Olphen,Bas S.H.N. van

shnvanolphen@gmail.com

Een onderzoek voor semester 7

Zouden wij als bedrijf onze nieuwe front-end applicaties moeten maken met Blazor?



# Voorwoord

Voor semester 7 mocht ik zelf kiezen over welk onderwerp ik een onderzoek wilde doen. Al snel kwam ik, vanwege terugkerende problemen rondom onze VSR KMS applicatie, op het idee te gaan onderzoeken of het mogelijk zou zijn deze app te herschrijven in Blazor. Ik wilde graag onderzoeken of het mogelijk was om als developer zonder front-end kennis, een web-based front-end applicatie te maken.

Door middel van zorgvuldig gekozen deelvragen ben ik dit onderzoek gestart en ben ik uiteindelijk tot een conclusie gekomen. Door het feit dat deze conclusie door middel van goed onderzoek tot stand is gekomen, zal dit binnen ons bedrijf tot doorslaggevend advies zorgen betreft het ontwikkelen van de huidige en nieuwe applicaties.

Inhoudsopgave

[Voorwoord 1](#_Toc122462340)

[1.1 Welke methodes van opslaan voor data kent Blazor? 4](#_Toc122462341)

[Doel 4](#_Toc122462342)

[Strategie 4](#_Toc122462343)

[Uitwerking 4](#_Toc122462344)

[LocalStorage 5](#_Toc122462345)

[SessionStorage 5](#_Toc122462346)

[Conclusie LocalStorage en SessionStorage 5](#_Toc122462347)

[IndexedDB 5](#_Toc122462348)

[TG.Blazor.IndexedDB 6](#_Toc122462349)

[BlazorIndexedDB 6](#_Toc122462350)

[Werken met meerdere geopende tabbladen 8](#_Toc122462351)

[Verder werken na sluiten browser 8](#_Toc122462352)

[Kan de IndexedDB de gebruikelijke payload aan data aan? 8](#_Toc122462353)

[Conclusie 8](#_Toc122462354)

[1.2 Welke vaardigheden zijn er nodig voor het maken van deze applicatie? 9](#_Toc122462355)

[Doel 9](#_Toc122462356)

[Strategie 9](#_Toc122462357)

[Uitwerking 9](#_Toc122462358)

[Ontwerpen 9](#_Toc122462359)

[Uitwerking 12](#_Toc122462360)

[Pagina’s aanmaken binnen de app 13](#_Toc122462361)

[Inline Razor en Databinding 13](#_Toc122462362)

[Components 15](#_Toc122462363)

[Conclusie 15](#_Toc122462364)

[Voorlopig advies 15](#_Toc122462365)

[1.3 Kan de gemaakte app in cache draaien, zodat er offline gewerkt kan worden? 17](#_Toc122462366)

[Doel 17](#_Toc122462367)

[Strategie 17](#_Toc122462368)

[Uitwerking 17](#_Toc122462369)

[Conclusie 19](#_Toc122462370)

[1.4 Heeft Blazor een toekomst? 19](#_Toc122462371)

[Doel 19](#_Toc122462372)

[Strategie 19](#_Toc122462373)

[Uitwerking 19](#_Toc122462374)

[Wat zeggen anderen? 20](#_Toc122462375)

[Conclusie 22](#_Toc122462376)

[1.5 Welke tooling en methodes kunnen er gebruikt worden van het schrijven van (automatische) tests? 22](#_Toc122462377)

[Doel 23](#_Toc122462378)

[Strategie 23](#_Toc122462379)

[Uitwerking 23](#_Toc122462380)

[Welke technieken zijn beschikbaar? 23](#_Toc122462381)

[Unit Tests 23](#_Toc122462382)

[E2E 24](#_Toc122462383)

[Wat is de criteria van deze techniek? 25](#_Toc122462384)

[Uitwerking gekozen methodieken 26](#_Toc122462385)

[bUnit 26](#_Toc122462386)

[Playwright for .NET 26](#_Toc122462387)

[Conclusie 27](#_Toc122462388)

[bUnit 27](#_Toc122462389)

[Playwright for .NET 27](#_Toc122462390)

[Algemene conclusie 28](#_Toc122462391)

[Bijlages 29](#_Toc122462392)

[Bronnen 30](#_Toc122462393)

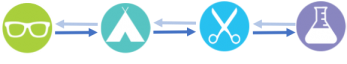
# 1.1 Welke methodes van opslaan voor data kent Blazor?

## Doel

Het doel van dit onderzoek is het vinden van een geschikte manier voor het opslaan van data binnen de app. Waarbij er gedacht moet worden aan gebruiksvriendelijke en toekomstbestendige oplossing.

## Strategie

Zoals in het projectplan beschreven ga ik dit onderzoek doen volgens het “Choose fitting” pattern.



Choose fitting pattern 1

Dit betekend dat ik volgens de volgende volgorde ga werken:

1. Library: Literature study & community research

2. Field: Explore (user)requirements

3. Lab: Usability testing

4. Workshop: prototyping

## Uitwerking

Voor het uitvoeren van dit onderzoek heb ik eerst verschillende documentaties bestudeerd. Er zijn twee omgevingen waar data opgeslagen kan worden; server-side storage en de client-storage, in het geval van Blazor de webbrowser. Zoals ik al eerder in het projectplan aangegeven had, is het voor deze app van belang dat data opgeslagen kan worden op de client, zodat de inspecteur zonder internetverbinding gebruik kan blijven maken van de app. Hierbij wordt dus meteen de server-side storage uitgesloten, aangezien er dan een constante verbinding moet zijn, welke er in ons geval niet is.

Nu de keuze voor client-side storage is gemaakt gaan we kijken welke mogelijkheden er hiervoor zijn. Het idee is om eerst een beeld te krijgen welke mogelijkheden welke functionaliteiten hebben. Vervolgens wil ik van de gekozen methodes elk een demo maken, waarin er een bulk data van het bestaande backend wordt gedownload. Vervolgens wil ik die data kunnen manipuleren, waarna ik er een aantal tests mee uit ga voeren. Zoals de browser sluiten, nieuwe tabs openen naar dezelfde URL etc. Het idee daarachter is dat ik dan snel een beeld krijg of een gekozen methode effectief is voor hetgeen wat er in de toekomst mee gedaan gaat worden.

Volgens de ontwikkelaar van Blazor, Microsoft, zijn er twee voorname methodes.

### LocalStorage

LocalStorage is gebonden aan de browser zelf. Wanneer een gebruiker de pagina reload of opnieuw opent blijft de opgeslagen data en state bestaan. Wanneer een gebruiker een nieuw tabblad zou opnen, is diezelfde data ook vanuit dit nieuwe tabblad te gebruiken.

### SessionStorage

SessionStorage is gebonden aan de tab waar de gebruiker zich in bevind. Wanneer de gebruiker zijn pagina refreshed blijft de state bestaan. Wanneer de tab gesloten wordt is alles kwijt. Dit lijkt op voorhand al niet een geschikte kandidaat. Het is al bij voorbaat niet uit te sluiten dat een gebruiker altijd maar een tabblad geopend heeft, en al helemaal niet dat deze nooit gesloten wordt.

### Conclusie LocalStorage en SessionStorage

Methode 1 valt af. De beperking dat er maximaal 5MB data opgeslagen kan worden zorgt ervoor dat dezer methode niet toereikend is voor het doel van de applicatie.

Mijn conclusie is dat mehode 2, de sessionstorage, niet in aanmerking komt voor hetgeen wat ik wil bereiken. Als voorbeeld kunnen gebruikers van de app wel eens pauze nemen. Wanneer ze de tab waar de app draait vervolgens sluiten is de app al zijn data kwijt. Dit is onaanvaardbaar en dus geen toereikende methode.

### IndexedDB

IndexedDB is een veelvuldig gebruikte Javascript API, gemaakt voor het opslaan van data binnen front-end applicaties. Ook wordt deze methode vaak gebruikt voor het opslaan van data binnen Blazor WASM applicaties. Vanwege dit veelvuldig gebruik binnen Blazor, zijn er ook NuGet packages te gebruiken, welke ervoor zorgen dat er een enkele regel javascript code geschreven hoeft te worden.

Bron: https://developer.mozilla.org/

*IndexedDB is a low-level API for client-side storage of significant amounts of structured data, including files/blobs. This API uses indexes to enable high-performance searches of this data. While Web Storage is useful for storing smaller amounts of data, it is less useful for storing larger amounts of structured data. IndexedDB provides a solution…*

### TG.Blazor.IndexedDB

Een van de meest besproken IndexedDB NuGet packages was TG.Blazor.IndexedDB, dit heeft er voor gezorgd dat ik meteen enthousiast was en een project heb opgezet om hier een prototype van te maken. Echter kreeg ik dit prototype niet aan de praat en kreeg ik niet snel in de gaten waarom dit niet lukte. Na wat meer zoek werk binnen de GitHub repo van dit project kwam ik erachter dat dit project “deprecated” was, iets wat ik had kunnen voorkomen door vooraf beter onderzoek te doen.

Afgezien van het feit dat deze package niet meer werkte, was het geen verloren tijd. Door deze les heb ik geleerd nauwkeuriger onderzoek te doen, vooral bij het kiezen van oplossing binnen de software. Dit heeft dan ook bijgedragen aan mijn “judgement” leeruitkomst.

### BlazorIndexedDB

BlazorIndexedDB is een doorstart op TG.Blazor.IndexedDB. Bij deze package hebben de developers hetgeen wat er voor zorgde dat TG.Blazor.IndexedDB niet meer compatible was met de laatste .NET versies er uit geschreven. Daarbij is te zien dat het originele project erg goed onderhouden project is, waar veel activiteit plaatsvindt en heeft gevonden en een hoog aantal downloads en commits heeft. Ook is de code open-source en erg goed te begrijpen. Dit betekent voor mij dat deze package een nagenoeg oneindige levensduur heeft, aangezien wij hem zelf zouden kunnen onderhouden mocht deze in de toekomst niet meer naar behoren werken.

Ook is de implementatie van deze package erg simpel, zoals te zien in onderstaande screenprints.

Om te beginnen wordt er een Singleton instance van de IndexedDBFactory interface aan de services van mijn app toegevoegd.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Toevoegen Singleton intance

Er moet een datastore gedefinieerd worden in program.cs. De datastore wordt vervolgens aan het IndexedDB object toegevoegd, zodat deze vervolgens als object te benaderen is vanuit mijn app.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Definieeren database object

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Configuratie datastore(database)

Wanneer deze drie stappen zijn gemaakt, kan er direct gebruik worden gemaakt van de IndexedDB database. En zoals te zien komt daar geen regel Javascript code aan te pas, zo hoeven de developers die hier mee aan de slag gaan geen Javascript kennis te hebben.

Ook het ophalen, aanpassen, opslaan en verwijderen van data is eenvoudig te realiseren met enkele regels code, zoals te zien in onderstaande screenprint.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Ophalen, aanpassen en verwijderen

### Werken met meerdere geopende tabbladen

Een belangrijke requirement voor het kiezen van een methode voor het opslaan van data, is dat de data vanuit alle mogelijk geopende tabs bereikbaar moet zijn. De data moet op een centraal punt opgeslagen zijn, zodat het niet uit maakt vanaf welke tab de data aangesproken wordt.

De conclusie van deze requirment is, is dat dit mogelijk is met BlazorIndexedDB. Zoals te zien in het bijgeleverde testdocument is deze requirement getest en afgevinkt.

### Verder werken na sluiten browser

Zoals ik eerder aangegeven heb, is het belangrijk dat de gebruiker de browser kan sluiten zonder dat dit effect heeft op de opgeslagen data. Dit is van cruciaal belang bij het kiezen van een methode en de conclusie hiervan zal dus doorslaggevend zijn.

Zoals te lezen in het bijgeleverde testdocument is ook deze requirement goed bevonden, de gebruiker kan op elk moment de browser sluiten zonder dat de applicatie data verloren raakt.

### Kan de IndexedDB de gebruikelijke payload aan data aan?

Ook is het van cruciaal belang dat er genoeg data opgeslagen kan worden op de device van de gebruiker. Een meting bevat van 0.05MB tot 10MB aan data, afhankelijk van of er wel of geen plattegronden met de meting worden meegestuurd. Om te testen of deze methode genoeg data kan opslaan heb ik een test opgezet, ook deze test is te lezen in het testdocument.

De conclusie van deze test is dat het normaal gebruik van deze applicatie niet leidt tot het bereiken van de limiet aan dataopslag.

## Conclusie

Ik ben erg tevreden over deze package, des te meer omdat de code hiervan is vrijgegeven volgens het opensource principe. Ook de code die vrijgegeven is, is makkelijk te begrijpen en te onderhouden.

Daarbij zijn ook alle requirements door middel van de opgestelde tests afgevinkt, wat wilt zeggen dat deze methode op dat gebied geschikt is voor gebruik in onze app.

Ook ben ik mij ervan bewust dat ik in de eerste instantie te snel met een methode in zee ben gegaan. Dit wil zeggen dat ik in mijn toekomstige onderzoeken meer aandacht moet besteden aan het vooraf doen van onderzoek. Gelukkig is dit geen verloren tijd gebleken, aangezien ik er een mooi leer punt uit heb gehaald.

# 1.2 Welke vaardigheden zijn er nodig voor het maken van deze applicatie?

## Doel

Het doel van deze deelvraag is het in kaart brengen van de vaardigheden die nodig zijn voor de re-maken van de huidige VSR-KMS applicatie in Blazor. De deelvraag is voor mij beantwoord wanneer ik, mede door het maken van een prototype, duidelijk in beeld heb gebracht wat erbij komt kijken. Dit betreft zowel het procesmatige als het technische aspect.

## Strategie

Om erachter te komen welke vaardigheden er nodig zijn voor het maken van de VSR KSM applicatie ligt, ga ik beginnen met een aanzet voor de VSR-KMS applicatie. Eerst ga ik door middel van het bekijken van documentatie en tutorials een beeld schetsen wat er nodig is voor het bouwen van een Blazor applicatie. Vervolgens ga ik de verzamelde informatie gebruiken om een prototype tot stand te brengen, waar ook daadwerkelijk een meting in gelopen kan worden.

## Uitwerking

### Ontwerpen

Voordat ik begin met het schrijven van de code, denk ik na over hoe de app er precies uit gaat zien. Ik wil het lekker strak en overzichtelijk houden, alleen de informatie die benodigd is, wordt getoond op het scherm. Daarbij komt er in tegenstelling tot de huidige app een strakke layout, die meegaat met de tijd. Na overleg met een aantal gebruikers heb ik voor elke pagina een schets gemaakt. Hier onder een voorbeeld van een schets van de foutenlijst.

Afbeelding met tafel

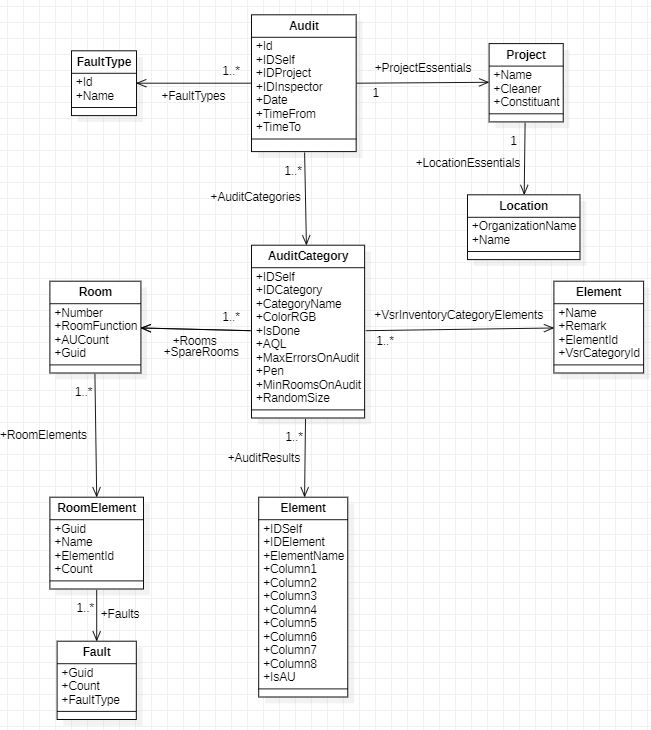
Automatisch gegenereerde beschrijving

Schets foutenlijst 1

Nadat ik alle schetsen voor de UI heb gemaakt, ben ik aan de slag gegaan met een klassendiagram. Ik ben de bestaande JSON, te zien in onderstaande screenshot, gaan bestuderen en heb gekeken welke data ik nodig had voor het maken van het prototype. Daarbij ben ik tot een klassendiagram gekomen, zoals te zien in afbeelding 3



Een screenshot van de JSON testdata



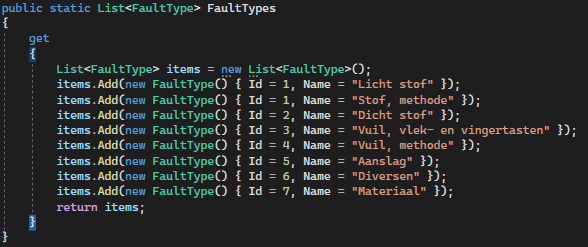
Screenshot datamodel 3

Tijdens het maken van de modellen kom ik al snel oude en slechte code tegen. Zo worden de foutsoorten op dit moment hardcoded in de front-end gegenereerd en dus niet vanuit de backend meegegeven, zoals het eigenlijk zou moeten. Dit heeft al die jaren goed kunnen gaan omdat er bij het VSR KMS systeem altijd dezelfde foutsoorten gebruikt worden. Echter wanneer er een nieuwe foutsoort bij zou komen, wat niet definitief uit te sluiten is, is de code niet meer bruikbaar. Aangezien ik de scope niet te groot wil maken, maar zelf niet de foutsoorten hardcoded wil aanmaken, maak ik er een class voor die de foutsoorten retourneert als een lijst, welke eenvoudig aan te passen is.



Foutsoorten hardcoded in de originele front-end

Wanneer deze applicatie in de toekomst gebruikt gaat worden, is het dan ook makkelijk om te bouwen naar de gewenste methode. Zo is het dan alleen nodig om de onderstaande list te vervangen met een in JSON meegegeven object.



Tijdelijke oplossing voor de hardcoded foutsoorten

## Uitwerking

Met Blazor kan men, zoals eerder aangegeven, front-end applicaties schrijven in C#. Een blazor applicatie bestaat uit verschillende componenten. Een component kan omschreven worden als UI element, bijvoorbeeld een pagina, dialoog, html-element of formulier. Een blazor component heeft altijd een .razor extensie, wat betekend dat er door middel van razor geprogrammeerd kan worden binnen een component. Met razor is het bijvoorbeeld mogelijk om in dezelfde regel code zowel html als C# te kunnen programmeren.

Ook is het mogelijk om met Blazor Javascript aan te roepen en andersom. Dit wordt gebruikt wanneer een Blazor applicatie toegang heeft tot bijvoorbeeld de browser APIs. Dit gebruik ik bijvoorbeeld met mijn IndexedDB API.

Blazor een index.html heeft. Dit is een normale html pagina, waar alle script en stylesheet imports gedefinieerd worden. Daar zit ook een body met id “app”, hierin worden alle Blazor components geladen. Hier wordt tijdens het ontwikkelen, op misschien het toevoegen van third-party scripts (in mijn geval IndexedDb.js), niets meer aan veranderd.



index.html

### Pagina’s aanmaken binnen de app

De VSR-KMS applicatie heb ik uiteindelijk gebouwd volgens mijn initiële schetsen. Voor elke schets (pagina) heb ik een blazor component aangemaakt. Tussen deze pagina’s valt erg makkelijk te navigeren, waarbij de history goed bijgehouden wordt. Zou kan de gebruiker, in tegenstelling tot de huidige app, wel standaard gebruik maken van de browser back- en forwardbutton. Dit was een grote ergernis van de gebruikers bij de huidige app en ik ben dan ook positief verrast dat hier geen zelfgeschreven code voor aan te pas hoeft te komen.

Om te navigeren tussen pagina’s dient er in het Blazor component een relatieve url gedefinieerd te worden. Bijvoorbeeld: @page "/Index"

Om vanaf elk willekeurige andere pagina naar de Index te navigeren dient er alleen NavManager.NavigateTo("/AuditMain"); te worden aangeroepen.

Dit betekent allemaal dat er geen kennis van browsers of kennis van het beheren van browser history meer nodig is, waar dit bij JQuery Mobile wel het geval was.

### Inline Razor en Databinding

Wat mij bij het maken ook positief verrast heeft, is het gemak van de in-line Razor mogelijkheden. Zo kan er, zoals in de onderstaande pagina van mijn prototype, makkelijk aangegeven worden of de tabel of de loadspinner weergegeven moet worden.

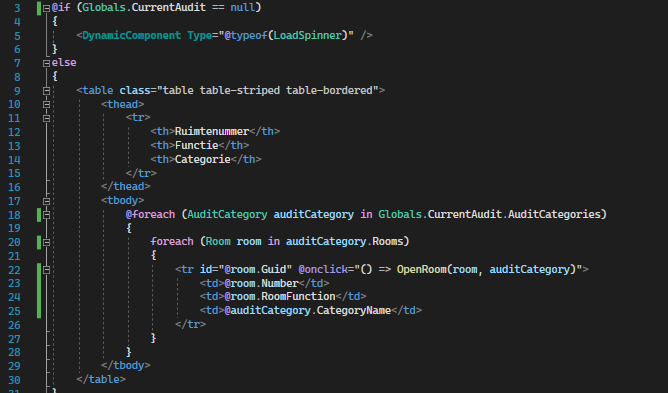
F

Figure 7: Voorbeeld in-line Razor

Wanneer Current audit null is, wordt de spinner weergegeven, en andersom. Wanneer CurrentAudit door een methode aangepast wordt en deze ineens wel of niet null wordt, wordt de if-else opnieuw gecheckt. Dit kan ervoor zorgen dat de tabel dan weer wel of niet zichtbaar wordt. Voorheen moesten voor het opnieuw doen van die validatie aparte functies geschreven worden. Inline Razor maakt het werk van de developer hiermee een stuk makkelijker.

Databinding speelt binnen Blazor ook een grote rol en zorgt wederom voor gemak van de developer.

Afbeelding met tekst, binnen, schermafbeelding, scherm

Automatisch gegenereerde beschrijving

In-line databinding en eventhandler 1

Zo kan er makkelijk aangegeven worden wat een value van, in dit geval, een spinbox is. Ook kan daarbij meteen aangegeven worden welke methode er aangeroepen moet worden in het geval van een aanpassing van die value. Ook in dit geval weer, wanneer count aangepast wordt, wordt dit ook in de UI aangepast.

### Components

Zoals genoemd is kan een Blazor component voor verschillende doeleinden worden gebruikt. Zo gebruik ik de components voornamelijk als pagina’s. Maar wanener er een bepaald stuk html code vaker voorkomt binnen de app, kan het makkelijk zijn deze te hergebruiken als component. Ik heb bijvoorbeeld een LoadSpinner die ik op elke pagina gebruik. Ik zou ervoor kunnen kiezen steeds dezelfde html code opnieuw te schrijven. Maar door deze in een component te zetten hoef ik steeds maar de component aan te roepen om de spinner weer te geven.



html voor LoadSpinner



Aanroepen LoadSpinner

Dit zorgt niet alleen voor meer gemak bij de developer, het zorgt ook voor een betere onder houdbaarheid. Ik zie, behalve dit simpele voorbeeld van een spinner, veel mogelijkheden voor de toekomst. Zo kunnen er bijvoorbeeld ook components worden gemaakt voor de vaak terugkerende tabellen (met de daarbij horende parameters en opties).

## Conclusie

Door dit onderzoek ben ik veel te weten gekomen over het maken van een applicatie met Blazor. Zo ben ik ervan overtuigd dat wij onze verouderde applicatie opnieuw kunnen bouwen door middel van Blazor en deze kunnen laten bouwen door enkel backend developers. Er is natuurlijk wel enige html kennis vereist, maar er hoeven geen lastige front-end frameworks te worden aangeleerd. Tijdens het hele proces van het maken van mijn prototype voelde het constant alsof ik aan een regulier backend aan het werk was. Wanneer iemand de juiste kennis heeft van .NET, heeft Blazor een erg kleine leercurve. Het heeft mij oprecht verbaasd hoe weinig code er nodig was om dit prototype te maken, helemaal wanneer je de bulk aan code van onze JQuery-based applicatie ernaast houdt.

### Voorlopig advies

- Er moet goed gekeken worden naar de binnenkomende data. Er moet goed gekeken worden wat er nog wel/niet wordt gebruikt. De huidige data zit vol vervuiling.

- Zoals aangegeven ben ik in de oude app hardcoded faulttypes tegengekomen. Er moet hiervoor, los van of we Blazor gaan gebruiken, een definitieve oplossing komen.

# 1.3 Kan de gemaakte app in cache draaien, zodat er offline gewerkt kan worden?

## Doel

De gebruikers van onze app hebben lang niet altijd internetverbinding, zo lopen ze vaak in kelders en parkeergarages, waar het hebben van internet lang niet altijd gegarandeerd is. Ook kan het voorkomen dat gebruikers geen tablet hebben met een data abonnement.

Toch moeten deze gebruikers onze app kunnen benaderen om zo toch de lokale data te kunnen gebruiken. Met dit onderzoek ga ik uitzoeken welke mogelijkheden hiervoor zijn.

## Strategie



Deze deelvraag wordt beantwoord met de hulp van Library(Literature study) waarna ik de vergaarde informatie ga toepassen door middel van een prototype(Workshop).

## Uitwerking

Om meer inzicht te krijgen in welke mogelijkheden er zijn, heb ik eerst zoekwerk gedaan, waarbij ik vele blogs en pagina’s af heb gezocht.

**PWA (service worker)**

Wanneer je naar offline Blazor app zoekt krijg je alleen maar het advies om er een PWA (progressieve Web Application) van te maken. Hiermee download je de applicatie, eigenlijk dus de hele web app, door middel van een knop op de browser. Hiermee wordt er op jouw device een icoontje op je home-scherm aangemaakt, waardoor je de app kan starten. Ook kan de app in dit geval gestart worden zonder dat er internetverbinding is, aangezien alle content, css, libraries en cache op de device zijn gedownload. De gebruiker heeft in principe alleen maar eenmalig een internetverbinding nodig om zijn applicatie te downloaden (en in ons geval om de metingen naar de lokale database te downloaden)



Huidige browser support PWA

Om een applicatie een PWA te maken zijn er een aantal requirements:

* De data wordt verstuurd over een beveiligde HTTPS verbinding.
* Er moet een service worker worden geïnstalleerd die het caching afhandelt.
* Er moet een app manifest worden ingevuld, bestaande uit enkele simpele properties.

Om een Blazor PWA applicatie te maken, dient er alleen een vinkje afgevinkt te worden bij het maken van een project. Helaas heb ik daar vooraf niet aan gedacht.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Aanmaken van PWA bij nieuw project

Toch is er een manier om een Blazor app simpel om te bouwen naar een PWA. Hiervoor dient de eerdergenoemde Service Worker toegevoegd te worden aan het bestaande project. De stappen om mijn niet-PWA om te zetten naar een PWA zijn:

* Het downloaden van vier files downloaden van Microsoft GITHub: icon-512.png, manifest.json, service-worker.js, service-worker.published.js
* Het toevoegen van deze files aan de index.html van mijn applicatie
* Het registreren van de serviceworkers in mijn project

Afbeelding met tekst, schermafbeelding, monitor, scherm

Automatisch gegenereerde beschrijving

Toevoegen van serviceworkers aan csproj

Met deze simpele stappen zorg ik ervoor dat ik binnen vijf minuten mijn regulieren WASM applicatie heb omgezet naar een Blazor PWA. Wanneer de app nu wordt geopend zie ik dat ik de app kan installeren.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Installeren PWA

Bij het testen is meteen te zien dat zelfs wanneer de browser offline is het hoofdscherm geladen wordt.

## Conclusie

Voorafgaand aan dit onderzoek was ik bang dat ik erg veel verschillende manieren voorgeschoteld kreeg. Gelukkig ben ik door de vele adviezen van lotgenoten in de richting van een PWA gedwuwd. Hiermee ben er ik meteen, zonder het uitproberen van andere opties, zeker van dat dit de manier is om de applicatie offline bruikbaar te maken. Hiermee blijven we zowel als website als offlineapplicatie beschikbaar voor al onze gebruikers, of ze nou een desktop, laptop, telefoon of tablet gebruiken.

# 1.4 Heeft Blazor een toekomst?

## Doel

Voordat we definitief kunnen zeggen of we Blazor gaan gebruiken voor het maken van onze apps, is het van groot belang om te weten wat de ‘houdbaarheid’ is. Natuurlijk is het in deze wereld moeilijk om een beeld te schetsen hoe lang iets meegaat, maar ik ga door middel van dit onderzoek toch een beeld proberen te schetsen.

## Strategie



Literature study (library) 1

Zoals aangegeven in mijn projectplan ga ik tijdens dit onderzoek gebruik maken van de “Literature study” onderzoeksmethode, welke valt onder het “Library” veld.

Initieel was de strategie het opzoeken van roadmaps gemaakt door de ontwikkelaars van Blazor. Hierbij wilde ik door middel van (officiële) posts in kaart brengen wat er nog aan zat te komen betreft toevoegingen en updates in de toekomst. Echter heb ik hier tijdens mijn onderzoek van af moeten wijken. Wel ben ik trouw gebleven aan de onderzoeksmethode.

## Uitwerking

Wanneer ik zoek naar een roadmap kom ik er al snel achter dat er geen officiële roadmap is voor releases van Blazor. Ook kan ik op de officiële platforms geen informatie vinden over eventuele releases in de toekomst of eventuele toekomstplannen van Microsoft zelf.

Wel zie ik dat het een hele populaire manier is voor het maken van front-end applicaties door C# developers. Veel van diezelfde developers stellen dezelfde vraag: “Heeft het zin om in te zetten op Blazor, heeft het een toekomst”. Ik moet dus op een andere manier gaan zoeken naar mijn eigen antwoord, aan de hand van informatie die mij die keuze kan doen maken.

Aangezien de release van Blazor pas in 2018 was, is het een relatief nieuwe methode voor het ontwikkelen van front-end applicaties. Behalve dit is het ook de enige manier voor het maken van front-end applicaties in C#. Dit bekent dat er geen splitsing is tussen verschillende kampen, waar er gekozen kan worden uit meerdere mogelijkheden voor het maken van C# front-end applicaties. Dit spreekt er in het voordeel van Blazor, omdat iedereen dit bovenstaande wilt bereiken, toegewezen wordt tot Blazor. Wat zorgt voor een community waarvan de ogen allemaal dezelfde kant op staan

Ook omdat dit zo nieuw is, wordt er nog veel over gesproken op fora. Er wordt ook door die gebruikers veel gespeculeerd over de toekomst van Blazor, maar niemand kan een duidelijk antwoord geven. Vaak wordt er gesproken dat Blazor, aangezien het gebruik maakt van WASM (WebAssembly), een grote kans heeft langer te overleven dan bijvoorbeeld oudere flops van Microsoft (bijvoorbeeld Silverlight). Omdat WASM een nieuwe standaard is voor web-development, waarbij het zeer goed ontvangen wordt, lijkt dit een bevestiging waar ik naar op zoek ben.

## Wat zeggen anderen?

Om tot een goed beeld te komen ga ik op zoek naar wat, naar mijn idee, goede bronnen er over schrijven om vervolgens tot een conclusie te komen.

**Matthew MacDonald, 10.000 volger op Medium.com en Microsoft MVP**

“…The technological underpinnings for Blazor are rock solid. It’s built on WebAssembly, a cross-platform standard that has significant vendor buy-in, universal browser support, and better performance than even today’s optimized JavaScript engines. WebAssembly also has some jaw-dropping proof-of-concept demonstrations, like 3D games and emulators running natively in the browser…”

“Blazor is a tough sell to current web developer, because it means leaving behind many of the libraries and technologies that have been up over a decade of modern JavaScript development. It’s not a seamless transition, and there’s no way to migrate a JavaScript application to a Blazor project. Switching to Blazor still means moving from the rough seas of JavaScript to a much smaller swimming pool. Most developers are going to check it out but won’t be ready to leave the ocean just yet.

For experienced C# developers, it’s a different story. If you have a team that knows .NET and they’re building an internal business application, Blazor lets them be more productive and build the front-end and back-end with one stack. If you were considering a Windows desktop application (built on something like WPF), Blazor is a more future-forward choice. But if you’re a business building modern, public web applications, there’s very little chance you’ll recommend a client start with Blazor today.

The bottom line? Blazor’s a niche — a powerful one — best suited to Microsoft developers who love .NET and C#. But don’t let the present blind you to the future. There’s a lot of promise in stacks that build on WebAssembly, and a lot of room for other languages to nibble away at the edges of JavaScript. And in the ever-changing world of web development, we’re overdue for another revolution.”

Bron: <https://medium.com/young-coder/is-blazor-the-future-or-just-another-walled-garden-441842cc249d>

**Paul Bradish, Digital marketing and software developer for over 15 years**

The verdict: Is Blazor is better than JavaScript?

While there’s little doubt to us that with each new release Blazor will continue to grow in popularity (and for good reason, too), we don’t see Blazor dethroning JavaScript in overall adoption any time soon. Even though Blazor has many clear and distinct advantages over JavaScript, JavaScript and its multiple, well-used frameworks that have stood the test of time, are here to stay for years to come. That said, the future appears to be bright for Blazor and we look forward to utilizing it on projects with goals that align well to what Blazor brings to the table.

Bron: <https://www.emergentsoftware.net/blog/is-blazor-better-than-javascript/>

**Emergents Software, award-winning software bedrijf met o.a. Philips als klant.**

Deciding whether to use Blazor WebAssembly instead of one of the many JavaScript frameworks readily available largely depends on your team’s experience and comfort level with learning a new language. If you or your team is more comfortable coding in C# vs JavaScript, Blazor is a solid option for you. Keep in mind, though, that Blazor’s developer community is relatively small and its knowledge base is still maturing. A team who already has a solid grasp of JavaScript and project constraints like timeline or budget may decide to avoid the hassle of learning a new framework and stick to one of the tried and true JavaScript Frameworks available.

Finally, developers who have the time and desire to learn new technology should feel confident giving Blazor a shot at this point. Our team at Emergent Software has numerous successful Blazor projects under our belts and will continue to leverage it on projects Blazor fits well with.

While there’s little doubt to us that with each new release Blazor will continue to grow in popularity (and for good reason, too), we don’t see Blazor dethroning JavaScript in overall adoption any time soon. Even though Blazor has many clear and distinct advantages over JavaScript, JavaScript and its multiple, well-used frameworks that have stood the test of time, are here to stay for years to come. That said, the future appears to be bright for Blazor and we look forward to utilizing it on projects with goals that align well to what Blazor brings to the table.

Bron: <https://www.emergentsoftware.net/blog/is-blazor-better-than-javascript/>

## Conclusie

Mijn conclusie is, dat het samen met vele andere frameworks, moeilijk te zeggen is wat de exacte levensduur van Blazor is.

Wat, zoals aangegeven, een groot voordeel is, is dat Blazor compileert naar de een nieuwe standaard WASM. Waarbij de verwachting is dat WASM alleen maar groter wordt en meer frameworks zich hier op gaan richten. Zoals ook door sommige ‘experts’ genoemd, is het als ervaren C# developer erg tijdbesparend om het wel te doen.

Wat ook in het voordeel van Blazor spreekt, is het feit dat het een erg populair onderwerp op Stacoverflow en Github is. Zo worden er erg veel actieve vragen gesteld en beantwoord. Daarbij staan er ook erg veel actieve projecten op Github. Waarbij veel gebruikers onderling ook bovenstaande vraagstelling bespreken. Vaak is het antwoord dat het pas in de kinderschoenen staat en dat het alleen maar groter zal worden.

Waarbij je als bedrijf zijnde zelf de afweging moet maken, aan de hand van het gevoel dat je erbij krijgt. Toch neig ik te zeggen dat we voorlopig niet bang hoeven te zijn dat Blazor ophoudt met het krijgen van support. Als je ziet hoe groot het aan het worden is en het feit dat er tot nu toe nog productief aan ontwikkel wordt, denk ik dat we er voorlopig nog niet van af zijn.

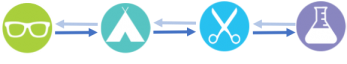
# 1.5 Welke tooling en methodes kunnen er gebruikt worden van het schrijven van (automatische) tests?

## Doel

Het doel van dit onderzoek is het vinden van een geschikte manier voor het testen van onze eventuele toekomstige Blazor applicaties. Hierbij wordt er gekeken naar welke methodes en strategieën op de markt zijn gebracht voor automatiseren van tests. Hierbij wordt dus niet gekeken naar bijvoorbeeld end-to-end tests, acceptatietests of integratietests.

## Strategie

Ik ga dit onderzoek doen volgens ondertstaande onderzoeksmethodes

*F* *igure 1: Choose fitting pattern*

Dit betekent dat ik volgens de volgende volgorde ga werken:

1. Library: Literature study & community research

2. Field: Explore (user)requirements

3. Lab: Usability testing

4. Workshop: prototyping

# Uitwerking

## Welke technieken zijn beschikbaar?

Bij het onderzoek naar testmethodieken voor Blazor, ben ik steeds uitgekomen op twee belangrijke methodes: Unit testing en E2E testing (end-to-end). Twee totaal verschillende methodes die elkaar aan kunnen vullen en waarmee je een Blazor app op bijna elk vlak kan testen.

### Unit Tests

Unit testing is de meeste bekende testmethode. Dit is een methode die, met enkele uitzonderingen, nagenoeg altijd automatisch wordt uitgevoerd. Toch is Unit testen in Blazor anders dan de gebruikelijk Unit Test. Over het algemeen wordt deze testmethodiek gebruik voor het testen van backends, toch kan deze ook gebruikt worden voor het testen van frontend, waar Blazor onder valt. Hiervoor wordt wel een speciale Unit Test library voor gebruikt. Deze libaries zorgen er voor dat de Blazor components geladen worden met de benodigde (Mock) data en de juiste state. Hierbij kunnen tests worden uitgevoerd op zowel de input, output, state, lifecycles, eventhandlers en meer. Dit is dan ook meteen het verschil met de conventionele unit tests, waar de conventionele unit tests alleen class-specifiek op een methode wordt uitgevoerd.

Er is eigenlijk maar een Unit Test library die genoemd wordt: bUnit.

### E2E

Deze testmethodetest de workflow van een applicatie van begin tot eind en kan zowel manueel als automatisch worden gedraaid. De bedoeling van deze methode is het nabootsen van echte user input, zodat daadwerkelijke voorkomende scenario’s getest kunnen worden. Hierbij worden, in tegenstelling tot de Unit Tests, meerdere components getest.

Bij E2E tests zijn er meer mogelijkheden, zo worden bijvoorbeeld Cypress en Playwright for .NET erg vaak genoemd.

Microsoft heeft met onderstaande tabel in kaart proberen te brengen wat je voor kan stellen bij de specifieke tests.

| **Capability** | **Unit testing** | **E2E testing** |
| --- | --- | --- |
| Test scope | Razor component (Razor/C#) only | Razor component (Razor/C#) with CSS/JS |
| Test execution time | Milliseconds | Seconds |
| Access to the component instance | Yes | No |
| Sensitive to the environment | No | Yes |
| Reliability | More reliable | Less reliable |

Het is niet de bedoeling om zomaar alles te testen met elke methode die er is. Zo is het verstandig om bij bepaalde scenario’s een keuze voor een bepaalde testaanpak te kiezen. Ook hier heeft Microsoft een interessante keuzetabel voor gemaakt.

| **Scenario** | **Suggested approach** | **Remarks** |
| --- | --- | --- |
| Component without JS interop logic | Unit testing | When there's no dependency on JS interop in a Razor component, the component can be tested without access to JS or the DOM API. In this scenario, there are no disadvantages to choosing unit testing. |
| Component with simple JS interop logic | Unit testing | It's common for components to query the DOM or trigger animations through JS interop. Unit testing is usually preferred in this scenario, since it's straightforward to mock the JS interaction through the [IJSRuntime](https://learn.microsoft.com/en-us/dotnet/api/microsoft.jsinterop.ijsruntime) interface. |
| Component that depends on complex JS code | Unit testing and separate JS testing | If a component uses JS interop to call a large or complex JS library but the interaction between the Razor component and JS library is simple, then the best approach is likely to treat the component and JS library or code as two separate parts and test each individually. Test the Razor component with a unit testing library, and test the JS with a JS testing library. |
| Component with logic that depends on JS manipulation of the browser DOM | E2E testing | When a component's functionality is dependent on JS and its manipulation of the DOM, verify both the JS and Blazor code together in an E2E test. This is the approach that the Blazor framework developers have taken with Blazor's browser rendering logic, which has tightly-coupled C# and JS code. The C# and JS code must work together to correctly render Razor components in a browser. |
| Component that depends on 3rd party class library with hard-to-mock dependencies | E2E testing | When a component's functionality is dependent on a 3rd party class library that has hard-to-mock dependencies, such as JS interop, E2E testing might be the only option to test the component. |

## Wat is de criteria van deze techniek?

De gekozen methodiek moet in elk geval zorgen voor een applicatie, waarvan de kwaliteit gegarandeerd is. Hiervoor wil ik gebruik gaan maken van zowel Unit tests als automatisch E2E tests. De keuze voor automatische E2E tests is voornamelijk de tijdsbesparing die gemoeid gaat met het laten uitvoeren van een constante test. Op dit moment doen wij veel E2E tests zelf handmatig, soms worden er dan bepaalde dingen over het hoofd gezien, iets wat moeilijker gaat bij een geautomatiseerde test. Het opzetten van de tests duurt natuurlijk langer, maar die tijd wordt snel ingehaald. Automatische E2E tests duren enkele seconden, waar handmatige E2E tests soms vele minuten kunnen duren.

## Uitwerking gekozen methodieken

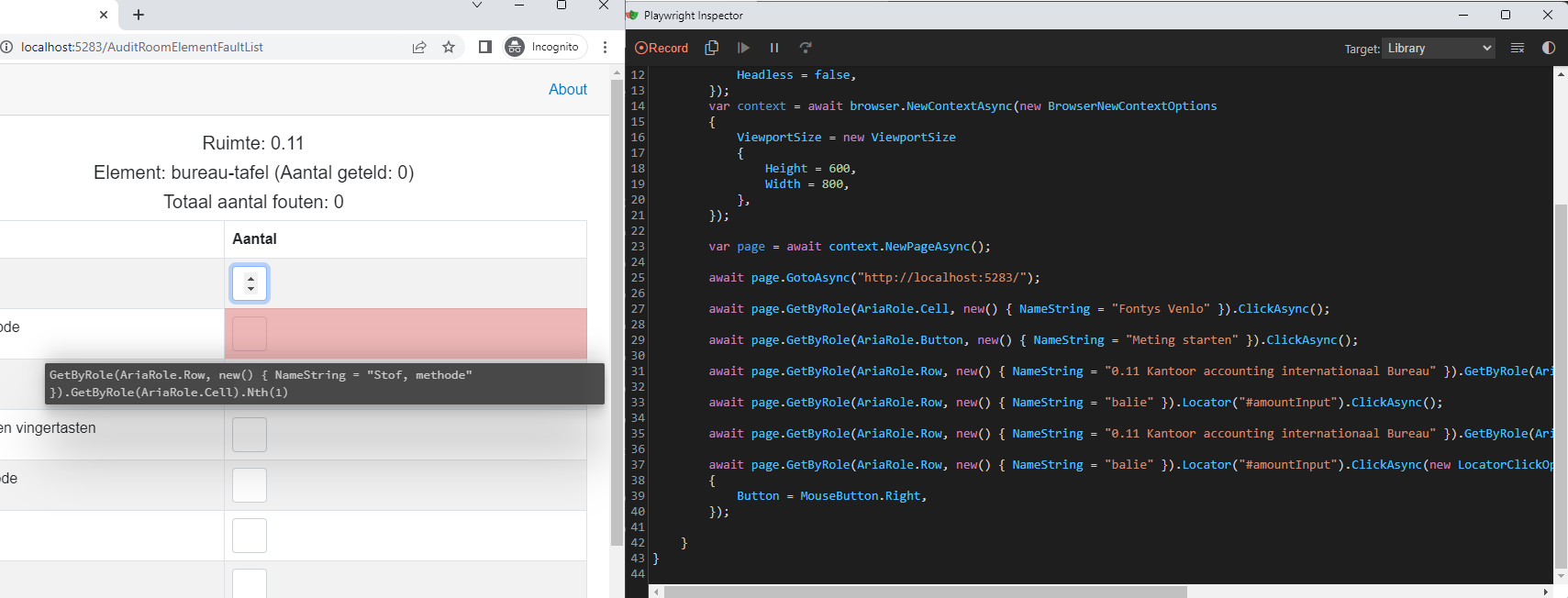
### bUnit

De keuze voor het uitvoeren van de Unit tests is gevallen op bUnit. Wanneer je zoekt op Unit Tests voor Blazor leidt elke pagina naar bUnit. Ook is bUnit gesponsord door de .NET foundation (een door Microsoft opgerichte non-profit organisatie) en verwijst Microsoft zelf in de officiële Blazor guide ook naar bUnit. Dit samen met het feit dat Microsoft ook nog eens de bedenker is van Blazor zelf, lijkt mij dit een goed onderbouwde keuze.

Bij bUnit kan er gekozen worden uit het schrijven van tests in de .cs of de .blazor file. Bij het schrijven van tests in de .blazor file is het makkelijker om HTML markup te gebruiken en is hier om ook de meest gekozen methode.

### Playwright for .NET

Voor het uitvoeren van de E2E tests is de keuze gevallen op Playwright. Playwright is een door Microsoft ontwikkelde cross-browser automatische E2E testing library. Voor Playwright bestaan handige tools, zoals de Codegen, waarbij een handmatige test door middel van een Record knop omgezet kan worden in een uitvoerbare test in het test framework naar keuze. Daarbij voorzien ze in een erg goed gedocumenteerde website, waarbij erg veel tutorials en documentatie te vinden is. Het enige nadeel zou zijn, dat in tegenstelling tot de Unittests, de tests per stuk enkele seconden duren. De browser wordt namelijk virtueel opgestart, waarbij de tests door de emulator worden uitgevoerd.



Playwright Codegen tool 1

## Conclusie

### bUnit

Ik heb het idee dat bUnit, wanneer er ook gebruik gemaakt wordt van een automatische E2E test, weinig toevoeging heeft. Het enige voordeel wat ik kan bedenken is dat de te gebruiken data makkelijk te managen en te mocken is, waar er bij Playwright gebruik wordt gemaakt van een draaiende instantie van de app. Het grootste nadeel is, wat Microsoft zelf ook zegt, ***“When a component's functionality is dependent on a 3rd party class library that has hard-to-mock dependencies, such as JS interop, E2E testing might be the only option to test the component.”.*** En aangezien mijn applicatie veel gebruik maak van JSInterop, zal het lastig worden bUnit volledig te kunnen benutten.

### Playwright for .NET

Binnen Factos maken wij bij front-end alleen gebruik van handmatige front-end tests. Hierbij is vooral Playwright een hele goede en tijdbesparende aanvulling. Wel moet ik er bij zeggen dat de leercurve erg hoog is, en het veel tijd kost om de vele mogleijkheden door te krijgen en volledig te kunnen benutten. Ik denk dat het een onderzoek op zich zal zijn om alle mogelijkheden uit te kunnen proberen ten te snappen. Het nadeel van Playwroght is dat de data lastiger te mocken is ddan met bUnit, waar er bij deze methode een draaiende instantie van de app moet draaien, waar Playwright door middel van een emulator zijn tests op gaat uitvoeren. Wel merk ik dat wanneer het eenmaal staat het een erg mooi testmiddel is.

Een andere conclusie is dat ik gemerkt heb, dat wanneer je deze methode gebruikt, het een groot voordeel is dat je werkt volgens een test-driven ontwikkelmethode. Wanneer de app eenmaal staat, en je de tests nog moet inbouwen, merk je dat je veel zaken (zoals id’s, extra classes etc), nog moet toevoegen succesvol te kunnen testen. Ook merk je dat sommige functionaliteiten beter zouden werken als je test-driven ontwikkeld. Deze conslusie is zeker iets om over na te denken bij eventueel maken van Blazor apps in de toekomst.

# Algemene conclusie

Wanneer alle conclusies per deelonderzoek in beraad worden genomen kan ik eigenlijk maar tot een conclusie komen: als bedrijf worden wij zeker beter wanneer wij onze huidige en nieuwe applicaties in Blazor gaan maken. Tijdens het maken van de prototypes en het doen van onderzoek ben ik persoonlijk al tot deze conclusie gekomen en de onderzoeksresultaten benadrukken deze persoonlijke conclusie alleen des temeer.

Om te beginnen is Blazor erg makkelijk te leren voor developers die een .NET achtergrond hebben. Hiervoor hoeven wij als bedrijf ook niet elke keer te schakelen tussen Javascript en C# wanneer wij wisselen tussen werk aan back-end en front-end. Ik hoopte al dat dit het geval was, maar tijdens het onderzoek is het mij nog positiever bevallen dan vooraf gedacht. Daarbij denk ik het als bedrijf veilig is om de kennis tot een bepaalde hoek te richten, zodat je erg sterk kan worden op een bepaald gebied. Waar we nu, zoals al aangegeven, vaak geschakeld moest worden tussen C# en Javascript.

Daarbij is Blazor erg krachtig; er is erg weinig code nodig voor het bereiken wat de klant wil. Het gemak waarmee ervan af scratch een volledige applicatie kan worden opgezet heeft mijn ogen echt geopend.

Ook de community is erg actief en enthousiast, er is veel documentatie te vinden en vragen worden door de community altijd nog erg snel beantwoord. Dit heeft ook bijgedragen aan de conclusie of wij met Blazor in zee moeten gaan.

Het enige waar ik geen duidelijk antwoord op heb kunnen vinden is het toekomstperspectief. Zoals ik in deelvraag 1.3 al aangaf: *“Mijn conclusie is, dat het samen met vele andere frameworks, moeilijk te zeggen is wat de exacte levensduur van Blazor is… Waarbij je als bedrijf zijnde zelf de afweging moet maken, aan de hand van het gevoel dat je erbij krijgt.”.*Natuurlijk is er in de ICT nooit een echte levensverwachting van een bepaalde methode of framework. Toch denk ik, dat wij als bedrijf alleen al, door de eerder genoemde voordelen Blazor moeten gaan gebruiken om als bedrijf onze apps naar het hoogste niveau te tillen.

Kortom: “Zouden wij als bedrijf onze nieuwe front-end applicaties moeten maken met Blazor?”. “Ja!”

# Bijlages

* Sourcecode en documenten: https://github.com/quatro32/VSRKMS
* Analyse.docs: context- en requirementsanalyse, MoSCoW en eiden en wensen layout.
* Data storage tests.docx: E2E tests voor onderbouwing van keuze data storage.
* Projectplan semester 7.docx: Projectplan
* Klassendiagram.png: klassendiagram van prototype
* SCHETS ELEMENTLIJST.png
* SCHETS FOUTENLIJST.png
* SCHETS HOOFDSCHERM.png
* SCHETS RUIMTELIJST.png
* SCHETS STARTSCHERM.png

# Bronnen

Guardrex, G. (2022, November 8). *ASP.NET Core Blazor state management*. Microsoft Learn. https://learn.microsoft.com/en-us/aspnet/core/blazor/state-management?view=aspnetcore-6.0

Huber, T. C., Huber, T. C., & Huber, T. C. (2021, April 19). *Store Data of Your Blazor App in the Local Storage and in the Session Storage*. Thomas Claudius Huber. https://www.thomasclaudiushuber.com/2021/04/19/store-data-of-your-blazor-app-in-the-local-storage-and-in-the-session-storage/

*IndexedDB in Blazor*. (2019, August 3). Steve Sanderson’s Blog. https://blog.stevensanderson.com/2019/08/03/blazor-indexeddb/

Jinjinov, J. (n.d.). *GitHub - Jinjinov/IndexedDB.Blazor: A Blazor library for accessing IndexedDB*. GitHub. https://github.com/Jinjinov/IndexedDB.Blazor

*NuGet Gallery | Packages matching blazor indexeddb*. (n.d.). https://www.nuget.org/packages?q=blazor+indexeddb

*TG.Blazor.IndexedDB 1.5.0-preview*. (n.d.). https://www.nuget.org/packages/TG.Blazor.IndexedDB/1.5.0-preview

*Window.localStorage - Web APIs | MDN*. (2022, September 21). https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

Bose, S. (2022, October 31). *What is End To End Testing?* BrowserStack. https://www.browserstack.com/guide/end-to-end-testing

dotnet. (2021, August 23). *The .NET Docs Show - Blazor component testing with bUnit*. YouTube. https://www.youtube.com/watch?v=JtyVBbcVplY

dotnet. (2022, November 12). *Testing Blazor Applications with Playwright | .NET Conf 2022*. YouTube. https://www.youtube.com/watch?v=gBky9\_AskNQ

(2022, November 8). *Test Razor components in ASP.NET Core Blazor*. Microsoft Learn. https://learn.microsoft.com/en-us/aspnet/core/blazor/test?view=aspnetcore-7.0

Maze, C. (2022, April 26). *Introduction to Testing Blazor WebAssembly With bUnit*. Code Maze. https://code-maze.com/test-blazor-webassembly-bunit/

Solau, X. (2022, November 14). *Enable Playwright so that you can easily use it (including on your build agents) [Part 2 ]*. Medium. https://medium.com/younited-tech-blog/end-to-end-test-a-blazor-app-with-playwright-part-2-3980b573e92a

TVUG. (2021, November 18). *Getting started with testing in Blazor*. YouTube. https://www.youtube.com/watch?v=TGMe3wE\_6BY

*Writing tests for Blazor components | bUnit*. (n.d.). https://bunit.dev/docs/getting-started/writing-tests.html?tabs=xunit

Bradish, P. (2022, 25 mei). *Is Blazor Better Than JavaScript? - Emergent Software Blog*. https://www.emergentsoftware.net/blog/is-blazor-better-than-javascript/

Galloway, J. (2020, 4 november). *Building a Progressive Web App with Blazor*. Visual Studio Blog. https://devblogs.microsoft.com/visualstudio/building-a-progressive-web-app-with-blazor/

Ma, C. (2021, 28 april). *Is Blazor the future in Web App Development?* iFour Technolab Pvt. Ltd. https://www.ifourtechnolab.com/blog/is-blazor-the-future-in-web-app-development

MacDonald, M. (2022a, januari 5). *Is Blazor Just Another Walled Gaden? | Why Microsoft’s most ambitious technology could power the future of the web | Young Coder | Matthew MacDonald | Young Coder*. Medium. https://medium.com/young-coder/is-blazor-the-future-or-just-another-walled-garden-441842cc249d

*Microsoft’s Long-Term Support (LTS) Plans for Blazor - Microsoft Q&A*. (2022, 24 januari). https://learn.microsoft.com/en-us/answers/questions/708050/microsoft39s-long-term-support-lts-plans-for-blazo.html

Campana, N. (2022, October 21). *What Does A Blazor Developer Do?* Freelancer Blog. https://www.freelancermap.com/blog/what-does-blazor-developer-do/

Danroth27. (2022, April 13). *An introduction to Blazor for ASP.NET Web Forms developers*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/architecture/blazor-for-web-forms-developers/introduction

(2022, November 8). *ASP.NET Core Blazor*. Microsoft Learn. https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-7.0

kudvenkat. (2020, February 24). *What is Blazor*. YouTube. https://www.youtube.com/watch?v=uuzi3SmCLVo

Kuijk, A. van. (2021, December 6). *Blazor, wat is het en wat kan je er mee?* Delta-N. https://www.delta-n.nl/blazor-wat-is-het-en-wat-kan-je-er-mee/

(2022, December 14). ASP.NET Core Blazor data binding. Microsoft Learn. <https://learn.microsoft.com/en-us/aspnet/core/blazor/components/data-binding?view=aspnetcore-7.0>

Galloway, J. (2020, 4 november). Building a Progressive Web App with Blazor. Visual Studio Blog. https://devblogs.microsoft.com/visualstudio/building-a-progressive-web-app-with-blazor/

How do I create a Blazor WebAssembly PWA to work offline? | Blazor FAQ. (z.d.). Syncfusion. https://www.syncfusion.com/faq/blazor/webassembly/how-do-i-create-a-blazor-webassembly-pwa-to-work-offline

Cache Storage in Blazor WebAssembly .NET 6 - Blazor School. (z.d.). https://blazorschool.com/tutorial/blazor-wasm/dotnet6/cache-storage-658620