**ТЕХНОЛОГИЧНО УЧИЛИЩЕ “ЕЛЕКТРОННИ СИСТЕМИ”**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: ERP за сервизна дейност

Дипломант: Научен ръководител:

*Александър Маринов Мартин Тасков*

СОФИЯ

2020

Дата на заданието: 15.11.2019 г. Утвърждавам:..............................

Дата на предаване: 15.02.2020 г. /проф. д-р инж. Т. Василева/

**ЗАДАНИЕ**

**за дипломна работа**

на ученика Александър Иванов Маринов 12A клас

1.Тема: ERP за сервизна дейност

2.Изисквания:

Възможност за създаване, редактиране, обновяване и изтриване на поръчки; история за всеки ремонт с поръчаните части, текстово описание и снимки; вход с сесия един час неактивност с пращане на код за потвърждение след изтичането на сесията; сортиране по няколко колони; четене на информация от снимка; качване на снимки на дефекта; правене на справки за определен период; Сайт с дизайн за компютри и телефони; история на ремонтите на всеки клиент; онлайн плащане на ремонта; Доставка с търсене на адрес и визуализация с google maps; Генериране на pdf фактура или printer friendly страница и изпращане на email; Интеграция със спедиторско API

3.Съдържание 3.1 Обзор

3.2 Същинска част

3.3 Приложение

Дипломант :...........................................

Ръководител:..........................................

/ Мартин Тасков /

Директор:................................................

/ доц. д-р инж. Ст. Стефанова /

**УВОД**

ERP представлява интегрирана софтуерна информационна система, която се грижи за планирането и управлението на всички ресурси в едно. Системата има за цел да обхване всички звена и бизнес процеси от живота на една компания.

Интегрирана система е тази, в която информацията се въвежда веднъж и без допълнителна обработка (в реално време) се отразява във всички модули и подсистеми, като става достъпна за всички потребители.

ERP замества традиционните информационни системи, като ги обединява в единна, унифицирана програма, разделена на взаимно свързани модули според отделните сектори в едно предприятие. Предназначена е за автоматизиране на управлението и отчетността.

Повечето системи от този тип разрешават имплементацията на отделни модули, покриващи един или повече бизнес процеси. В дипломната си работа съм създал само модулите свързани със сервизната дейност по приемането, ремонта и издаването на дефектирал уред, както и управлението на складовите наличностти.

Реших да създам този сервизен модул за гаранционно и извън гаранционно поддържане и ремонт на бяла и черна техника, защото след мое проучване на пазара на подобни продукти и разговори с хора от бранша се оказа, че се предлагат основно два вида софтуер – с много добри параметри и функционалност, но с висока цена или с по-ниска цена, но не достатъчно функционални или пък много тромави. Останалите модули от ERP системата не са предмет на разработката ми, поради следните причини: нямам достатъчно знания в съответните области; в основата си тези модули са стандартни/ напр. счетоводем модул/ или типа дейност, която покрива модула, е по-изгодно да бъде аутсорсната към външна фирма / напр. куриерски фирма/.

Създаденият от мен продукт е насочен основно към малкия и среден бизнес, тъй като считам, че той ще се търпи развитие в следващите няколко години/ особено в районите с по-малобройно население/. Модулите са изградени на основата на едни от най-новите софтуерни продукт.

Целта им е да се автоматизира създаването на ремонтна карта, като се избегнат грешките при попълване на модел, сериен номер и продуктов код/ IMEI за дефектиралите уреди. Също така максимално опростения му, но с достатъчна функционалност за този вид дейност, дизайн ще доведе до повишаване ефективността на персонала, опростяване на административните дейности, извършвани от техниците, което от своя страна ще доведе до повишаване на ефективността им при ремонта на уредите.

Системата позволява към нея да бъдат прикачени и допълнителни модули, в зависимост от изискванията на съответния клиент, но това ще доведе до оскъпяване на продукта като цяло.

Информацията за клиентите са защитени от секюрити модул, разработен от мен, в съответствие с европейските принципи за защита на личните данни.

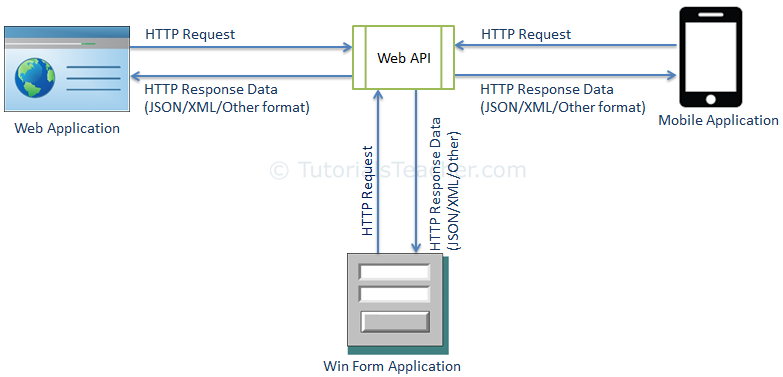
**ПЪРВА ГЛАВА**

Преглед на използваните технологии и съществуващи решения

* 1. Обзор на използваните технологии
     1. ASP.NET Web API

API (Application Programming Interface) е тип интерфейс, който има набор от функции, които позволяват достъп до специфични свойства или данни на приложение, операционна система или други сървиси. Web API както името показва, е API, което може да бъде достъпено чрез HTTP протокола.

ASP.NET Web API е extensible framework за изграждане на HTTP базирани сървиси, които могат да бъдат достъпени от различни приложения на различни платформи като уеб, уиндоус, мобилни и други. Принципа на действие е показан на фиг. 1.1.



фиг. 1.1

* + 1. SignalR

SignalR е библиотека с отворен код, която опростява процеса по добавяне на финкционалности към дадено приложение в реално време (real – time web functionality (RTWF)) и е направена, за да бъде използвана от разработчици на ASP.NET. RTWF е способността на даден сървър да качва ново съдържание в приложението и то да става достъпно веднага за всички свързани клиенти вместо клиента да прави нов request за данни.

SignalR може да бъде използвана за добавяне на всякакъв тип RTWF към вече съществуващо ASP.NET приложение. Най – често давания пример за това е чат приложение, но може да бъде направено много повече. Вески път, когато потребител обнови уеб страница, за да види нови данни, или страницата имплементира long polling, за да вземе нови данни, това е кандидат за използването на SignalR.

SignalR също така предоставя възможността за напълно нов тип уеб приложения, при които се налага обновявания много често от сървъра като например real-time gaming.

SignalR предоставя просто API, което създава server-to-client remote procedure calls (RPC), което извиква JavaScript функции в клиентските браузъри от server-side .NET code. SignalR също така съдържа API за управление на връзката и връзка между групи. Принципът на действие е показан на фиг. 1.2.



фиг. 1.2

SignalR handles connection management automatically, а това позволява на данните да бъдат разпрострени до всички клиенти едновременно, както и до специфичен клиент. Връзката между сървъра и клиента е постоянна, за разлика от класическата HTTP връзка, която трябва да се подновява за всяка комуникация.

SignalR поддържа “server push“ functionality, при която кодът в сървъра може да бъде извикан от кода при клиента в браузъра, използвайки RPC вместо request-response модела, който може да се види във всеки сайт днес. Тази бибиотека е вградена по default в по – долу описаната технология.

* + 1. Blazor

Blazor е фреймуърк за създаване на интерактивен client-side web UI, използвайки .NET: създаване на интерактивни UIs, използвайки C# вместо JavaScript (JS); server-side and client-side логика на приложението, написана на .NET; render UI като HTML и CSS и поддрежка за много браузъри, включително мобилни.

Използването на .NET за client-side web development предоставя следните предимства:

* Писане на C# вместо на JavaScript
* Използва съществуващата .NET екосистема от .NET библиотеки
* Share app logic across server and client
* Предимствата на производителността, надежността и сигурността на .NET.

Типовете проекти, който могат да се изграждат с този фреймуърк, са два: Blazor Server App и Blazor WebAssembly, но първо ще бъде обяснено какво е Razor Component.

* + - 1. Razor Components (RC)

Blazor приложенията са баизрани на компоненти. Компонент в Blazor елемент от UI като страница, изскачащо прозорче или форма за попълване на данни.

Компонент са .NET класове, компилирани в .NET assemblies, които дефинират flexible UI rendering logic, обработват user events, могат да бъдат вложени и преизползвани.

Класът на компонентите обикновено се записва под формата на страница за маркиране на Razor с разширение на файла .razor. Компонентите в Blazor официално се наричат Razor Components. Razor е синтаксис за комбиниране на HTML маркиране с C # код, предназначен за производителност на разработчиците. Razor Pages и MVC също използват Razor. За разлика от Razor Pages и MVC, които са изградени около модела на заявка – отговор, компонентите се използват специално за логиката и състава от страна на клиента от потребителския интерфейс.

Фиг. 1.3 демонстрира компонент (Dialog.razor), който може да бъде вложен в друг компонент:

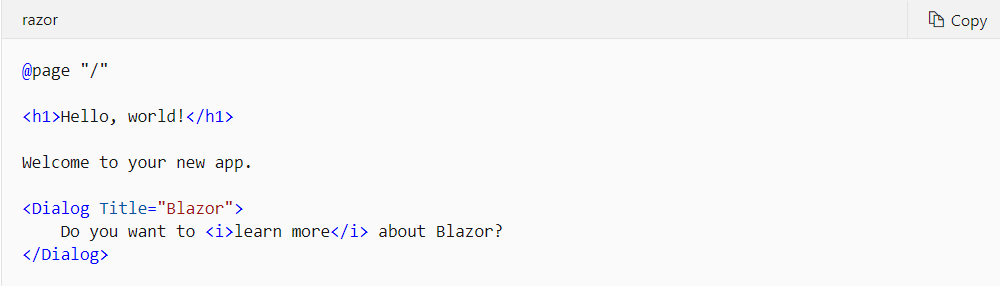


фиг. 1.3

Съдържанието на тялото на диалога (ChildContent) и заглавието (Title) се предоставят от компонента, който използва този компонент в своя потребителски интерфейс. OnYes е метод на C #, задействан от събитието onclick на бутона.

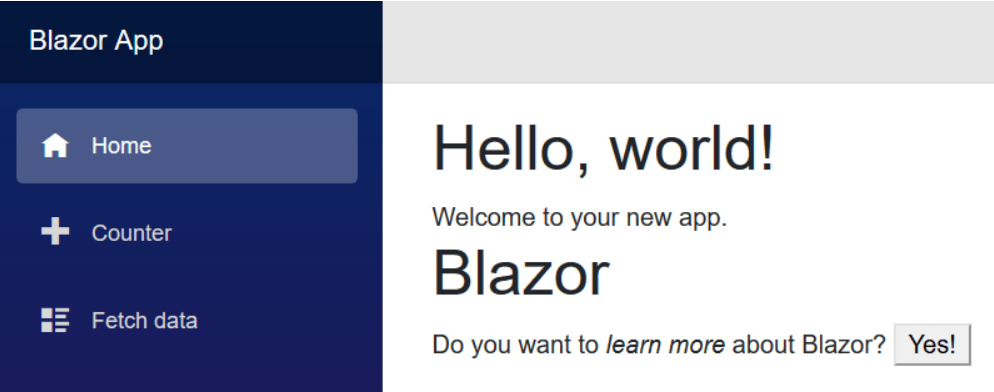
Blazor използва естествени HTML тагове за UI композиция. HTML елементите определят компоненти, а атрибутите предават стойности на свойствата на компонента.

В следващия пример компонентът Index използва компонента Dialog. ChildContent и Title се задават от атрибутите и съдържанието на елемента <Dialog> (фиг 1.4).



фиг. 1.4

Диалогът се визуализира, когато родителят (Index.razor) бъде достъпен в браузъра (фиг. 1.5)



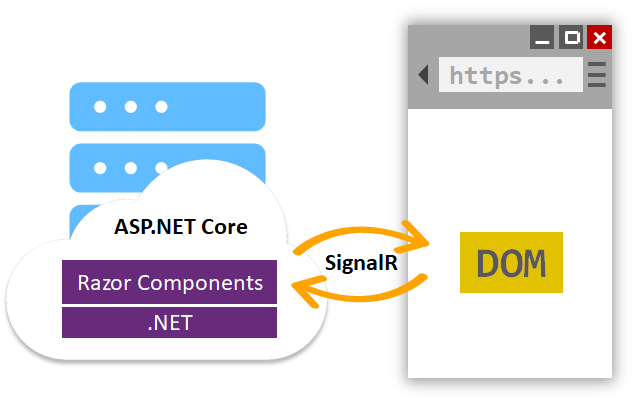
фиг. 1.5

* + - 1. JavaScript interop

Приложения, при които се налага използването на third-party JavaScript libraries и достъп до APIs в браузъра, компонентите си взаимодействат с JS. Компонентите имат способността да използват всяка библиотека или API, което JS е способен да използва. C# код може да извиква JS код, както и JS код да извиква C# код.

Типовете проекти, който могат да се изграждат с този фреймуърк, са два: Blazor Server App и Blazor WebAssembly.

* + - 1. Blazor Server App (BSA)

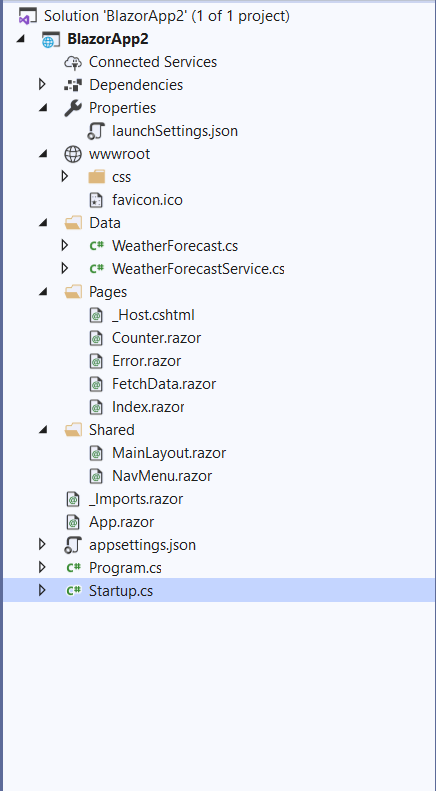


фиг. 1.6

На фиг. 1.6 е показано на как работи едно BSA приложение. Blazor отделя component rendering logic от начина на прилагане на актуализациите на UI. Blazor Server предоставя поддръжка за хостинг на Razor компоненти на сървъра в ASP.NET Core приложението. Актуализациите на UI се обработват чрез SignalR connection.

The runtime обработва изпращането на UI събития от браузъра до сървъра и прилага актуализации на UI, изпратени от сървъра обратно към браузъра след стартиране на компонентите.

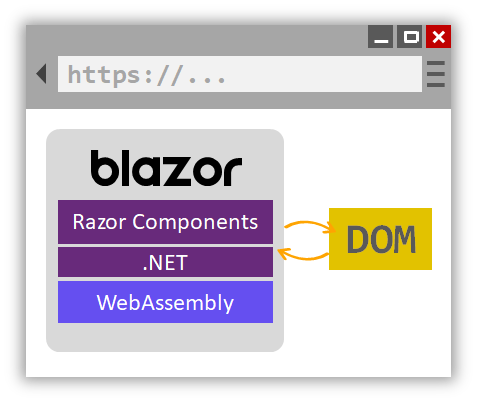
Връзката, използвана от BSA за комуникация с браузъра, се използва и за обработка на JS interop calls.



фиг. 1.7

На фиг. 1.7 е показано каква е структурата на едно BSA приложение. То е разделено на следните части:

* wwwroot – в нея се съдържа всичкия css, js, icons и т.н., които са използвани в приложението
* Data – там се съдържат всички модели и сървиси, а сървисите трябва да се добавят в Startup.cs.
* Pages – в нея се съдържат всички страници, което е BSA приложение е възможно да има и те са .razor файлове, които са обяснени как работят по – нагоре. \_Host.cshtml е Razor Page, в която се добавят всички връзки към css и js файллове.
* Shared – в нея се съдържат файловете NavMenu.razor и MainLayout.razor. В NavMenu.razor се пишат всички елементи на менюто. В MainLayout.razor се пише как ще изглежда основната страница на приложението.
* Startup.cs и Program.cs – съответно съдържа конфигурацията на цялото приложение и компилира цялото приложение.
  + - 1. Blazor WebAssembly (Wasm)



фиг. 1.8

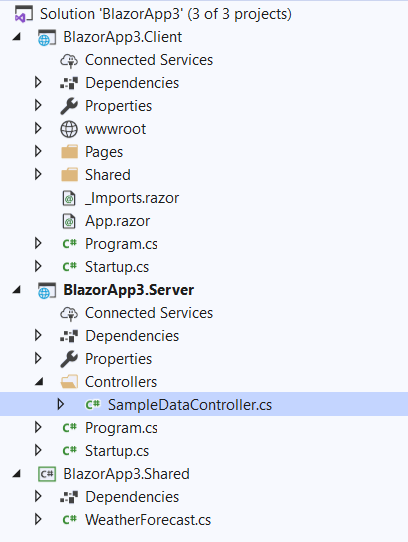
На фиг. 1.8 е показано как работи едно Wasm приложение. Blazor WebAssembly е SPA framework за изграждане на интерактивни client-side уеб приложения с .NET. Blazor WebAssembly използва отворени уеб стандарти без плъгини или преобразуване на код и работи във всички съвременни уеб браузъри, включително мобилни браузъри.

Изпълнението на .NET код в уеб браузърите става възможно от WebAssembly (съкратено Wasm). WebAssembly е компактен bytecode формат, оптимизиран за бързо изтегляне и максимална скорост на изпълнение. WebAssembly е отворен уеб стандарт и се поддържа в уеб браузъри без plugins.

Кодът на WebAssembly може да има достъп до пълната функционалност на браузъра чрез JS, наречен оперативна съвместимост на JavaScript (или JS interop). .NET код, изпълнен чрез WebAssembly в браузъра, работи в JavaScript sandbox на браузъра със защитите, които sandbox осигурява срещу злонамерени действия на клиентската машина.

Когато Wasm приложението е компилирано и работещо в браузъра:

* C# и Razor файловете се компилират в .NET assemblies
* The assemblies and the .NET runtime се свалят в браузъра
* Wasm стартира .NET runtime и конфигурира runtime, за да зареди монтажите за приложението. Wasm runtime използва JS interop за манипулация DOM дървото и извиквания на API в браузъра



фиг. 1.9

На 1.9 e показано каква е структурата на едно Wasm приложение. То се състои от три модула:

* Client – в него се съдържат целия front-end на приложението.
* Server – в него се съдържа цялата back-end логика
* Shared – в него се съдържа всички модели на приложението

В .NET Core 3.1 Wasm не е налично като тип проект, защото дава грешки, които са от .NET Core 3.

* + 1. Google Firebase

Google Firebase е Google – backed софтуер за разработка на приложения, който дава възможност на разработчиците да разработват приложения за iOS, Android и Web. Firebase предоставя инструменти за проследяване на аналитиката, отчитане и коригиране на сривове на приложения, създаване на маркетинг и експеримент с продукти.

Firebase предлага редица услуги, включително:

* Analytics – Google Analytics за Firebase предлага безплатно, неограничено отчитане за повече от 500 отделни събития. Analytics представя данни за поведението на потребителите в приложения за iOS и Android, което позволява по-добро вземане на решения за подобряване на производителността и маркетинга на приложенията.
* Authentication – Firebase Authentication улеснява разработчиците при изграждането на защитени системи за authentication и подобрява практиката на влизане и вграждане за потребителите. Тази функция предлага цялостно решение за идентичност, поддържащо имейл акаунти и акаунти за парола, оторизиране на телефона, както и Google, Facebook, GitHub, вход в Twitter и други. Този продукт на Firebase е използван в настоящата дипломна работа.
* Crashlytics – Firebase Crashlytics е real-time crash reporter, който помага на разработчиците да проследяват, дават приоритет и да коригират проблеми със стабилността, които намаляват качеството на техните приложения. С Crashlytics разработчиците отделят по-малко време за организиране и отстраняване на неизправности и повече функции за изграждане на време за своите приложения.
* Realtime database – Firebase Realtime database e cloud-hosted NoSQL база данни, която позволява съхраняването и синхронизирането на данни между потребителите в реално време. Данните се синхронизират за всички клиенти в реално време и все още са достъпни, когато приложението е офлайн.
* Cloud Firestore – Cloud Firestore е flexible, scalable база данни за разработка на мобилни устройства, уеб и сървър от Firebase и Google Cloud Platform. Подобно на Firebase Realtime Database, тя поддържа данните синхронизарани между клиентските приложения чрез realtime listeners и предлага офлайн поддръжка за мобилни и уеб, за да можете да се изграждат отзивчиви приложения, които работят независимо от латентността на мрежата или интернет връзката. Cloud Firestore също предлага безпроблемна интеграция с други продукти на Firebase и Google Cloud Platform, включително Cloud Functions. Този продукт на Firebase е използван в настоящата дипломна работа.
  + 1. Google Cloud Platform (GCP)
       1. Cloud Computing (CC)

CC, казано накратко, е достъп до и съхраняване на данни през интернет, вместо да бъдат съхранявани на твърд диск.

Предлага услуги като съхранение, база данни, работа в мрежа и повече през Интернет, за да предостави по – бързи, иновативни и flexible resources на своите клиенти. Клиентите плащат само за ресурсите, които използват, като по този начин им помага да намалят оперативните си разходи и да управляват бизнес инфраструктурата си по – ефективно.

* + - 1. GCP

GCP е набор от Cloud Computing сървиси, предлагани от Google. Платформата предоставя различни услуги като изчисляване, съхранение, работа в мрежа, големи данни и много други, които работят на същата инфраструктура, която Google използва вътрешно за своите крайни потребители като Google Търсене и YouTube.

GCP предлага голям набор от функции, включително:

* Compute Services
  + Google App Engine – платформа като услуга за deploy на Java, PHP и други приложения. Това е Cloud Computing платформа за разработване и хостинг на уеб приложения в Google-managed data centers. Предлага функция за автоматично мащабиране, т.е. с увеличаването на броя на заявките за приложение, App Engine автоматично разпределя повече ресурси за приложението, за да се справи с допълнителното търсене.
  + Compute Engine – инфраструктура като услуга за стартиране на виртуални машини на Microsoft Windows и Linux. Това е компонент на платформата Google Cloud, която е изградена на същата инфраструктура, която управлява Google търсачката, YouTube и други услуги. Compute Engine е използван в настоящата дипломна работа.
  + Kubernetes Engine: Той има за цел да предостави платформа за автоматизиране на deployment, scaling и операции с контейнери за приложения в клъстери от хостове. Работи с широк набор от инструменти за контейнери, включително Docker.
* Storage Services
  + Google Cloud Storage – интернет услуга за съхранение на файлове в Интернет за съхранение и достъп до данни в инфраструктура на GCP. Услугата съчетава ефективността и мащабируемостта на Google Cloud with advanced security and sharing capabilities.
  + Cloud SQL – уеб услуга, която позволява създаването, конфигурирате и използването на релационни бази данни, които живеят в Google Cloud. Поддържа, управлява и администрира базите данни.
  + Cloud Bigtable - бърза, напълно управлявана и високо мащабируема NoSQL database service. Той е предназначен за събиране и задържане на данни от 1 TB до стотици PB.

**ВТОРА ГЛАВА**

Изисквания към програмния продукт, аргументация за използваните технологии и структура на приложението

* 1. Аргументация за използваните технологии
     1. Blazor

Blazor е SPA фреймуърк за разработване на уеб приложения. В настоящата дипломна работа е избран поради следните причини:

* Напълно съвместим е с .NET libraries и .NET tooling. За разлика от client-side Blazor (Wasm), който се предлага с непълни възможности за отстраняване на грешки, грешките на server-side Blazor могат да бъдат отстранявани по абсолютно същия начин като всяко друго ASP.NET приложение. По същия начин, всички други стандартни инструменти, които могат да бъдат използвани с всякакви други видове .NET приложения, могат да се използват с server-side Blazor.
* Използва точно същия синтаксис като client-side Blazor(Wasm). Въпреки че Blazor от страна на сървъра може да не е най-добрият избор за определени сценарии, когато client-side Blazor е полезен, все още първоначално приложението да се напише на server-side Blazor, за да се преодолеят ограничените възможности за отстраняване на грешки на client-side Blazor. След като приложението е готово за release, кодът от приложението се копира в проекта, който е clients-side Blazor. Синтаксисът позволява да бъдат написани class libraries, които ще работят с server-side Blazor или client-side Blazor.
* Малък размер на клиентски компоненти. Когато става дума за изтегляне на клиентски компоненти на server-side Blazor, те ще включват само сравнително малко количество HTML и JavaScript, нито един от които не е задължително да бъде написа.
* Works with thin clients. Server-side Blazor не го интересува в кой браузър се показва. Клиентът получава само стандартни HTML и JavaScript, които работят практически навсякъде. Следователно server-side Blazor e съвместим с почти всеки браузър.
  + 1. Web API

В настоящата дипломна работа ASP.NET Web API е избрано, защото Blazor WebAssembly(Wasm) все още е в beta версия и не е напълно изправно. Затова чрез Blazor server app и Web API се симулира Wasm.

* + 1. Google Firebase
       1. Cloud Firestore

Google Firebase Cloud Firestore е пуснат официално през 2018 година като част от Google Firebase Platform. Това е тип база данни, която е хоринзонтално нарастваща, тоест

* + - 1. Authentication

**Съдържание**

**Използвана литература**

* Описание на ASP.NET Web API – <https://www.tutorialsteacher.com/webapi/what-is-web-api>
* Описание на SignalR – <https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-3.1>
* Описание на Blazor – <https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-3.1>
* Опиcание на Firebase – <https://searchmobilecomputing.techtarget.com/definition/Google-Firebase>
* Описание на Cloud Firestore – <https://firebase.google.com/docs/firestore>
* Описание на Google Cloud Platfrom – <https://intellipaat.com/blog/what-is-google-cloud/#cloud-services>