WEB BROWSER SCREEN SHARING PLATFORM VIA ASP .NET SIGNALR

Assoc. Prof. PhD Dimitar Minchev, Burgas Free University, mitko@bfu.bg
George Dimov, graduate student, 16321042@students.bfu.bg

Abstract: This publication presents a working software solution for sharing screen and web content in real time in the web browser, using Microsoft .NET and SignalR technologies.

Keywords: Web Browser Screen Sharing, Microsoft, ASP, .NET, SignalR.

ПЛАТФОРМА ЗА СПОДЕЛЯНЕ НА ЕКРАНА В УЕБ БРАУЗЪРА ЧРЕЗ ASP .NET SIGNALR

доц. д-р Димитър Минчев, Бургаски свободен университет, mitko@bfu.bg
Георги Димов, дипломант, 16321042@students.bfu.bg

Абстракт: Тази публикация представя работещо софтуерно решение за споделяне на екран и уеб съдържание в реално време в уеб браузъра, посредством Майкрософт технологиите ASP, .NET и SignalR.

Ключови думи: Web Browser Screen Sharing, Microsoft, ASP, .NET, SignalR.

Характеристика на платформата

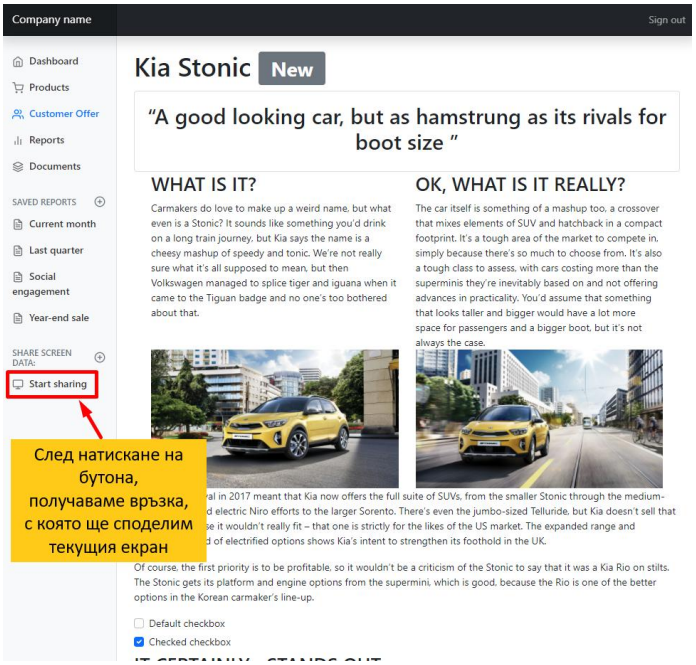
Приложение и Функция

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.

Направен е 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

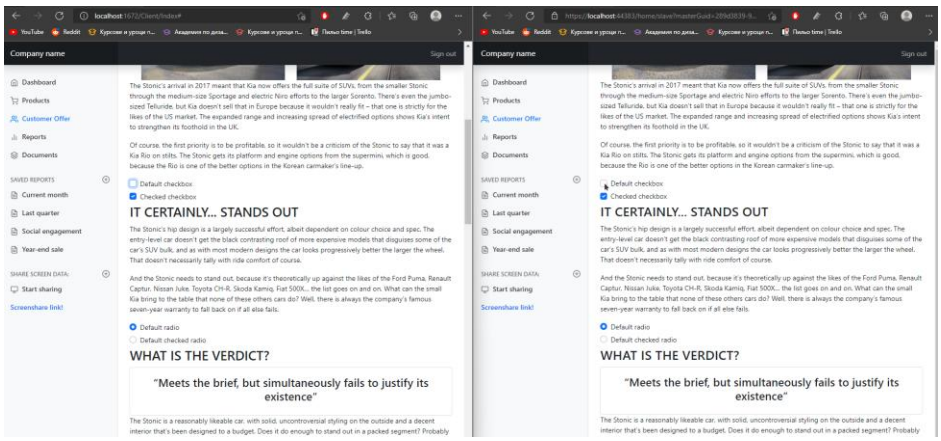
неговите данни, като адрес, име и други са правилно изписани от служителите, намалявайки значително допускането на различни грешки. На Фиг. 1 може да видим как изглежда примерния тестови клиент.



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.

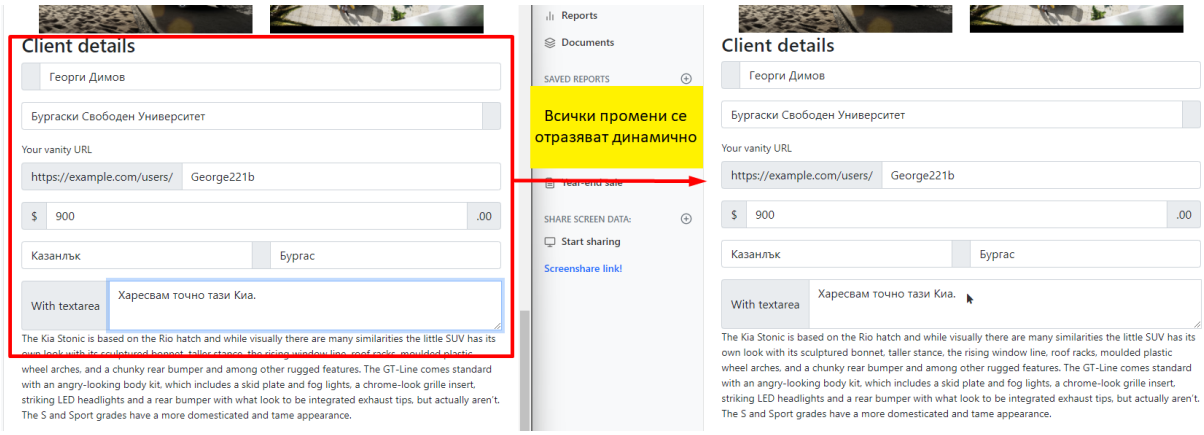
След като връзката за споделяне на екрана бъде отворена от един или повече потребители, съдържанието на клиентската страница се зарежда мигновено при тях. Всички действия и промени на клиента се отразяват динамично при неговите потребители. Като това се случва без да е нужно опресняване на страницата.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

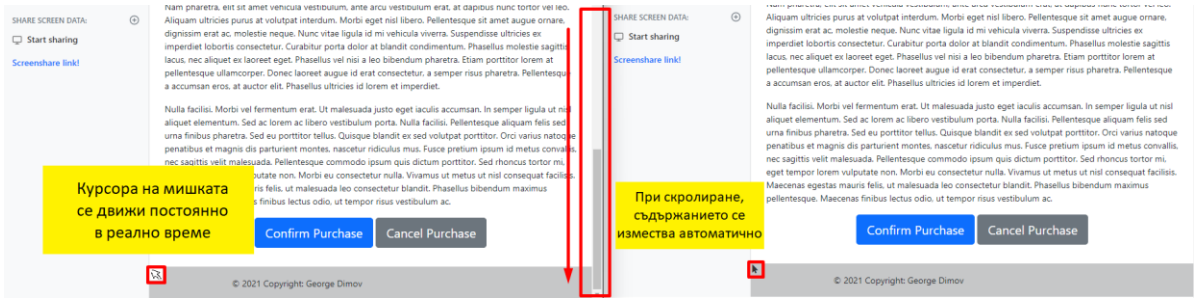
Функциите, които предлага платформата за споделен екран, са динамичното зареждане на страницата при всеки потребител в зависимост от всички действия, които предприема клиента на неговата страница. Тези действия включват:

1. Промяна съдържанието на текстови полета (HTML[1] input, select, textarea[2] елементи) и всички възможни елементи за промяна на техния статус/избор.



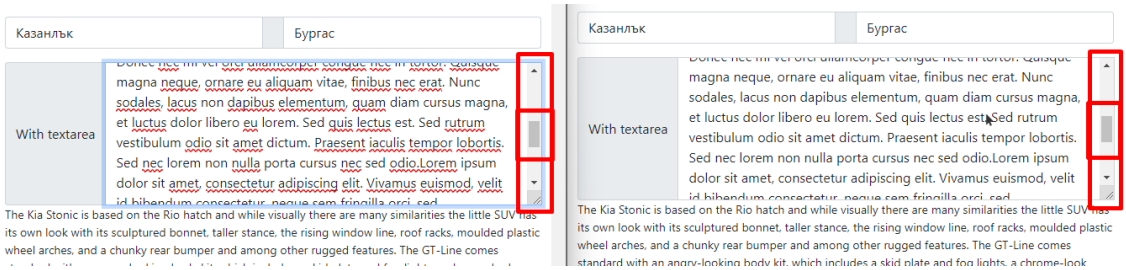
Фиг. 3 Демонстрация на функцията за динамично пренасяне на съдържание въведено в текстови форми.

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



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

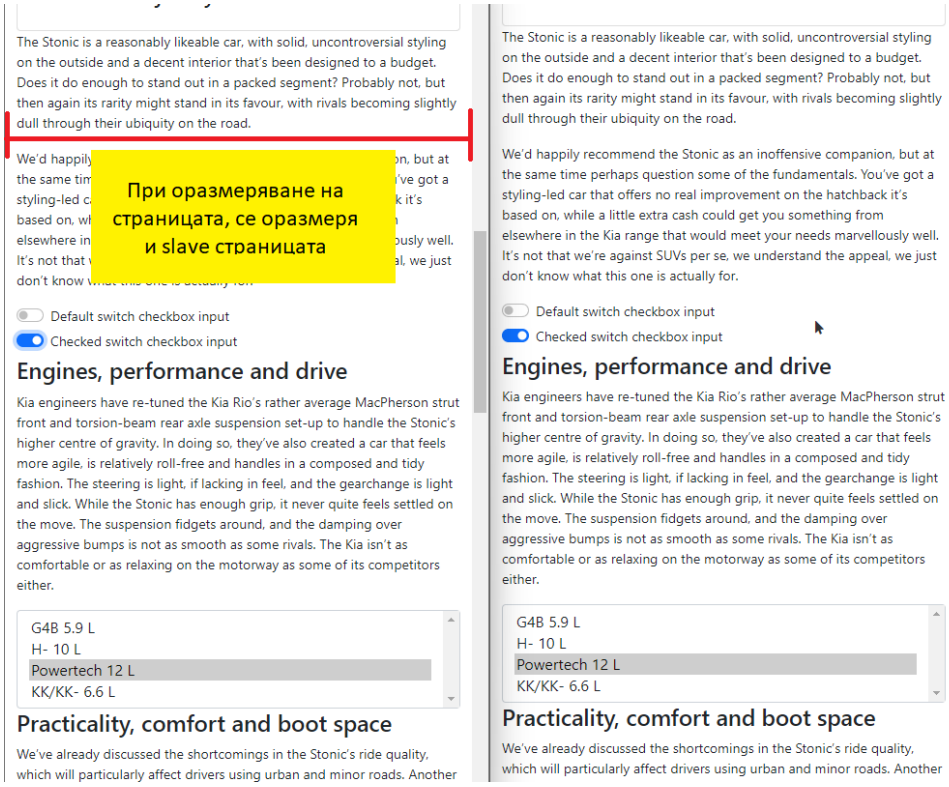
4. В случай, че в даден елемент се появи скрол, като това примерно би се случило при въвеждане на много информация в textarea HTML елемент, то скролирането в този елемент се пренася също.



Фиг. 5 При поява на скрол в даден/и елементи, скролирането в тези елементи се пренасят в съответствие със скрол позицията на клиента.

[1] **HTML** = HTML (HyperText Markup Language, произнасяно най-често като „ейч-ти-ем-ел“) е основният маркиращ език за описание и дизайн на уеб страници.
[2] **input, select, textarea** = видове HTML елементи, които служат за приемане на данни от клиента.

5. Оразмеряване на страницата.



Фиг. 6 Оразмеряване на страницата. Свиване на страницата хоризонтално. Промените се отразяват съобразено с това.

6. При сложни промени в DOM[3] дървото на клиента, като това се конфигурира допълнително.

За да бъде по-ясно демонстрирано приложението и функциите на платформата за споделен екран са използвани 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 code[4] на платформата за споделен екран, както и оказания за нейното използване.

Схема на изпълнение

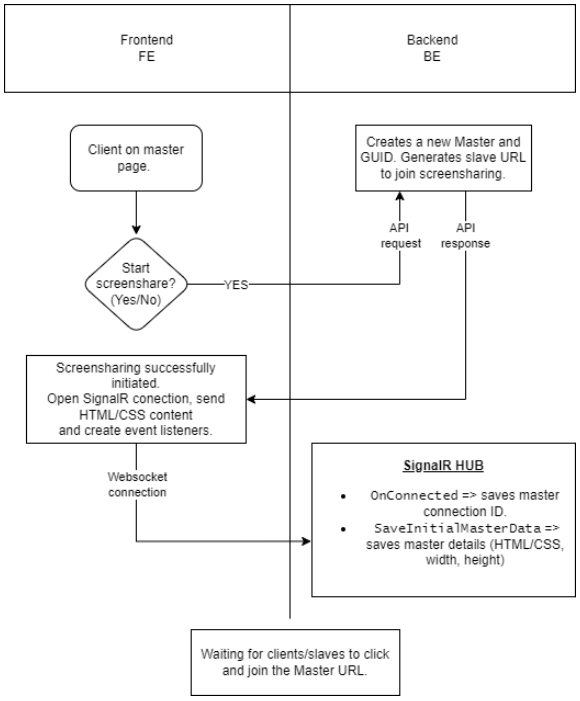
Във Фиг. **7**, **8** и **9** съдържат три блок схеми, чиято цел е да демонстрират цялостната работа, ход и изпълнение на платформата за споделен екран.

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.

[3] **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.

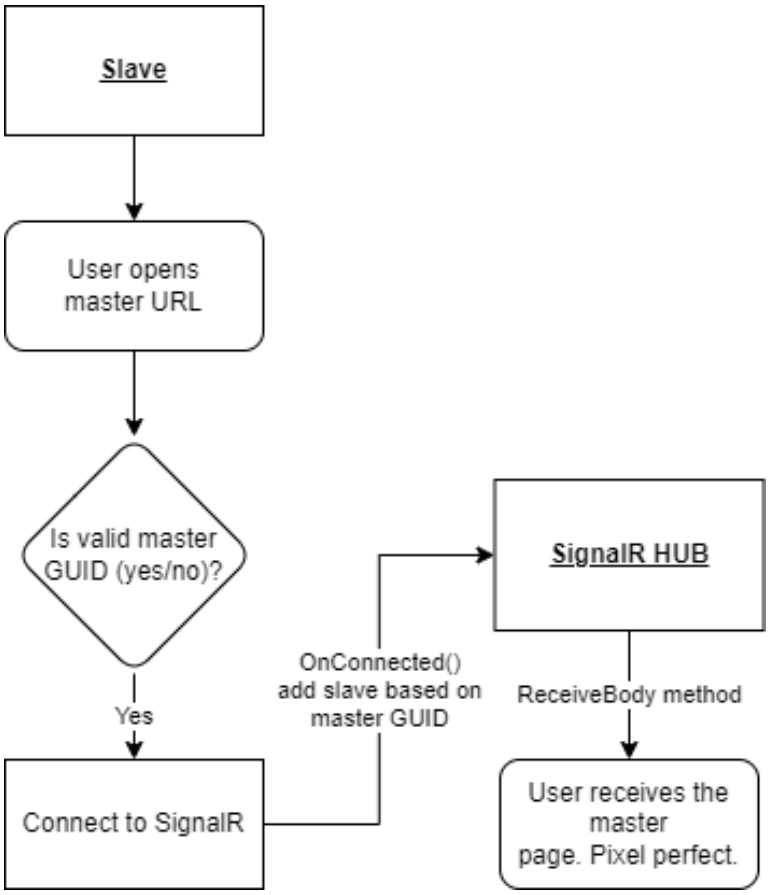
[4] **source code** = изходен код. Изходният код е сбор от инструкции (заедно с коментарите), написан на разбираем за човека език за програмиране обикновено като текст. Изходният код позволява модификация на компютърната програма, разглеждане на начина, по който тя работи, откриване на грешки и други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

Фиг. **7** показва, че когато клиент се намира във клиентската част на приложението, то той взаимодейства с нашия backend API [5]. Това се осъществява, чрез изпращане на HTTP[6] request[7]. 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 протокола[8]. В по-голяма дълбочина ще разгледаме темите за SignalR и WebSocket (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 неговия HTML, CSS[9], дължина и височина на страницата.
След това се закачат всички event listeners[10] за всички клиентски действия. Накрая като имаме цялата информация за дадения master/клиент и сме закачили всички event listerners, 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.

[5] **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.
[6] **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.
[7] **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.
[8] **WebSocket протокол** = WebSocket е компютърен комуникационен протокол, осигуряващ двупосочна интерактивна комуникационна сесия между браузъра на потребителя и сървъра през една TCP връзка.
[9] **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
[10] **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.

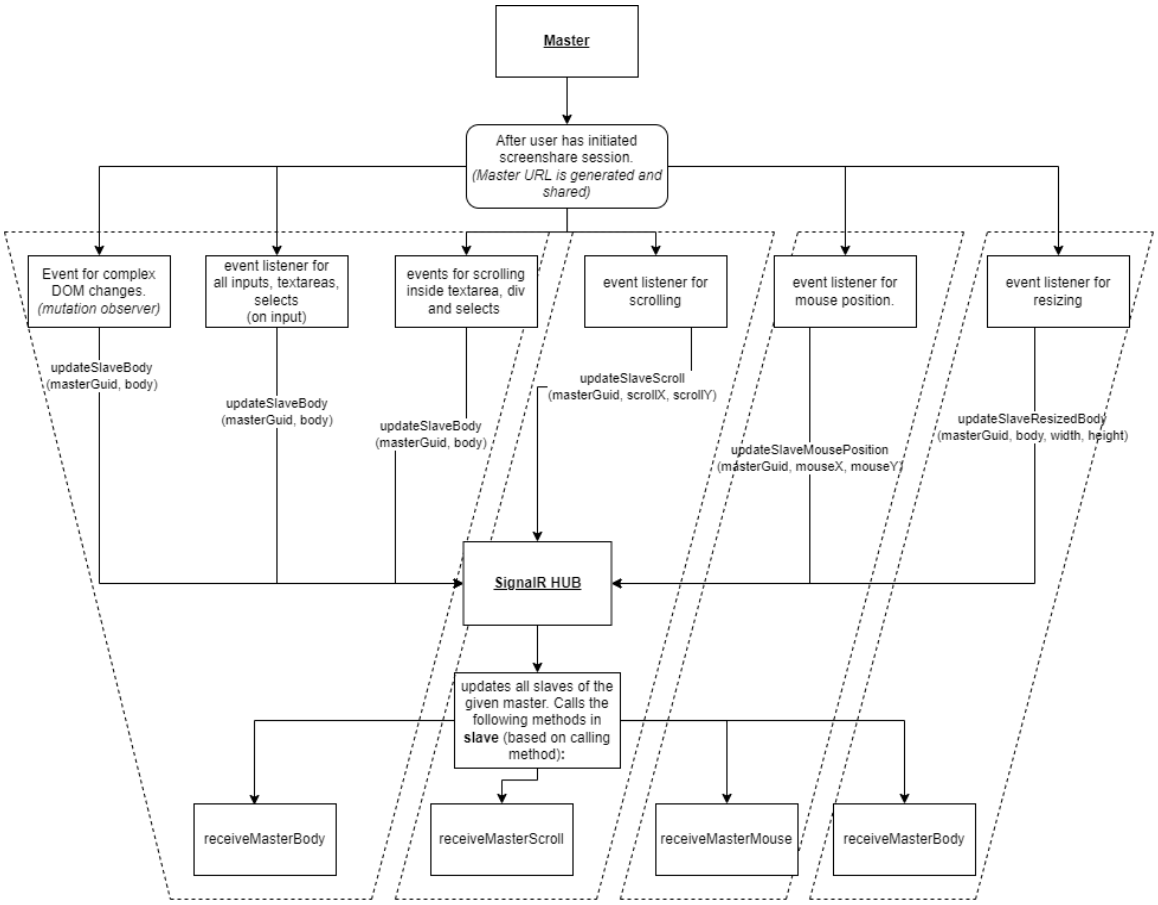


Фиг. 8 Втора блок схема демонстрираща хода на действията след като се присъедини потребител към платформата за споделен екран.

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[11] 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 уебсокети.

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 сървъра, 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 сървъра.

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



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

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 Solution[12] файл. Solution файлът представлява структура за организиране на различните проекти във Visual Studio[13]. Той съдържа информация за състоянието на проектите, техните референции и средата за разработка.

Папка ***Source*** съдържа backend частта на платформата. Backend е сървърната страна на един уебсайт. Той съхранява и подрежда 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; 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

[12] **Solution** = .sln файл, текстово базиран, споделен, вж: https://docs.microsoft.com/en-us/visualstudio/ide/solutions-and-projects-in-visual-studio?view=vs-2019

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

ни позволява да осъществяваме RPC [14] 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 споделения екран. Този модел се нарича Master/Slave[15].

Другия основен backend компонент в платформата е контролера [16] “*ScreencastController*”. Той се извиква първоначално преди да е започнало споделянето на екран в платформата. Отговорен е за създаването на master и генерирането на уникален GUID[17], 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*” view[18]. Тук използваме най-широко разпространения дизайн Модел-Изглед-Контролер[19]. Клиентите, който достъпват този изглед са slaves. Тук се очаква задължително подаден query [20]параметър, който идентифицира точно на кой master да бъде споделен екрана. В случай, че достъпим този изглед без подадения уникален идентификатор (GUID) за master, ще видим грешка, че въведения URL е грешен.

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.

---

[14] **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.

[15] **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.

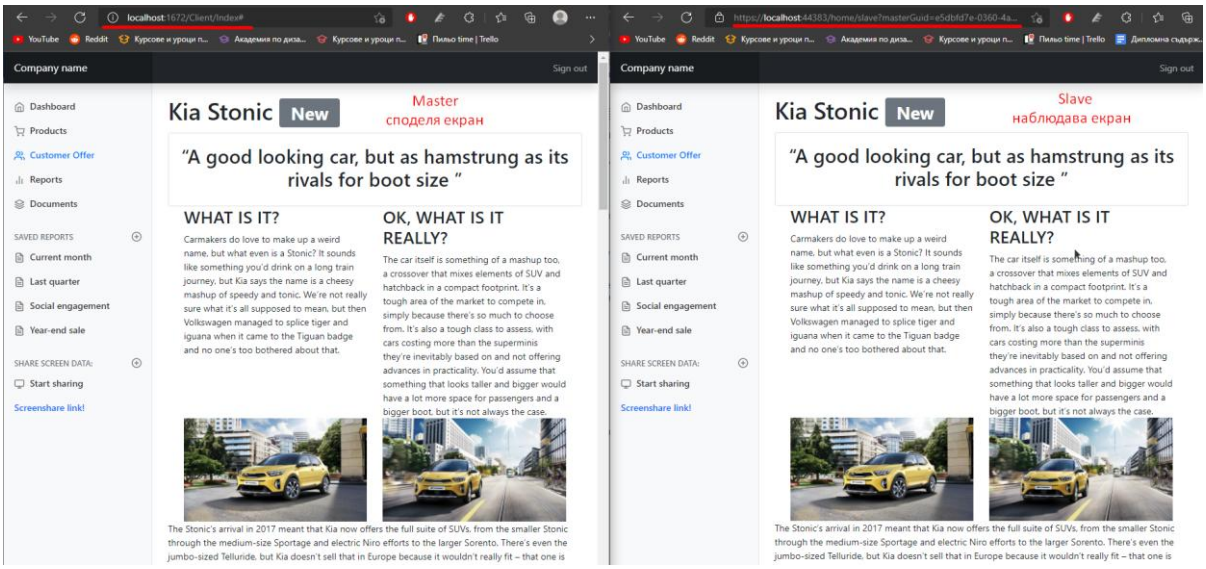
[16] **Controller** = В уеб API контролер е обект, който обработва HTTP заявки.

[17] **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.

[18] **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.

[19] **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

[20] **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.



Фиг. 10 Успешно установена връзка за споделяне на екран между master и slave

Важно нещо да се отбележи е, че клиентското приложение за оторизирани сервизи (master) се изпълнява на различен domain[21] 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 случай домейна на master се очаква да е frontend [22]частта на едно приложение, докато slave ще се изпълнява, където е API или самия backend[23] .

Фиг. 11 Споделянето на екран се изпълнява на различни домейни.

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.

---

[21] **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

[22] **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.

[23] **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

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-разпространения, 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)

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.

Избор и обосновка на използваните технологии

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#.

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.

C# е един от най-широко използваните езици за програмиране днес и постоянно се нарежда сред водещите езици в индекса TIOBE[24] и проучването за разработчици на Stack Overflow[25]. 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 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.

Microsoft Visual Studio 2019 & 2022

Интегрирана среда за разработка (IDE[26]) 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.

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.

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.

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 сървъра, 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 сървъра за създаване на постоянно отворена връзка между клиента и сървъра.

[24] **TIOBE** = Създаден от TIOBE Company. Името му е абревиатура 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/

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

[26] **IDE** = Integrated development environment = Интегрирана среда за разработка

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. Използва се само една постоянно отворена връзка, а SignalR ще се погрижи за останалото, правейки всичко да функционира.

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 изгледа съчетава C# код и HTML, като дадения файл, които 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. За превключване от HTML в C# код се използва символа @, след който вече може да пишем нашия код на C#.

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 Netscape[27] 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 предназначение е вградено в Netscape, Internet Explorer и други браузъри. Спецификацията ECMA-262 дефинира стандартна версия на основния JavaScript език.

JavaScript, който се изпълнява от страна на клиента, като например в браузър, е най-често срещаната форма на езика. Скриптовия файл съдържащ езика трябва да бъде включен или реферирaн от HTML документ, за да може кодът да се интерпретира от браузъра. 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 браузъра 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.

Едно от основните предимства на JavaScript е, че не изисква скъпи инструменти за разработка. Можете да започнете с прост текстов редактор като Notepad[28]. Тъй като това е интерпретиран език в контекста на уеб браузъра, 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 Ajax[29] взаимодействия за бързо уеб развитие. jQuery е помощна библиотека заJavaScript, предназначена да опрости различни видове често срещани задачи чрез писане на по-малко код.

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

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

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

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 SaaS[30] 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 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).

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.

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.

Продуктът е подходящ както за начинаещи, така и за напреднали потребители, богатият набор от функции на Sourcetree помага на продуктивноста.

Разработване на платформата

Ту ще разгледаме разработката на 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, 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.

Структура на платформата, създаване на проектите

Започваме със създаването на основната структура. За тази цел създаваме един Blank Solution, който кръщаваме “*Screenshare*”. След това създаваме един нов проект от тип ASP.NET Web Application (.NET Framework). На следващия екран даваме правилното име на проекта, в нашия

[30] **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.

случай “*Screenshare.Main*”. 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.
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.
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. За версия на контролите използваме Sourcetree и GitHub.
Историята на промените до пълното изграждане на платформата е достъпна в Интернет на адрес: (https://github.com/George221b/screenshare-in-the-web-signalr/commits/main).

Основни модели на платформата

Да 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.
“*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 малко-по късно. 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. Вече обяснихме функцията и разликата между клиент/master и потребител/slave. Тук ще създадем съответните класове за това. 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.

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 малко по-късно.
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.

Фрагмента с код 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** [31] 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 *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 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[32]), след което връщаме модел попълнен с информация.

Добавяне на SignalR. Сървърна SignalR част.

Разгледахме подробно технологията SignalR как тя работи, функционира и успява да осъществи двупосочна постоянна комуникация между сървър и клиент. Ще разгледаме как се имплементира .NET SignalR в платформата за споделен екран.

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.

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 OWIN[33] Startup клас.

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

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.

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.

Щом се изпълни метода “**OnConnected**” това означава, че клиентската част (JavaScript) успешно е направила връзката с нашия SignalR Hub. В API Контролера 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.

---

[31] **CORS** = Cross-Origin Resource Sharing - политика със същия произход

[32] **in-memory** = база данни, съхранявана в оперативната памет.

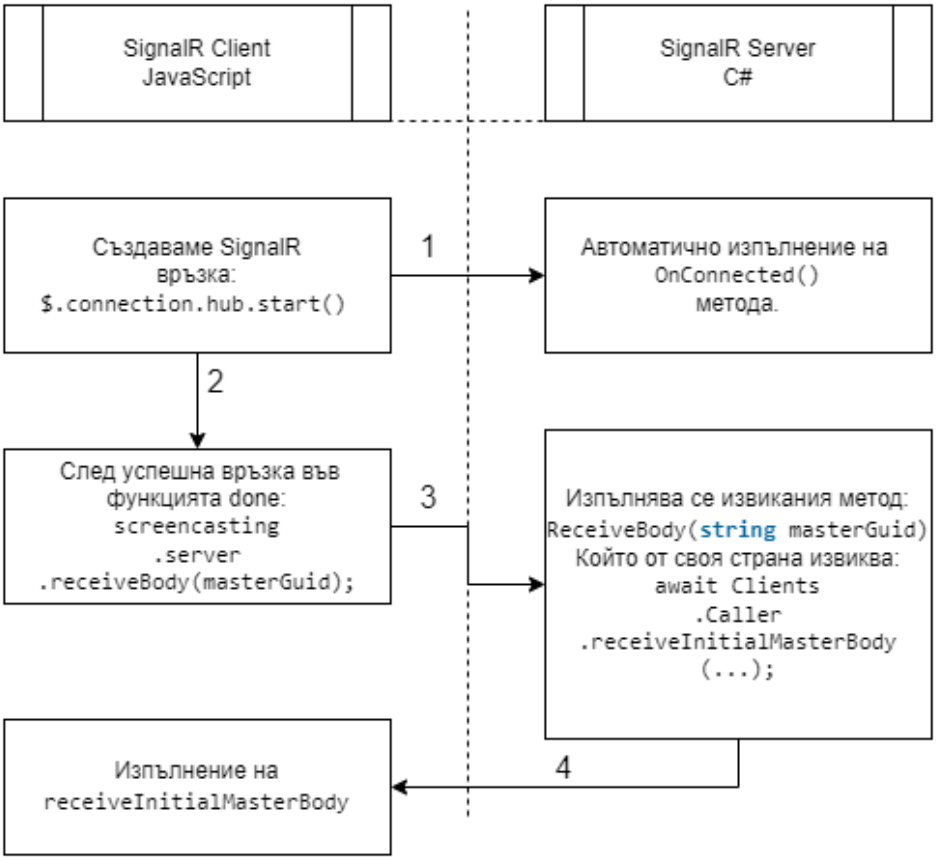
[33] **OWIN** = Open Web Interface for .NET, OWIN позволява на уеб приложенията да бъдат отделени от уеб сървърите. Той дефинира стандартен начин за използване на междинен софтуер в конвейер за обработка на заявки и сървърни отговори (request, response).

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.
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.

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.
За успешното създаване на тази крайна точка създаваме изглед.
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.
В този екстракт от код използваме Razor изглед, 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, които ще използвам за Slave JavaScript функционалността.
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”.
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.
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 сървъра повика и потърси такава със съвпадащо име.
Ред единадесети отваря връзка със SignalR, като след успешна установена връзка на сървъра се изпълнява “OnConnected” метода. 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. 12. Ред на изпълнение на функции и методи между клиентска и сървърна част на SignalR

Имплементация на Master/клиент функционалност. Примерен TestClient проект.

Тук 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.

За по-бързото построяване на страницата използваме bootsrap 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.

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”.
В тази функция правим заявка до backend частта и контролера. В случай на успешна заявка и отговор от сървъра се изпълнява функцията “onMasterGuidGenerated”, която е една от най-основополагащите за платформата.
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 двадести и седми правим директно повикване на метод с името “saveInitialMasterData”, които се намира на сървъра. Дадения метод запазва всички данни за текущия клиент.

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...”
След стартиране на проектите трябва да предприемем няколко стъпки. Започваме сесия за споделен екран след натискане на бутон, след което ни се генерира уникалния мрежови адрес, които 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”.
Проверявайки глобалнота 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.
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.
Друго важно, което искам да демонстрирам е, че използваме и функции, чиято цел е изцяло спомагателна, както например е “*serializeInputs*”.

Тази функция е с по-голямо съдържание и цялата може да бъда намерена в GitHub, тъй като платформата за споделен екран 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 сървъра. Като входящ параметър приема целия HTML, които 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.
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.
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.
Аналогично на по-горния слушател на събития, имаме същата логика. Взимаме всички потребители/slaves за даден клиент/master и извикваме съответните методи при тях.
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.
Това е метода, който се изпълнява на сървъра след като клиент промени своя размер на страницата.
Показахме 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 пренадлежат 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.
Ако следваме инструкциите за разработка на платформата, 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.

Заключение

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 браузъра 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 браузър 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.

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 Complience: 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/