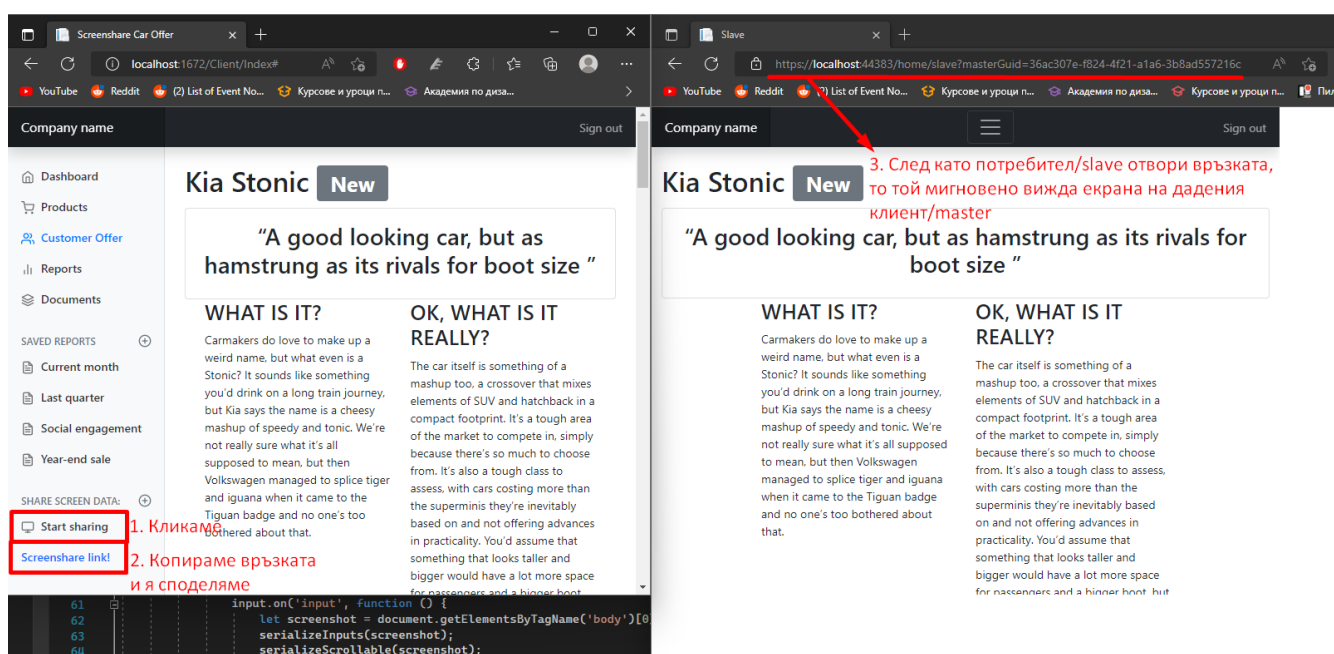
7. public async Task SaveInitialMasterData(string masterGuid, string body, string
   head, int width, int height)
1. {
2.     Master masterClient = GlobalCollections.users[masterGuid];
3.     masterClient.Body = body;
4.     masterClient.Head = head;
5.     masterClient.Width = width;
6.     masterClient.Height = height;
7.
8.     await Task.CompletedTask;
9. }

```

За да тестваме написания код до тук, може да стартираме и двата проекта във Visual Studio. Това се случва, чрез натискане на десен бутон върху Solution файла и след това избор на “Set Startup Projects...”



Фиг. 31 Избор на проекти, които да бъдат стартирани във Visual Studio 2022



Фиг. 32 Тестване на функционалността до този момент. Тестване на двата проекта: "Screenshare.Main" и "Screenshare.TestClient"

След стартиране на проектите трябва да предприемем няколко стъпки, които са демонстрирани във фигура 32. Започваме сесия за споделен екран след натискане на бутон, след което ни се генерира уникалният мрежови адрес, който копираме и изпращаме на потребителите си. В нашия случай за успешното тестване на цялата функционалност, трябва да отворим още един прозорец в браузъра, където поставяме

дадената връзка, симулирайки потребител/slave. Страницата на клиента веднага се появява и в другия прозорец. Тук може да забележим, че виждаме страницата едно към едно, като размери. Също така последния ред от текст визуализиран от HTML е “срязан”, както при клиента, така и при потребителя, въпреки че последния разполага с повече вертикално пространство на своята страница. Това ни демонстрира, че споделянето на екрана е пиксел перфектно. Ще забележим, че виждаме началната страница на клиента, но ако местим курсора, оразмерим страницата, пишем в текстово поле или скролираме страницата, то при потребителя промяна няма. За да направим цялата тази динамика, трябва да закачим слушатели на събития. Екстрактите от код, които следват се намират в проверката “if”.

```
1. if (window.areEventsAttached === false) {
2.
3.     window.areEventsAttached = true;
4.
5.     // To be shown and discussed below
6.
7. }
```

Проверявайки глобалната променлива “areEventsAttached” от обекта прозорец за стойност от “false”, ние реално проверяваме дали бутона за споделяне на екрана е бил натиснат няколко пъти. Това е важно, понеже следващите блокове от код искаме да се изпълнят само веднъж и да не закачим няколкократно еднакви слушатели на събития.

```
1. // Events for all inputs, textareas and selects
2. $('input, select, textarea').each(
3.     function (index) {
4.         var input = $(this);
5.         input.on('input', function () {
6.             let screenshot = document.getElementsByTagName('body')[0];
7.             serializeInputs(screenshot);
8.             serializeScrollable(screenshot);
9.             let body = screenshot.innerHTML;
10.            screencasting.server.updateSlaveBody(masterGuid, body);
11.        });
12.    }
13. );
```

Следния екстракт от код ще следи за промени в текстови полета, текстови зони и полета с опции за избор. След успешното закачане на този слушател, може клиента да попълва данни на потребителя в реално време, като последния ще вижда всичко, което се попълва и може да свери данните за тяхната вярност. В глава 1.1 имаме демонстрация на тази функционалност, която е показана във фигура 3.

Друго важно, което искам да демонстрирам е, че използваме и функции, чиято цел е изцяло спомагателна, както например е “*serializeInputs*”.

```
1. function serializeInputs(ele) {
2.     var _self = this;
3.
4.     if (!ele || !ele.nodeName) {
```

```

5.     return;
6. }
7.
8. switch (ele.nodeName) {
9.     default:
10.         if (ele.children) {
11.             for (var a in ele.children) {
12.                 serializeInputs(ele.children[a])
13.             }
14.         }
15.         break;
16.     case 'SCRIPT':
17.         ele.parentNode.removeChild(ele);
18.         break;
19.     case 'INPUT':
20.         switch (ele.type) {
21.             default:
22.                 ele.setAttribute('value', ele.value);
23.                 break;
24.             case 'checkbox':
25.             case 'radio':
26.                 if (ele.checked) {
27.                     ele.setAttribute('checked', 'checked');
28.                 }
29.                 else {
30.                     ele.removeAttribute('checked');
31.                 }
32.                 break;
33.             case 'file':
34.                 break;
35.             case 'password':
36.                 //Do not send password values...
37.                 // ele.setAttribute('value', ele.value);
38.                 break;
39.         }
40.         break;
41.     case 'TEXTAREA':
42.         ele.innerText = ele.value;
43.         break;
44.     case 'SELECT':
45.         // Some code here
46.         break;
47. }
48. }

```

Тази функция е с по-голямо съдържание и цялата може да бъде намерена в GitHub, тъй като платформата за споделяне на код е с отворен код. Нека разгледаме само най-важните части от тази функция. Тя е спомагателна, защото основната дейност на цялата платформа няма да се промени значително и без нея, но тя ни дава една допълнителна гъвкавост и сигурност. Ние изпълняваме тази функция всеки път преди да изпратим цялото HTML тяло до сървър. Като входящ параметър приема целия HTML, който ние обхождаме рекурсивно, докато не стигнем до най-вътрешните елементи. Проверяваме типовете елементи и правим допълнителни промени по наша преценка, като:

- Ако HTML елемента е от тип “<script>”, ние го пропускаме и не го изпращаме до сървъра. Това увеличава сигурността и намаля възможните атаки върху софтуера.
- Ако HTML елемента е “<input>” от тип парола, то тогава не изпращаме неговата стойност по SignalR, тъй като предпочитаме паролите да не се пренасят.
- Чрез тази функция може да направим всякакви допълнителни модификации по DOM дървото преди да го изпратим до сървъра.

След като клиент въведе някаква информация в текстово поле, то той в момента ще изпълни кода в дадения слушател на събития, а именно “*screencasting.server.updateSlaveBody(masterGuid, body);*”. Този ред се свързва със SignalR и извиква съответния метод на сървъра.

```

1. public async Task UpdateSlaveBody(string masterGuid, string body)
2. {
3.     Master masterClient = GlobalCollections.users[masterGuid];
4.     masterClient.Body = body;
5.
6.     foreach (Slave hubClient in masterClient.Slaves)
7.     {
8.         var client = Clients.Client(hubClient.ConnectionID);
9.
10.        await client.receiveMasterBody(body, masterClient.Width,
11.        masterClient.Height);
12.    }
13. }
```

Това, което се случва е, че презаписваме HTML тялото за всички потребители/slaves, които са се закачили за нашия клиент/master. Тъй като връзката е двупосочна и постоянно отворена, тази промяна се случва мигновено.

```

1. // Event for scrolling
2. document.addEventListener('scroll', function (event) {
3.     let screenshotScrollY = $(document).scrollTop();
4.     let screenshotScrollX = $(document).scrollLeft();
5.
6.     screencasting.server.updateSlaveScroll(masterGuid, screenshotScrollX,
7.     screenshotScrollY);
8. }, true /*Capture event*/);
9. // Event for mouse position
10. setTimeout(function () {
11.     $(document).on('mousemove', function (event) {
12.         let mouseX = event.clientX;
13.         let mouseY = event.clientY;
14.
15.         screencasting.server.updateSlaveMousePosition(masterGuid, mouseX,
16.         mouseY);
17.     });
18. }, 1000);
```

В този екстракт от код показваме два слушателя на събития, единия е при скролиране на страницата, а другия при промяна положението на курсора в клиентската страница. Двата слушателя извикват SignalR методи на сървъра, като те се изпълняват светкавично бързо, понеже като параметри подаваме само числови стойности, които реално са X хоризонтална и Y вертикална стойност на позицията на страницата, както и съответните и X и Y стойности на позицията на курсора.

```
1. public async Task UpdateSlaveScroll(string masterGuid, int scrollPositionX, int
   scrollPositionY)
2. {
3.     Master masterClient = GlobalCollections.users[masterGuid];
4.     masterClient.ScrollTop = scrollPositionY;
5.     masterClient.ScrollLeft = scrollPositionX;
6.
7.     foreach (Slave hubClient in masterClient.Slaves)
8.     {
9.         var client = Clients.Client(hubClient.ConnectionID);
10.
11.         await client.receiveMasterScroll(scrollPositionX, scrollPositionY);
12.     }
13. }
14.
15. public async Task UpdateSlaveMousePosition(string masterGuid, int mousePositionX,
    int mousePositionY)
16. {
17.     Master masterClient = GlobalCollections.users[masterGuid];
18.     masterClient.CursorTop = mousePositionY;
19.     masterClient.CursorLeft = mousePositionX;
20.     foreach (Slave hubClient in masterClient.Slaves)
21.     {
22.         var client = Clients.Client(hubClient.ConnectionID);
23.
24.         await client.receiveMasterMouse(mousePositionX, mousePositionY);
25.     }
26. }
```

Аналогично на по-горния слушател на събития, имаме същата логика. Взимаме всички потребители/slaves за даден клиент/master и извикваме съответните методи при тях.

```
1. // Event for window resizing
2. window.addEventListener('resize', function (event) {
3.     let screenshot = document.documentElement;
4.     serializeInputs(screenshot);
5.     let body = screenshot.children[1].innerHTML;
6.
7.     let height = $(window).height();
8.     let width = $(window).width();
9.
10.     screencasting.server.updateSlaveResizedBody(masterGuid, body, width, height);
11. });
12.
13. //Events for textarea div scrolling
14. let allElementsToAttachOnScroll = document.querySelectorAll('textarea, div,
    select');
```

```

15.
16. allElementsToAttachOnScroll.forEach(function (entry) {
17.     entry.onscroll = logScroll;
18. });
19.
20. function logScroll(e) {
21.     let screenshot = document.documentElement;
22.     serializeInputs(screenshot);
23.     serializeScrollable(screenshot);
24.     let body = screenshot.children[1].innerHTML;
25.
26.     screencasting.server.updateSlaveBody(masterGuid, body);
27. }

```

Последните слушатели на събития се изпълняват, когато клиентът промени големината на своята страница или въведе прекалено много съдържание в “textarea” елемент и се появи скрол.

```

1. public async Task UpdateSlaveResizedBody(string masterGuid, string body, int
   width, int height)
2. {
3.     Master masterClient = GlobalCollections.users[masterGuid];
4.     masterClient.Body = body;
5.     masterClient.Width = width;
6.     masterClient.Height = height;
7.
8.     foreach (Slave hubClient in masterClient.Slaves)
9.     {
10.         var client = Clients.Client(hubClient.ConnectionID);
11.
12.         await client.receiveMasterBody(body, masterClient.Width,
           masterClient.Height);
13.     }
14. }

```

Това е метода, който се изпълнява на сървъра след като клиент промени своя размер на страницата.

Покажахме всички функции в “screenshare_master.js”, които се изпълняват след действие на клиента. Това действие от своя страна предизвиква даден слушател на събитие, който извиква SignalR метод. Всички backend SignalR методи горе следват еднаква логика, която е да изпълнят дадена функция при потребителите/slaves, които принадлежат на дадения клиент/master. Имплементацията на тези функции може да намерим в “screenshare_slave.js”, като в екстракта от код долу ще покажем споменатите до тук.

```

1. function receiveMasterBody(body, width, height) {
2.     let iframe = document.getElementById('screenshare');
3.     let iframeWindow = iframe.contentWindow;
4.
5.     $("#screenshare").width(width);
6.     $("#screenshare").height(height);
7.
8.     iframeWindow.document.body.innerHTML = body;

```



```

9.     iframeWindow.document.body.innerHTML.replace('</body>', '<style> *{animation-
duration: 0s !important} html, body {user-select: none !important; pointer-event:
none !important;}</style></body>');
10.
11.     let jBody = $(iframeWindow.document.body);
12.     let scrollElems = jBody.find('[data-scroll]');
13.     for (let a = 0; a < scrollElems.length; a++) {
14.         let scrollPositions = scrollElems[a].getAttribute('data-
scroll').split(',').map(x => +x);
15.         $(scrollElems[a]).scrollTop(scrollPositions[0]);
16.     }
17. }
18.
19. function receiveMasterScroll(xPercent, yPercent) {
20.     let iframe = document.getElementById('screenshare');
21.     let iframeWindow = iframe.contentWindow;
22.
23.     iframeWindow.scrollTo(xPercent, yPercent);
24. }
25.
26. function receiveMasterMouse(mousePositionX, mousePositionY) {
27.     $('#cursor')
28.         .css('position', 'absolute')
29.         .css('top', `${mousePositionY}px`)
30.         .css('left', `${mousePositionX}px`)
31.         .css('z-index', '1');
32. }

```

Ако следваме инструкциите за разработка на платформата показани в глава 3, в момента ще имаме напълно работещо решение. Показаните C# методи, JavaScript функции и HTML съдържание са основополагащи за платформата за споделен екран. След имплементацията на тази базова функционалност ще имаме работещ проект, който ще предоставя опция за споделяне на екран. След споделяне на екрана се получава уникален мрежови адрес, който ако бъде споделен и достъпен от потребители, те ще виждат екрана на клиента в реално време, пиксел перфектен.

Заклучения и перспективи за развитие

В почти всеки един сектор в страната може да се наблюдава силна дигитализация. След глобална пандемия, като COVID-19, този процес се засилва многократно. Всичко, което до преди се е извършвало на място, сега се предпочита да е изцяло онлайн или от разстояние. Това са предпочитания не само клиентите, но и на големите фирми и предприятия.

Основна цел на тази дипломна работа е разработката на платформа, чрез която може да се споделя екрана в реално време с клиенти, като това да улесни максимално комуникацията между двете страни. Основната таргет група на този софтуер са големи компании, които искат техния продукт да предлага допълнителни услуги. Платформата е абстрактна и може лесно да бъде имплементирана в друг уеб базиран продукт. Голямата нужда и плюс на тази платформа е, че може да стане част от самия уеб продукт на дадена фирма, като така се отстранява нуждата от допълнителен софтуер, който трябва да бъде инсталиран от двете страни в нужда на споделяне на екран. Освен това отпада притеснението и на каква операционна система са клиентите, тъй като всичко се случва в брауъра на самия сайт на дадена фирма.

Поради засилващата се дигитализация все повече фирми започват да търсят софтуерни компании, които да им помогнат за този процес. Софтуерния пазар става все по-голям и конкуренцията в него нараства с големи темпове. Поради тази причина търсенето на софтуер, който използва по-иновативни технологии, като двупосочна комуникация в реално време между сървър и брауър нараства. Този тип приложения позволяват промени по страниците без нужда от презареждане. Друга основна цел на дипломната работа е разглеждането на точно такъв тип технология, която е именно .NET SignalR.

Разработената платформа е изключително перспективна поради възможностите, които предлага за допълнително развитие:

- Най-голямата перспектива за развитие би била направата на един глобален конфигурационен файл, в който може да се намират всички настройки, чрез които може да бъде контролирана самата платформа. Тези настройки може да константи, които да се съхраняват на едно общо и удобно място или настройки, които да оказват дали при споделяне на екрана да се пренасят пароли (глава 3.6).
- Друга перспектива би била, чрез допълнителна разработка, да е възможно за потребителите да модифицират полетата на клиента. Текущо потребителите само наблюдават екрана на клиента, но така биха участвали в действия също.
- Друга много добра идея за допълнително развитие на платформата е, да се взима на предвид големината на прозореца, както на клиента, така и на потребителя. В момента потребителите виждат страницата на клиента пиксел перфектно, което означава, че ако те са на екран с по-малка резолюция биха виждали вертикален или хоризонтален скрол. Това може да бъде доработено.
- Често местата на които се изпълнява кода на сървъра имат специфични характеристики и особености. Както разгледахме в глава 2.3 SignalR избира сам

коя е най-добрата техника за постоянно отворена връзка. Това също може да е допълнителна настройка, според която се позволява или забранява Long polling например.

- Друга идея за допълнително развитие би била наличието на избор от различни курсори, които да вижда клиента, както и те да бъдат сменяни.
- В момента ако даден HTML елемент има ефект, който се изпълнява, когато курсора е върху него, то той няма да бъде видян от потребителите. Това може да бъде променено с допълнителна разработка.

Платформата за споделен екран е с отворен код и нейното развитие може да бъде следено в GitHub. Този проект би запълнил реална нужда на софтуерни компании, които биха искали да предложат допълнителни екстри на своите клиенти, а също така да направят процеса на онлайн покупко-продажби много по-лесен, особено след настъпилата глобална пандемия.

Използвана литература

[1] **Architect Modern Web Applications with ASP.NET Core and Azure**, EDITION v6.0, PUBLISHED BY Microsoft Developer Division, .NET, and Visual Studio product teams

<https://dotnet.microsoft.com/en-us/download/e-book/aspnet/pdf>

[2] **Introduction to SignalR**, Article, by Patrick Fletcher, 09/10/2020

<https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr>

[3] **ASP.NET SignalR Hubs API Guide – Server (C#)**, Article, by Patrick Fletcher, 02/11/2022

<https://docs.microsoft.com/en-us/aspnet/signalr/overview/guide-to-the-api/hubs-api-guide-server>

[4] **Free/Open Source Software**, A General Introduction by Kenneth Wong and Phet Seyo, ISBN: 983-3094-00-7

[5] **Принципи на програмирането със C#**, Светлин Наков, Веселин Колев и колектив, издателство: Фабер, Велико Търново, 2018 г., ISBN: 978-619-00-0778-4

[6] **Open Source Security & License Compliance**

<https://www.blackducksoftware.com/resources/infographics/deep-license-data>

[7] **Wikipedia:**

- <https://en.wikipedia.org/wiki/SignalR>
- <https://en.wikipedia.org/wiki/WebSocket>

[8] **Състояние и развитие на ИТ сектора**, София, Септември 2017

https://investsofia.com/wp-content/uploads/2017/10/IT_Sector_Analysis_Sofia_Sept-2017_BG.pdf

[9] **The benefits of using web-based applications**, December 20, 2019, Paymon Khamooshi

<https://www.geeks.ltd.uk/insights/the-benefits-of-using-web-based-applications>

[10] **Build Real-time Applications with ASP.NET Core SignalR**, By Anthony Chu, 2018 July/August

<https://www.codemag.com/article/1807061/Build-Real-time-Applications-with-ASP.NET-Core-SignalR>

[11] **Use hubs in SignalR for ASP.NET Core**, Article, 07/16/2022, By Rachel Appel and Kevin Griffin

<https://docs.microsoft.com/en-us/aspnet/core/signalr/hubs?view=aspnetcore-5.0>

[12] **SignalR Programming in Microsoft ASP.NET**, José M. Aguilar, ISBN: 978-0-7356-8388-4

[13] **Push Service with ASP.NET SignalR**, LAHTI UNIVERSITY OF APPLIED SCIENCES

https://www.theseus.fi/bitstream/handle/10024/134041/Kekkonen_Maija.pdf;jsessionid=56556290E1173A6F279EF5204A4B30F6?sequence=1

[14] **Eloquent JavaScript**, 3rd edition (2018), Written by Marijn Haverbeke.

<https://eloquentjavascript.net/>

[15] **DevOps Management with GitHub**

<https://www.happiestminds.com/wp-content/uploads/2022/06/DevOps-Management-with-GitHub.pdf>

[16] **How GitHub Democratized Coding, Built a \$2 Billion Business, and Found a New Home at Microsoft**, Hiten Shah

<https://nira.com/github-history/>

Съдържание

Увод, цел и задачи.....	3
Глава 1. Характеристика на платформата.....	5
1.1 Приложение и Функция.....	5
1.2 Схема на изпълнение.....	9
1.3 Архитектурна характеристика.....	14
1.4 Софтуер с отворен код.....	17
1.5 Лиценз.....	19
Глава 2. Избор и обосновка на използваните технологии.....	22
2.1 C# и .NET.....	22
2.2 Microsoft Visual Studio 2019 & 2022.....	25
2.3 SignalR.....	27
2.4 Razor View Engine.....	28
2.5 JavaScript & jQuery.....	29
2.6 Github & Sourcetree – GIT GUI.....	31
Глава 3. Разработване на платформата.....	34
3.1 Структура на платформата, създаване на проектите.....	34
3.2 Основни модели на платформата.....	37
3.3 Основен API контролер.....	39
3.4 Добавяне на SignalR. Сървърна SignalR част.....	40
3.5 Имплементация на Slave/потребител функционалност. SignalR JavaScript част.....	44
3.6 Имплементация на Master/клиент функционалност. Примерен TestClient проект.....	47
Закljučения и перспективи за развитие.....	58
Използвана литература.....	60
Съдържание.....	62