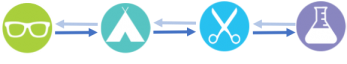
# Welke methodes van opslaan voor data kent Blazor?

## 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

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 internet verbinding 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 hier voor 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.

Er zijn twee standaard mogelijkheden volgens Microsoft.

### 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 er voor 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 er voor 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 enthausiast 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 er achter dat dit project “deprecated” was, iets wat ik had kunnen voorkomen door vooraf beter onderzoek te doen.

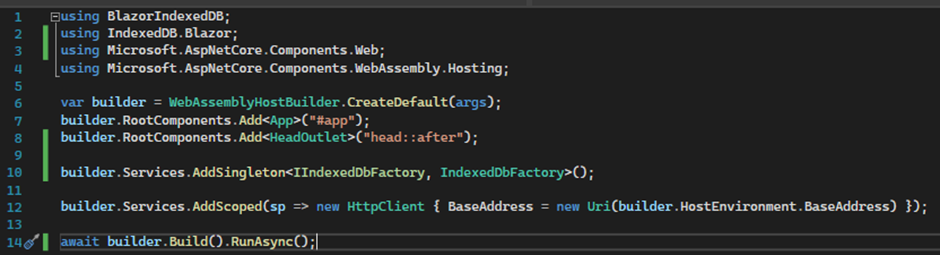
Afgezien van het feit dat deze package niet meer werkte, was het geen verloren tijd. Door deze les heb ik geleerd nauwkeuriger onderzoek te doen, vooral bij het kiezen van oplossing binnen de software. Dit heeft dan ook bijgedragen aan mijn “judgement” leeruitkomst.

### 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 een erg goed onderhouden project is, waar veel activiteit plaatsvind en heeft gevonden. Ook is de code open-source en erg goed te snappen. 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.

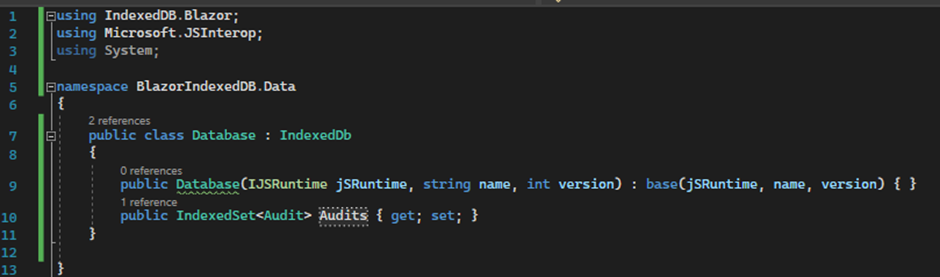
Ook is de implementatie van deze package erg simpel, zoals te zien in onderstaande screenprints.

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



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.



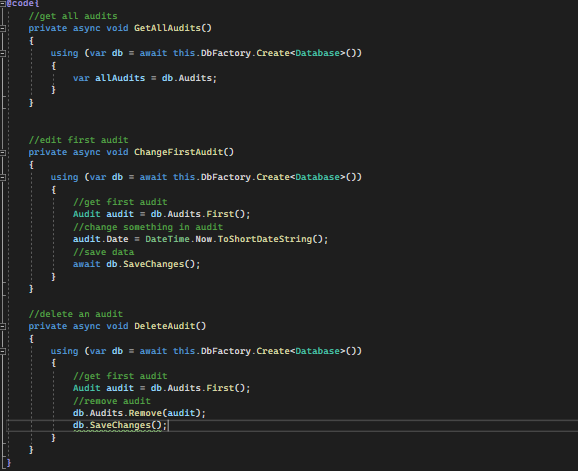
Definieeren database object



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.



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 er van bewust dat ik in de eerste instantie te snel met een methode in zee ben gegaan. Dit wilt 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.

# 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