# 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 er bij komt kijken. Dit betreft zowel het procesmatige als het technische aspect.

## Strategie

Om er achter 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

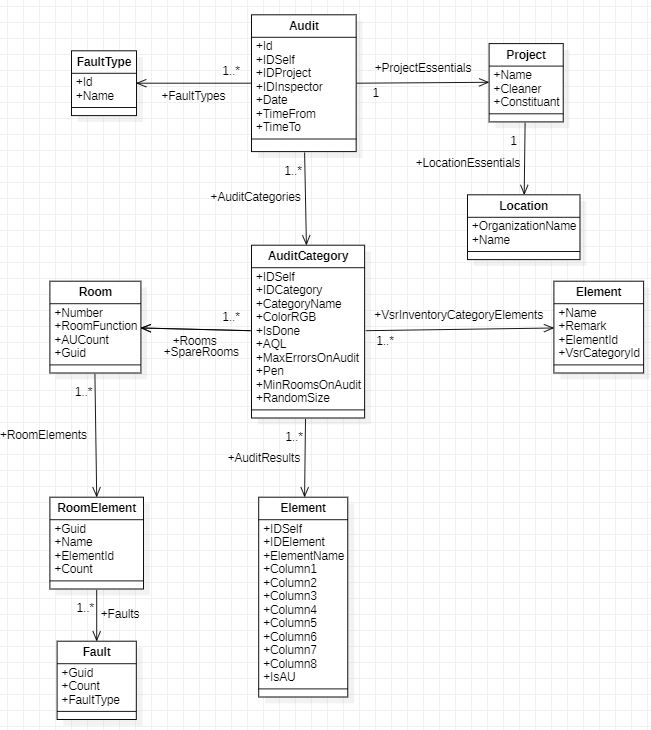
Automatisch gegenereerde beschrijving

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



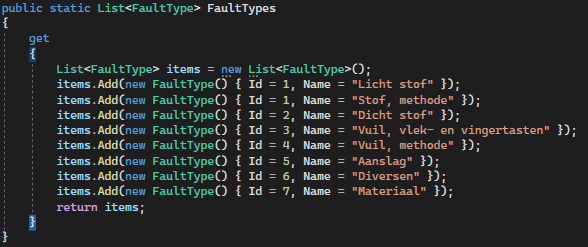
Screenshot datamodel 3

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 zelf geschreven 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 betekend 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.

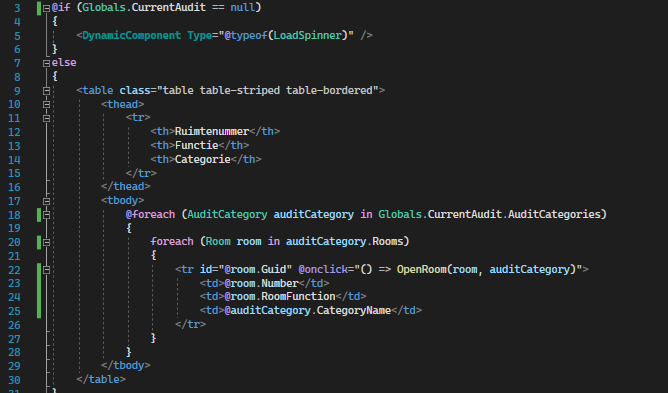
F

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 er voor 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

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.

**Hergebruik html door middel van 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 er voor kunnen kiezen steeds dezelfde html code opnieuw te schrijven. Maar door deze in een component te zetten hoef ik steeds maar het 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 er van 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 er naast 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.

## Bronnen:

<https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-7.0>

<https://www.delta-n.nl/blazor-wat-is-het-en-wat-kan-je-er-mee/>

<https://learn.microsoft.com/en-us/aspnet/core/blazor/components/data-binding?view=aspnetcore-7.0>