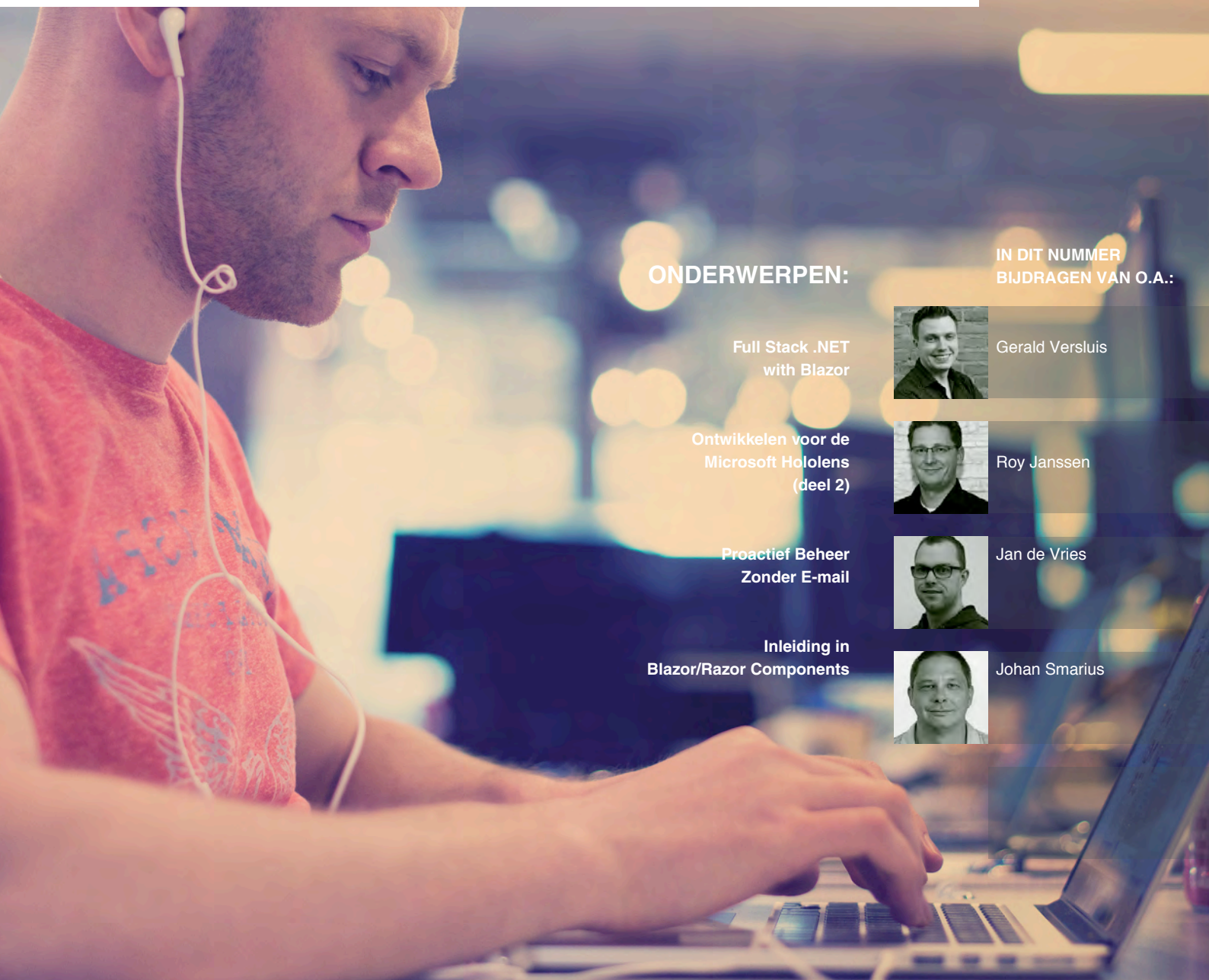
ISSN: 2211-6486

SDN MAGAZINE KNOW-LEDGE #137

Nummer 137 Juni 2019 SDN Magazine verschijnt elk kwartaal en is een uitgave van Software Development Network

SDN Magazine is hét lijfblad van en voor programmeurs, met artikelen over ontwikkelen voor Windows, Web, Mobile of Cloud. Of u nu ontwikkelt voor .NET, Delphi of iets anders, als lid van het SDN weet u alles.



ONDERWERPEN:

IN DIT NUMMER BIJDRAGEN VAN O.A.:

Gerald Versluis

Roy Janssen

Jan de Vries

Johan Smarius

KUNNEN WE JOU DE BEVEILIGING VAN MILJOENEN KLANTGEGEVENS TOEVERTROUWEN?

Achmea heeft de ambitie om de meest vertrouwde verzekeraar te zijn. Daarbij speelt IT een cruciale rol. We zijn dan ook voortdurend op zoek naar ambitieuze IT'ers. Bijvoorbeeld IT'ers die gaan meewerken aan zaken zoals security monitoring, Intrusion Prevention/Detection, security testing, en auditing. Professionals die grote uitdagingen aangaan in een complexe infrastructuur en met de opkomst van cybercrime. Zo bouwen we verder aan ons bedrijf als de grootste digitale verzekeraar en het vertrouwen van onze klanten. We bieden je een groot aantal opleidingen en doorgroeimogelijkheden. Meer weten? Kijk op werkenbijachmea.nl

WIJ STAAN VOOR GROTE IT-UITDAGINGEN

KUNNEN WE DIE JOU TOEVERTROUWEN?

AGIS
AVÉRO ACHMEA
CENTRAAL BEHEER ACHMEA
FBTO
INTERPOLIS
ZILVEREN KRUIS ACHMEA

achmea

IT VACATURES

- Senior Security Tester in Apeldoorn
- Security Specialist in Apeldoorn
- Technisch Tester in Amsterdam

Bekijk alle vacatures op werkenbijachmea.nl

voorwoord

Beste SDN Magazine lezer,

Het jaar 2019 is alweer halverwege en voor je ligt/staat het SDN magazine. Net op tijd voor de vakantie en met super mooie artikelen.

De periode van grote events ligt alweer even achter ons. Tijdens het bekende Microsoft Build conferentie werden weer allerlei nieuwtjes gebracht, tijdens onze SDN Cast hebben we daar uitgebreid op terug gekeken. Door razende reporter Roy hebben we ook een mooie Behind the Scene met een interview van Scott Hanselman. Kijk op ons Youtube kanaal: youtube.com/sdncast. Ook Apple en Google hebben hun developer events gehad allebei met Dark Mode voor hun devices. Ook deze hebben we tijdens de SDN Cast besproken. Tune in op elke donderdag om bij gesproken te worden over de laatste nieuwtjes en geplande events!

In dit magazine artikelen over Full Stack .NET with Blazor, Proactief Beheer Zonder E-mail, Ontwikkelen voor de Microsoft HoloLens (deel 2), Inleiding in Blazor/Razor Components, Truman Show Agile en XAML markup extensions geschreven door onze collega's. De auteurs Fons Sonnemans, Gerald Versluis, Jan de Vries, Roy Jansen, Johan Smarius en Sander Hogendoorn geven allemaal een inkijkje in hun ervaringen. Bedankt schrijvers, ik heb er weer veel van geleerd.

Veel leesplezier en tot ons volgende event of uitzending van de SDN Cast. Wil je schrijven of spreken of meedoen aan onze cast meld je dan op redactie@sdn.nl.

Namens de SDN medewerkers alvast fijne vakantie!

Groeten,

Marcel Meijer
Eindredacteur Magazine en Voorzitter SDN

SDN MAGAZINE KNOW-LEDGE#137

Over Marcel Meijer:
Marcel Meijer is Freelance (Enterprise) Architect, Teamlead, CTO, CTO Advisor of interim manager. Daarnaast is Marcel voorzitter, bestuurslid, event organisator en eindredacteur bij de SDN. Sinds 2010 mag hij zich MVP noemen.

Colofon

Uitgave
Software Development Network
Zevenentwintigste jaargang
No. 137 • juni 2019

Bestuur van SDN
Marcel Meijer, voorzitter
Rob Suurland, penningmeester
Remi Caron, secretaris

Redactie
Marcel Meijer (redactie@sdn.nl)

Aan dit magazine werd meegewerkt door
Roel Hans Bethlehem, Bob Swart, Maarten van Stam, Arjen Bos, Fanie Reynders, Rob Suurland, Remi Caron, Marcel Meijer en natuurlijk alle auteurs!

Listings
Zie de website www.sdn.nl voor eventuele source files uit deze uitgave.

Contact
Software Development Network
Postbus 506 7100 AM Winterswijk
Tel. (085) 21 01 310
info@sdn.nl

Vormgeving en opmaak
Reclamebureau Bij Dageraad
Winterswijk www.bijdageraad.nl

Adverteerders

Adverteren
Informatie over adverteren en de advertentietarieven kunt u vinden op www.sdn.nl onder de rubriek Magazine.

SDN MAGAZINE KNOW-LEDGE #137

AGENDA 2019/2020

1-2 oktober
Techorama Ede

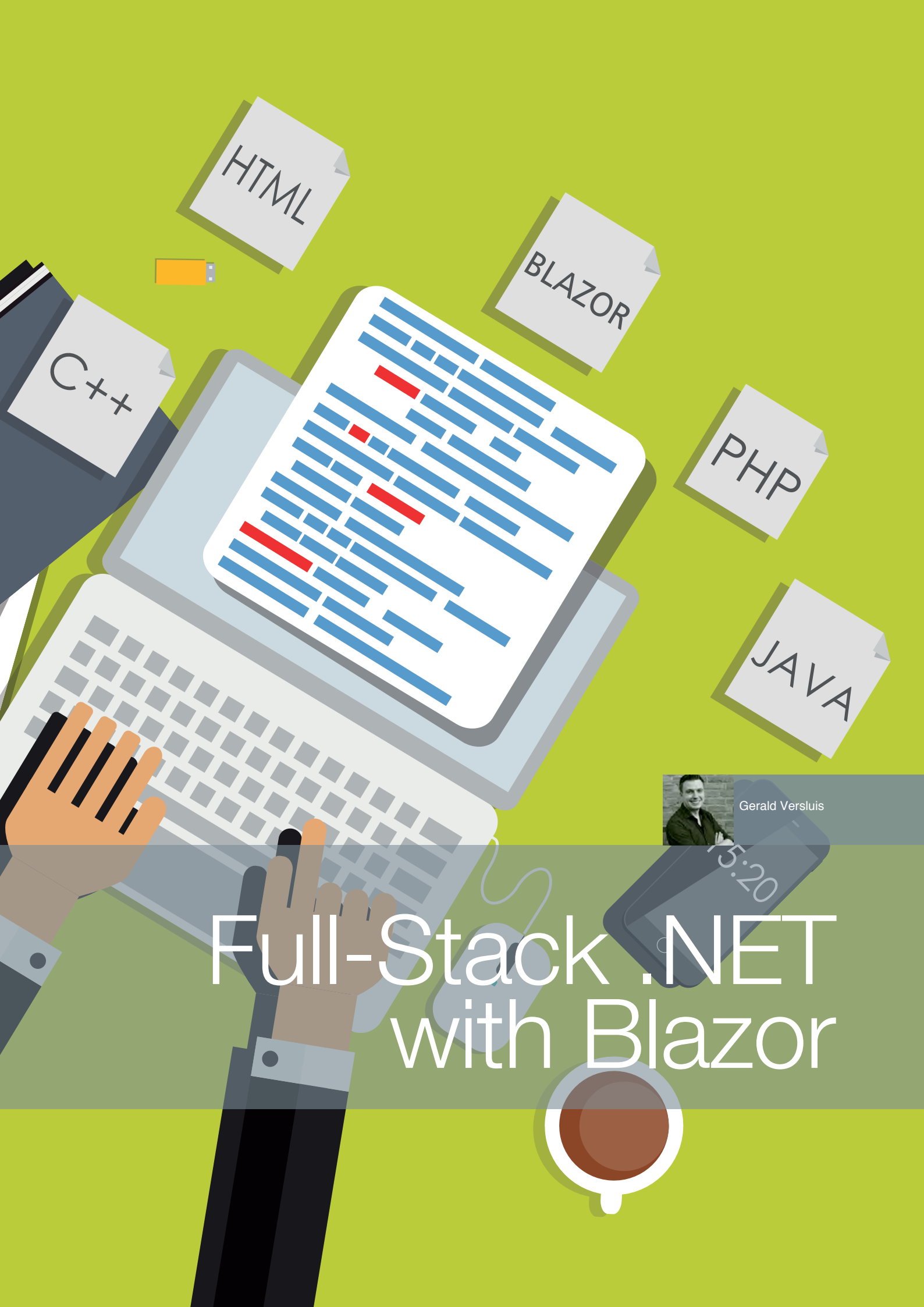
4-8 november
Ignite
Orlando Florida

13 december
SDN event 3
Zeist

25 maart 2020
FutureTech
Utrecht

Medio mei 2020
Build

Inhoud



Full-Stack .NET with Blazor

Imagine a world where you can build web applications without Angular, NPM, React, jQuery, Bootstrap and name all the JavaScript and web frameworks that are released at the speed of light. Imagine, that you can write a full blown, full-stack web application all right from C# and .NET without leaving your safe Visual Studio and Microsoft ecosystem. Sounds awesome, right?!

Let me introduce you to Blazor. You have probably already heard of it. Many people are asking themselves if this new Microsoft technology will (finally) kill JavaScript and clear the path for typed, object-oriented web development from front to back. In this article we will dive into WebAssembly, the technology behind Blazor and of course Blazor itself. After reading this, you will be able to get started with your very first Blazor web application. And you'll love it!

Before we start looking into Blazor itself, we have to take a step back and explore WebAssembly. WASM, the short form for WebAssembly, is a new type of intermediate code that can be run in modern web browsers. WASM provides a compact binary format that can perform with near-native performance. It isn't tied to one specific language, it is rather a compilation target. This means that whenever a language implements support for WebAssembly, you can write in the language you want, but it can be compiled to WebAssembly and run as you would expect. In our case, that means you can simply write C#, that C# code is then compiled to WebAssembly and will execute in a browser without any problems. WASM is designed to work alongside JavaScript and makes use of the same sandbox environment as JavaScript, so the code should be contained within your browser.

As mentioned, the performance is near-native, but there is an initial overhead of downloading the WASM modules. In the case of .NET that would be the .NET runtime and all referenced DLL files necessary to run your app. Of course, not the full, few hundred megs of .NET runtime are downloaded, but just a very slimmed down, bare necessities version. At the time of writing it is a couple of megabytes, but in the near future this will be trimmed down to several kilobytes.

Ultimately, the WebAssembly technology is not exclusively tied to browsers. Whenever it becomes more and more mature it could be used in mobile apps, desktop apps and possibly other platforms.

https://www.infoworld.com/article/3291780/web-development/what-is-webassembly-the-next-generation-web-platform-explained.html

Before we dive into Blazor, a couple of warnings. At the time of writing Blazor is still advertised as being an experimental technology. You should not expect any support and you should not use it in production environments. Releases are following up one after another pretty fast, so this can change quickly, but something to keep in mind.

Also, because of the technique being experimental, there is only support for Visual Studio for Windows right now. For this article, the latest Blazor release is 0.8.0 and Visual Studio 2019 preview 3 will be used. By the time you are reading this, Visual Studio 2019 is probably released as it is scheduled to launch in April 2019.

GETTING STARTED

Make sure you have all the prerequisites installed which is the .NET Core 3 SDK and Visual Studio 2019 preview 2 with the ASP.NET workload selected. Additionally, you will need the Blazor extension for Visual Studio. The latter you can find here: https://marketplace.visualstudio.com/items?itemName=aspnet.blazor. This link will also point you in the right direction for the prerequisites I mentioned earlier. The installation should be pretty straight-forward. The extension is needed right now to install the right project templates. In the future these might be included by default, but for now you need to install them separately.

After everything is installed correctly, we can simply go ahead and create a new ASP.NET Core Web Application. Start VS2019 and choose to create a new project. If you haven't worked with VS2019 before, the welcome screen might look a bit different then you are used to. In the "Create a new project" screen, choose ASP.NET Core Web Application, in the next screen name your project and in the last screen you will notice a couple of Blazor related templates. You can see the screens in the image underneath.

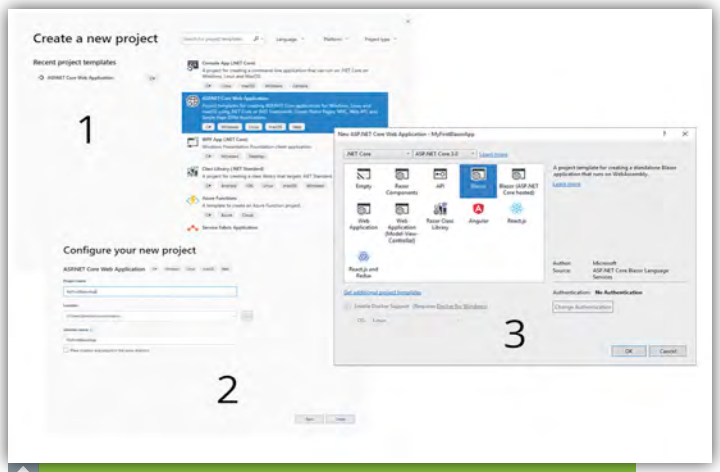


FIGURE 1: CREATING A NEW BLAZOR PROJECT

Basically, there are two variants: Blazor and Blazor (ASP.NET Core hosted). The first one can be hosted in IIS and needs an actual webserver to run, the latter is self-contained and can run from its own ASP.NET Core hosting. These variants are also known as server-side hosted and client-side hosted respectively.

SERVER-SIDE HOSTED VS. CLIENT-SIDE HOSTED

This immediately raises the question why should I choose one or the other? To be able to make a decision, we need to understand

which does what. When using the server-side hosted variant, your Blazor app is executed on the server in an ASP.NET Core host. All UI updates, event handling and JavaScript calls will happen through a SignalR connection. You can see this in a schematic overview in the image underneath.

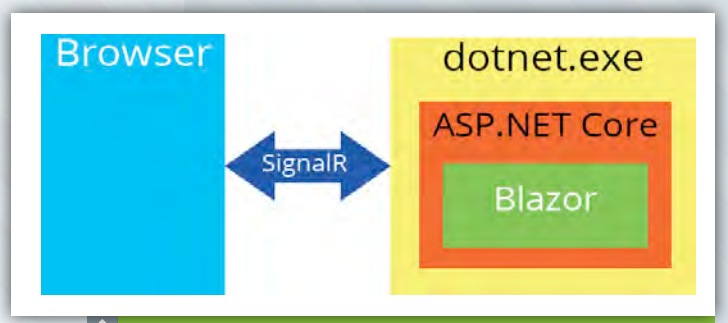


FIGURE 2: SERVER-HOSTED BLAZOR APPLICATION SCHEMATIC

Server-side Blazor applications are also referred to sometimes as Razor Components.

For the client-side hosted method, the whole Blazor app, including the .NET runtime and all dependencies are transferred to the client, the visitor of your web app, and is run there. You Blazor application runs on the browser's UI thread and all updates and event handling happens within that same process. The client-hosted way is how Blazor was built intentionally, just like JavaScript, the engine is on the client and all of the applications code will also run on that same client. A schematic overview of that can be seen in the image underneath.

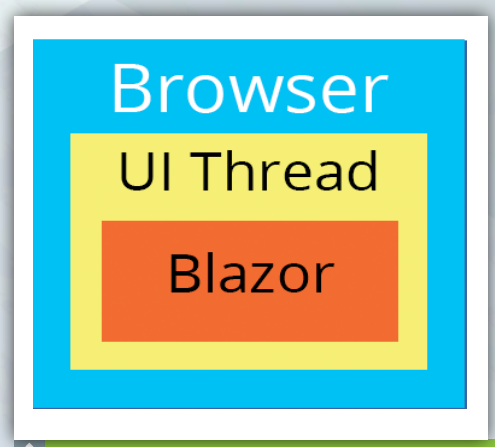


FIGURE 3: CLIENT-HOSTED BLAZOR APPLICATION SCHEMATIC OVERVIEW

You can read about it in more detail on the Microsoft Docs page for this: https://docs.microsoft.com /aspnet/core/razor-components/hosting-models?view=aspnetcore-3.0. On this page, the (dis) advantages of each method are described as shown below.

Another thing that you might be wondering about is: how is a server-side Blazor app different from a normal ASP.NET MVC app? The biggest difference is that Blazor has the ability to separate the execution of rendering the UI and executing the actual application logic. As an example, imagine a Blazor application that runs in a Web Worker - ability in browsers to run code on the background - thread with the UI thread handling all incoming and outgoing events. This makes Blazor suitable for creating Progressive Web Apps (PWA) or even desktop apps with Electron.

Server-hosted Blazor	
Pros	**Cons**
Write your app in .NET and C# using the component model	Higher latency, each interaction is a network call
Rich interactive feeling without unnecessary page refreshes	No offline support, app stops working when no connection is available
Significantly smaller app size and thus loads much faster because not all dependencies need to be downloaded to the client	Reduced scalability, server handles multiple client connections and state
Component logic can take full advantage of server capabilities, including .NET Core APIs	Requires an ASP.NET Core server. Cannot deploy from a static server.
Runs on .NET Core, which enables you to use existing tooling like debugging	
Works with clients that do not support WebAssembly or are otherwise constrained	
Client-hosted Blazor	
Pros	**Cons**
No .NET server-side dependency	App is limited to the capabilities of the browser
Rich interactive UI	Capable client hardware and software required
Can leverage client resources to its full capabilities	Larger download size and thus longer loading times
Offloads work from the server to the client	Less mature .NET runtime and tooling support
Supports offline usage	

For this article, I will go for the client-side hosted application and thus choose the regular Blazor template. Click the Create button to confirm your choice.

INSPECTING THE INTERNALS

When the project is created, we already get a lot of scaffolded code to help us get started. Go ahead, simply run the application in debug mode and wait for your favorite browser to pop up. When it does, it should look something like the screenshot underneath.

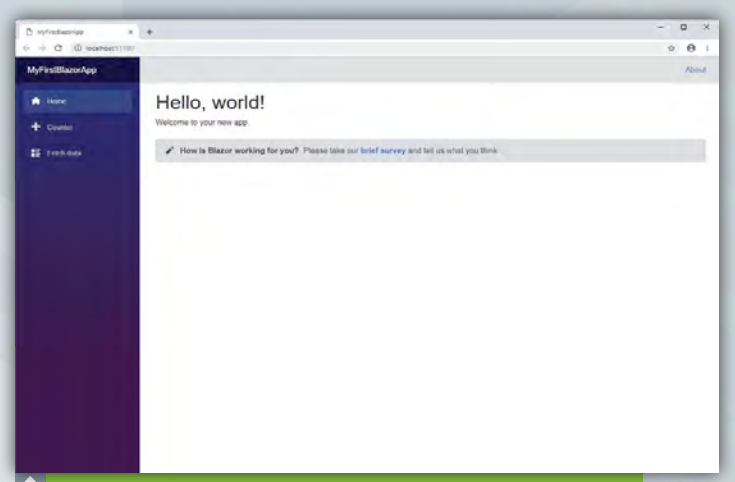


FIGURE 4: OUR VERY FIRST BLAZOR APPLICATION

There are three sections in our app: Home, Counter and Fetch data. On each of these sections you can find very simple working examples of what Blazor can do for you. It will work just like other Single Page Application (SPA) frameworks that you might know like Angular or React. Fun fact: the survey on the homepage is actually working, so if you like Blazor, please go over to the page that is linked, fill out the survey and help make it even better!

As we have learned a little bit earlier, the client-hosted Blazor application will download all the necessary components to the client and then run in the browser using WebAssembly. Open the developer tools in your browser – Chrome in my case – and inspect the network traffic when reloading. You will see that a file called mono.js is downloaded followed by mono.wasm and then MyFirstBlazorApp.dll, followed by some other DLLs that might look familiar when you have some experience with .NET. You can see this in my Chrome instance in the image underneath.

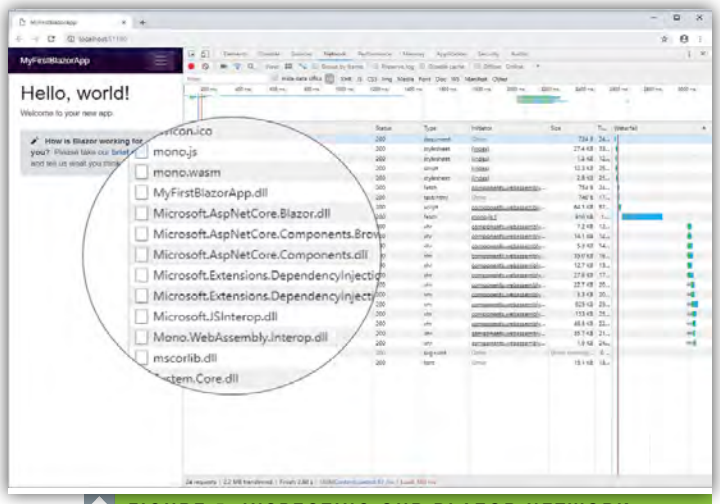


FIGURE 5: INSPECTING OUR BLAZOR NETWORK TRAFFIC IN CHROME

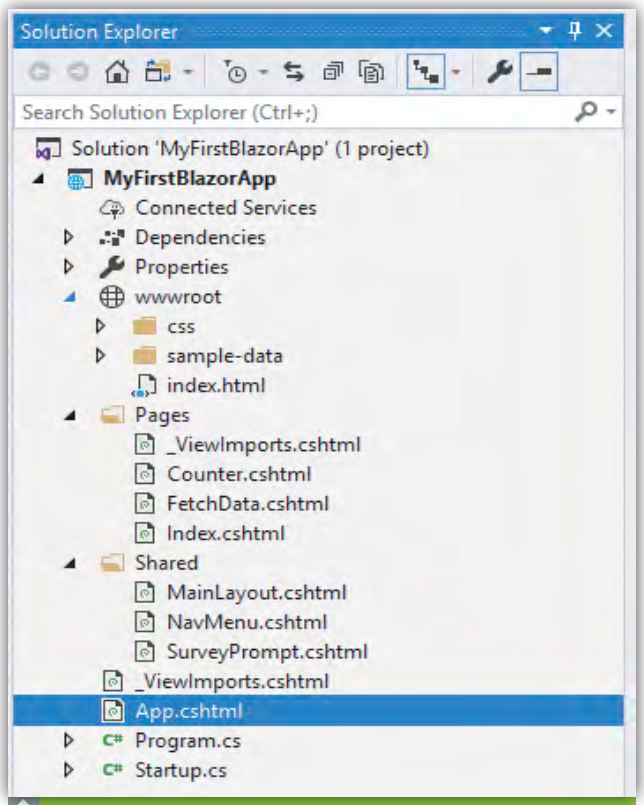


FIGURE 6: OUR BLAZOR APP PROJECT STRUCTURE

It is probably too small to read, but the total download of all of this is little over 2 MB. It's a lot, but not overwhelming, considering that you have just downloaded all the parts to run a native .NET application. Of course, whenever you start adding more features, NuGets, and DLL references, this can quickly grow larger.

SHOW ME THE CODE

Of course, this is great and all, and I am very amazed by the fact that this is all possible, but I'm a developer, so I'm even more interested in what this does to our code. First, let's have a quick look at the project structure. If we look in the Solution Explorer, we see the project structure as shown in the image underneath. When you have worked with ASP.NET solutions before, you should see no surprises here. It looks like a regular ASP.NET web application. We have the Program.cs and Startup.cs files, to bootstrap our web application and a bunch of cshtml files in different subfolders. If you think back to a little earlier, when we inspected the actual running application, you can probably deduce which files are shown where.

Let's have a closer look at some Blazor magic. Under the Pages folder open the Index.cshtml file. You can see the content in Figure 7 underneath. The contents are a mix of plain HTML with a bit of Razor and a custom tag named SurveyPrompt. I'm going to assume that you know what regular HTML is and what it does.
Razor has been part of ASP.NET quite some years now and it allows you to embed server-based code in your webpages. Since this article focusses on Blazor primarily, please have a look at this link if you want to know more about Razor: https://docs.microsoft.com/aspnet/core/mvc/views/razor. Back to the SurveyPrompt tag. This tag is a component. In Blazor you can create reusable components like this.

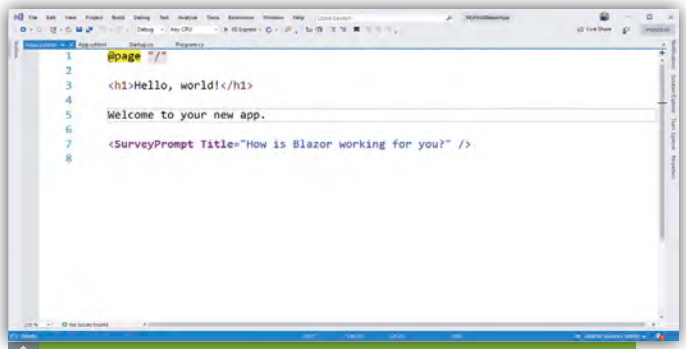


FIGURE 7: CONTENTS OF THE INDEX.CSHTML PAGE

Basically, what this is is a shorthand for another piece of HTML (and possibly code) that is embedded at the position of the tag. Open the SurveyPrompt.cshtml under the Shared folder. This shows you the internals of the component. You can see the contents of the

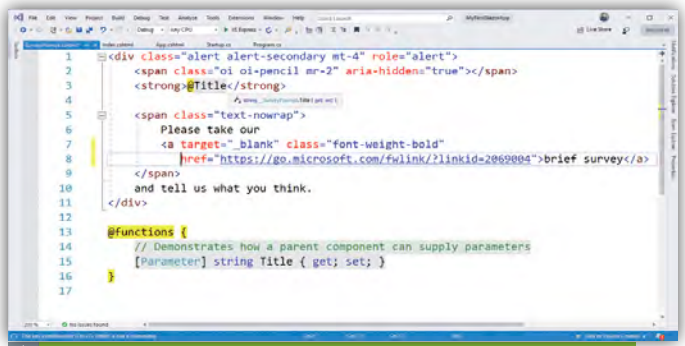


FIGURE 8: CONTENTS OF THE SURVEYPROMPT .CSHTML FILE DEFINING A COMPONENT

SurveyPrompt.cshtml file in the screenshot underneath.
In this code you can see how a simple div is included. This will be shown in the Index.cshtml page at the place of the SurveyPrompt tag. If you go back to Figure 4, you can see the actual output in the browser. But the big advantage is that you can now easily add this survey block in other parts of your application by simply putting in the SurveyPrompt tag. You could even wrap your components in a separate library and distribute it through NuGet so other can benefit from them.
Besides the HTML, you can also see the @functions Razor section. In this case, it is only used to specify a parameter. With parameters you can specify attributes on your tag that can be used in the HTML. With the SurveyPrompt, you can see a Title is specified. In Figure 7, where the tag is used in a page, you can simply provide a value for the title. That same title, is then used in the actual component to show a heading for this section.
However, this is not the only thing you can do. You can also use it to write logic here. If we go into the Counter.cshtml file under the Pages

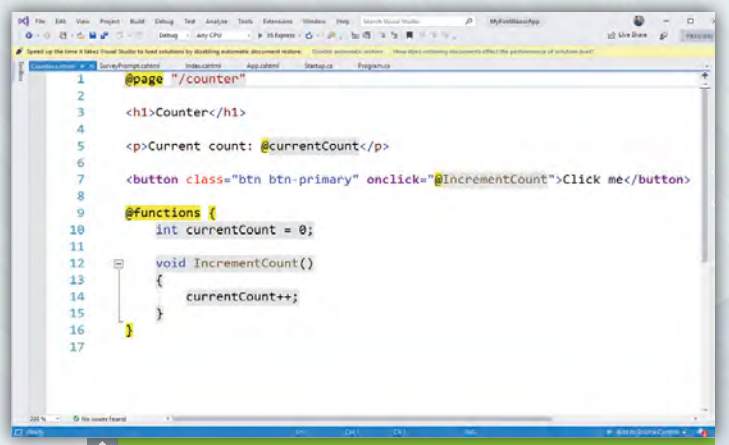


FIGURE 9: CONTENTS OF THE COUNTER.CSHTML FILE, SHOWING HOW TO WORK WITH C# LOGIC IN BLAZOR

folder, you will see the contents of Figure 9 as shown underneath.
In this file, in the functions block there is a method defines that increments the value of a variable that is also declared there. In the HTML above that, the variable and method are referenced using Razor.
As you can see, no JavaScript, just plain C# code that you know and love and when you run it, it behaves just like you would expect it to and it might've just as well be written in some form of JavaScript. This means, you can now be a full-stack developer by using C# and .NET. With this, I will leave you to it. Of course, there is a lot more to discover. How routing works, using databinding, how to communicate with actual backends and much, much more. Actually, concerning the backend communication, check out the FetchData.cshtml file by yourself. There is a great example to get you started with loading external data and showing it in the UI.

SUMMARY

In this article I have shown you what Blazor is and what the relation with WebAssembly is. We have seen how we can create our first Blazor application and how that is handled in the background by a browser. I hope you are just as amazed as I am with this technology. This stuff really opens up doors to neat stuff. Speaking of which, there is a curated list on GitHub that collects great projects built with Blazor. You can find it here: https://github.com/AdrienTorris/awesome-blazor.

The great thing is that you as a backend .NET developer can dive right in with technologies and tools that you already know and love. No need to learn a new JavaScript framework each week, you can just stick with C# and build a great looking web application. That being said, we still have the outstanding question: will WebAssembly replace JavaScript?

The short answer is: no. Personally, I don't think that is the primary intention. On the long run WASM might be able to take some of the dominance of JavaScript away, but don't forget there are a lot of people that use JavaScript and even people that like it. The ecosystem around JavaScript is very big and has been around for a long time. The support in browsers for JS at this point is far better than for WebAssembly. The WASM technology is still very young and has a lot of potential but it also has some tradeoffs in comparison to JavaScript like the larger payload, for instance. This can be solved in the future, when support becomes better. And on the other hand, WebAssembly opens up a whole new world of not being tied to one language to be used in the browser, but any! And it should outperform JavaScript. If you look at what business leaders say, they think JS will not remain a monopoly and WASM might grow overtime, but that it still a long time away.

Only time will tell, but for now just have a close look at what your use case is and choose the right solution. There's even integration possible between WebAssembly and JavaScript, so why not both?
There is a lot more to this great new tech than I can fit in this article, so if you want to get started with Blazor yourself, you can start on the website: https://blazor.net/ or, if you really want to dive into it, find the project on GitHub: https://github.com/aspnet/Blazor. Keep in mind that this technology is still experimental and might have already changed before this article is published.

GERALD VERSLUIS

Gerald Versluis (@jfversluis) is an all-round software developer, Software Engineer at Microsoft and three-time author from The Netherlands. After years of experience working with Azure, ASP.NET, Xamarin and other .NET technologies, he has been involved in a number of different projects and has been building several real-world apps and solutions.

Not only does he like to code, but he is also passionate about spreading his knowledge - as well as gaining some in the bargain. Gerald involves himself in speaking, providing training sessions and writing blogs (https://blog.verslu.is) or articles in his free time.
Twitter: @jfversluis | Website: https://gerald.verslu.is

Jan de Vries

PROACTIEF BEHEER

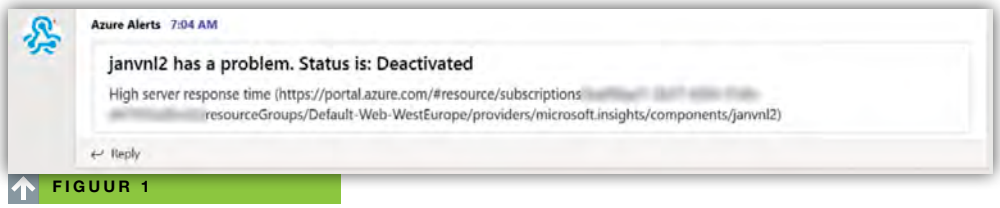
Zonder E-mail

Het is een trend die al een enkele jaren aan de gang is en hopelijk ook nog even door blijft zetten: minder e-mail versturen.
Niet alleen wordt er minder gemaild tussen mensen, maar ook tussen mensen en systemen. Persoonlijk vind ik dit enorm fijn, aangezien veel van de automatisch verstuurde e-mail met een regel in Outlook naar een map worden verplaatst waar ik ze nooit meer zie. Hier ben ik helaas niet de enige in, waardoor het versturen van alarm- of monitoring e-mails volstrekt zinloos is geworden.

In het team waar ik momenteel werkzaam ben maken we veel gebruik van monitoring e-mails welke worden gegenereerd door zelfgemaakte software of verschillende Azure Alerts. Op zich is dit fijn, want veel problemen kunnen gelijk worden geanalyseerd wanneer de e-mail binnen komt. Echter hebben de meeste collega's verschillende regels ingesteld, waardoor de e-mails niet direct worden gelezen zijn. Persoonlijk vind ik het een goede eigenschap dat men verschillende regels instelt voor binnenkomende e-mails, maar dat is een volledig andere discussie en artikel.
Een relatief nieuw medium voor het communiceren met collega's en systemen zijn de zogenaamde chat clients zoals Slack en Microsoft Teams. Waar Slack veel wordt gebruikt in de open communities, wordt Microsoft Teams vaak gebruikt binnen bedrijven voor de directe communicatie.

Beide mediums hebben de mogelijkheid om externe systemen berichten te versturen danwel te ontvangen. Deze systemen zijn dan ook meer geschikt voor het ontvangen van kritieke problemen in productie omgevingen, want chat berichten vragen direct de aandacht zijn ook bedoeld voor directe communicatie (in tegenstelling tot e-mailberichten).
Binnen Teams zie ik dan ook graag verschillende kanalen, met ieder hun eigen doel. Een van die kanalen dient dan voor het weergeven van Azure Alerts, zoals in deze afbeelding is te zien.

Naast dat alle teamleden direct worden geïnformeerd, kan er ook een discussie plaatsvinden. Alle belangstellenden kunnen hieraan deelnemen en later teruglezen.
Nu vraag jij je natuurlijk af hoe zoiets eenvoudig kan worden opgezet.



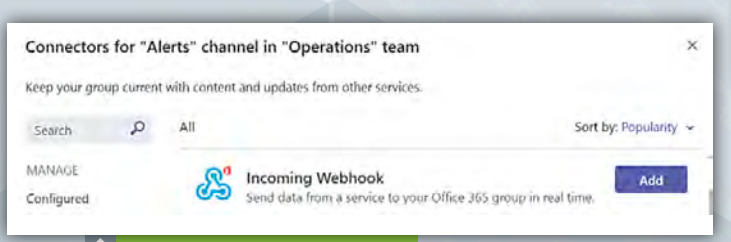
FIGUUR 1

Zonder E-mail

Helaas is dit (nog) niet triviaal en moeten er verschillende technologieën aan elkaar worden geknoopt.

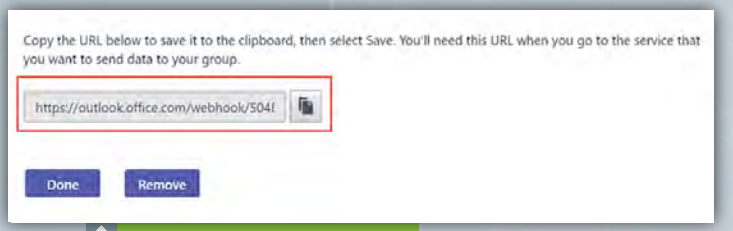
TEAMS

Het versturen van een bericht richting Teams vanuit een extern systeem is relatief eenvoudig. Er zijn tientallen verschillende connectoren waar gebruik van gemaakt kan worden. Een daarvan is de `Incoming Webhook`-connector welke ik wil gaan gebruiken omdat dit later in het proces de meeste vrijheid biedt.



FIGUUR 2

Deze connector kun je zonder al te veel problemen toevoegen en aanmaken. Na het aanmaken wordt je geconfronteerd met de URL van de webhook welke we gaan gebruiken.

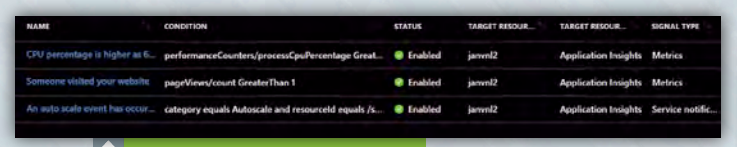


FIGUUR 3

Dit is gelukkig het enige dat we in Teams hoeven te doen voor het ontvangen van de Alerts.

APPLICATION INSIGHTS

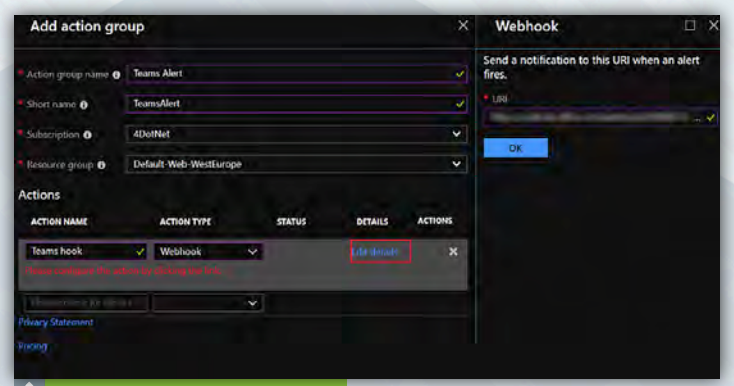
De volgende stappen zullen worden genomen binnen Application Insights.
Dit platform stelt je in staat om Alerts in te stellen wanneer er iets gebeurd in een applicatie. Zo kunnen er alerts worden ingesteld op de metrieken van de onderliggende virtuele machines, management acties in Azure, administratieve acties en ook op custom Logging informatie. Een enorm krachtige feature!
Zelf heb ik een drietal Alerts ingesteld waar ik over geïnformeerd wil worden, een hoog CPU gebruik, wanneer iemand m'n pagina bezoekt en wanneer er een autoscale event plaatsvindt.



FIGUUR 4

Een van de belangrijkste onderdelen bij het instellen van de alerts is het configureren van een correcte Action Group.

Bij het configureren van de Action Group heb ik weer gekozen voor de optie Webhook, dit omdat het nagenoeg onbeperkte vrijheid geeft. Ook kan er worden gekozen om een e-mail te sturen, Azure Logic App of Azure Function aan te roepen.

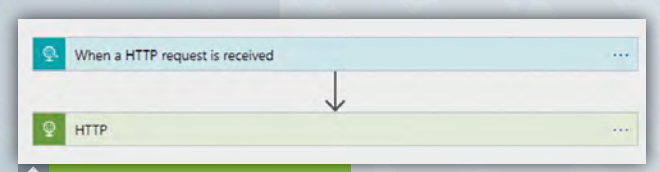


FIGUUR 5

Het direct gebruik maken van een Logic App of Azure Function maakt het leven een stuk eenvoudiger, maar vanwege de vrijheid geniet een webhook momenteel mijn voorkeur.
De eerste keer dat ik een Action Group aan het configureren was, was m'n aanname dat hier de URL van de eerder aangemaakte Teams webhook moest worden ingevuld. Dit is dus NIET het geval!
Het schema van de Azure Alert komt totaal niet overeen met wat Teams verwacht, dus zul je nooit een bericht te zien krijgen.
Wat moet er dan wel gebeuren? Het alert moet worden geconverteerd naar een formaat dat Teams wel begrijpt. Het transleren van een bericht naar een bericht in een ander formaat is iets dat heel eenvoudig gedaan kan worden met een Azure Logic App. In deze Action Group dient dus een webhook adres te worden ingevuld van een Azure Logic App.

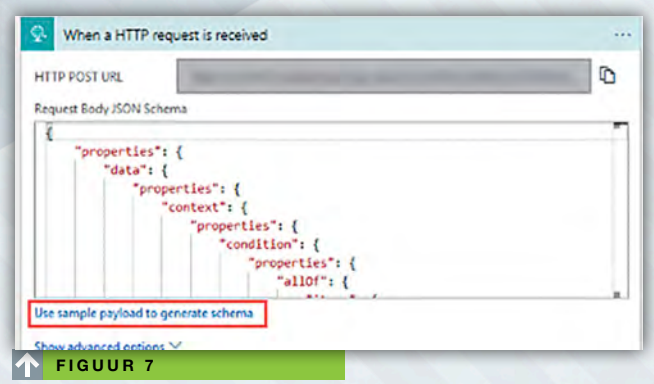
AZURE LOGIC APP

Het enige dat de Logic App moet doen is te reageren op een binnenkomend webhook bericht en zelf een POST bericht sturen richting de eerder aangemaakte Teams webhook.



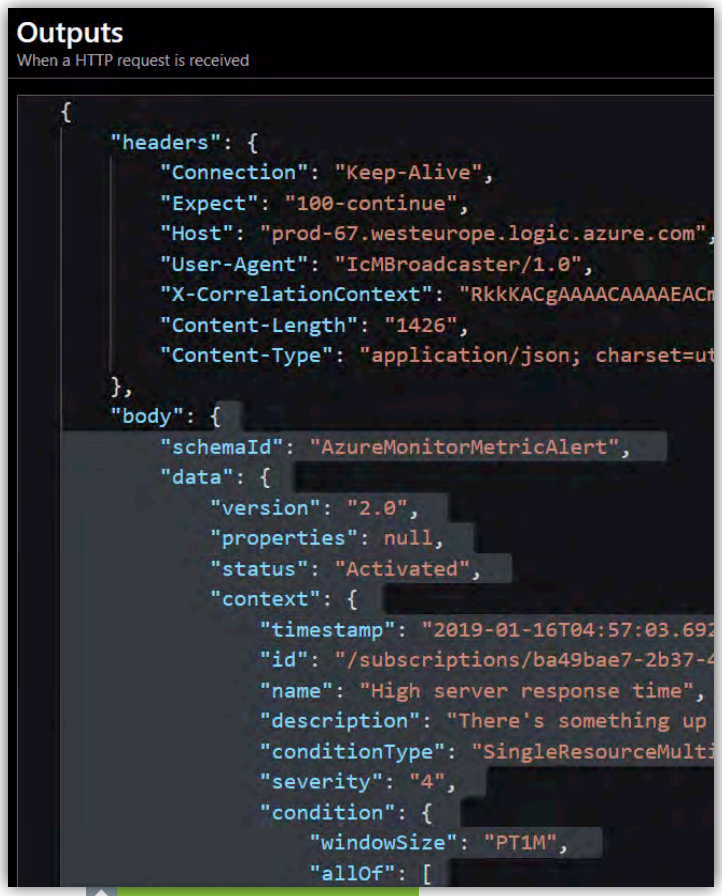
FIGUUR 6

De webhook URL wordt automatisch gegenereerd wanneer de betreffende stap wordt opgeslagen.
In de eerste stap dient het schema te worden geconfigureerd van het binnenkomende bericht. Dit is relatief eenvoudig, aangezien een bestaand bericht kan worden gebruikt om het schema te genereren met de `Use sample payload to generate schema` link.

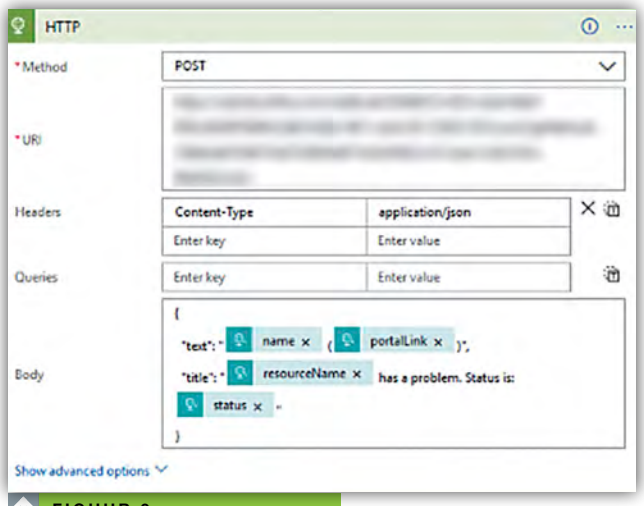


FIGUUR 7

Alle type payloads staan beschreven op de documentatie site (https://docs.microsoft.com/en-us/azure/azure-monitor/platform/activity-log-alerts-webhook).
In plaats van de documentatie in te duiken kun je er ook kiezen voor een iets pragmatischere oplossing, namelijk de payload van een gefaald bericht te gebruiken. Dat heb ik dus gedaan.
Ten tijde van het aanmaken van de Logic App waren er namelijk al enkele Alerts af gegaan welke niet konden worden verwerkt. Dit had nu als voordeel dat ik nu deze payload als sample data kon gebruiken voor het genereren van het juiste schema.



FIGUUR 8



FIGUUR 9

Ik kan me voorstellen dat dit niet de officiële way-to-go is, maar het is wel de meest eenvoudige manier
Nadat er een valide schema in de eerste stap is toegevoegd kan de tweede stap worden geconfigureerd, het afvuren van een POST bericht naar de Teams webhook. Het formaat van deze payload kan ook weer worden opgezocht, maar het volstaat al door alleen een `text` en `title` property op te geven met een bepaalde waarde.

Vergeet niet om in dit blok de eerder aangemaakte Teams webhook URL te specificeren en het `Content-Type` in `application/json` te wijzigen.
Als alles goed is ingericht zul je vanaf nu berichten in het geconfigureerde Teams kanaal te zien krijgen. De eerste stap naar een lege Inbox is nu voltooid!

TOT SLOT

In de huidige configuratie is er gekozen om alleen gebruik te maken van webhooks. Dit heeft als groot voordeel dat je enorme vrijheid hebt voor wat betreft de implementatie van de webhook. Het grootste nadeel is dat je geen gebruik maakt van standaard ingebouwde functionaliteiten.
Zo kan een Alert ook direct een connectie maken met een Logic App, dan hoef je dus niet zelf een webhook te specificeren.
Ook is het mogelijk om vanuit een Logic App direct een bericht naar Teams te sturen, dus ook weer zonder zelf een webhook te introduceren. Dit maakt het configureren natuurlijk wel vele malen eenvoudiger.
Of de ene implementatie beter is dan de andere laat ik aan jou zelf over. Het is in ieder geval goed om te weten dat er andere, betere, methoden zijn om alerting & monitoring in te regelen ten opzichte van de aloude e-mails.

JAN DE VRIES

Jan de Vries heeft in zijn loopbaan verschillende rollen gehad. Tegenwoordig werkt hij bij 4DotNet en wordt vaak ingezet als solution architect en/of ontwikkelaar op diverse projecten. Zijn focus is daarbij het opzetten van schaalbare en performante software oplossingen, waar Microsoft Azure een primaire rol in speelt. Hij verdiept zich graag in alles wat Azure te bieden heeft, waar serverless oplossingen op dit moment de meeste aandacht krijgen.
Met enige regelmaat blogt Jan op https://jan-v.nl en kunt hem volgen op @Jan_de_V

Roy Janssen

ONTWIKKELEN VOOR DE

Microsoft HoloLens (deel 2)

In SDN magazine 136 van december 2018 hebben we een begin gemaakt met het ontwikkelen voor de Microsoft HoloLens. In dit artikel wil ik jullie wat nader kennis laten maken met de examples uit de MRTK.

FIGUUR 1

EEN NIEUWE VERSIE VAN DE MRTK

Zoals in het vorige artikel al te lezen stond komen er regelmatig nieuwe versies van de MRTK beschikbaar, ook nu is dit het geval. De mensen achter de MRTK zijn al een tijdje bezig met de MRTK – vNext zoals ze deze zelf noemen. Deze opvolger moet het makkelijker maken applicaties op een breder scala aan devices te kunnen gebruiken. Deze versie is echter nog niet aan een release stadium toe. Toch hebben ze er nu al voor gekozen de vNext branch naar de Release branch te mergen. Om toch gebruik te blijven maken van de laatste stabiele versie van de MRTK moet je naar de juiste branch navigeren op github.

https://github.com/Microsoft/MixedRealityToolkit-Unity/tree/master
https://github.com/Microsoft/MixedRealityToolkit-Unity/releases

In dit artikel zal ik dan ook gebruik blijven maken van de stabiele versie 2017.4.2.0 van de toolkit.

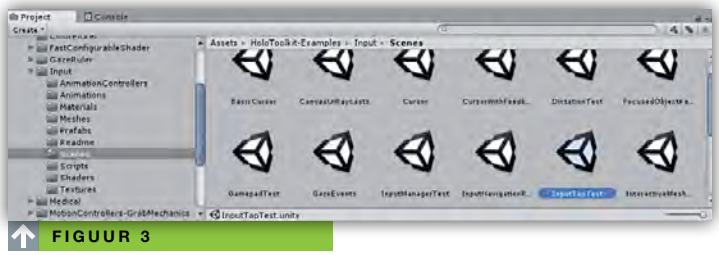
DE MRTK EXAMPLES

Deze keer ga ik niet direct door waar we vorige keer zijn gestopt. Ik kies hiervoor omdat ik de laatste tijd vaak merk dat de examples uit de MRTK bij weinig mensen bekend zijn, en dat terwijl deze enorm kunnen helpen je doel te bereiken in een HoloLens applicatie. Deze examples zijn dan wel geen volledige oplossingen, maar geven je soms net even een duwtje de juiste richting in. In deel 1 heb ik laten zien hoe de examples aan je project toe te voegen zijn, als je dat op de juiste manier hebt gedaan zal deze dan ook in je project structuur terug te vinden zijn.

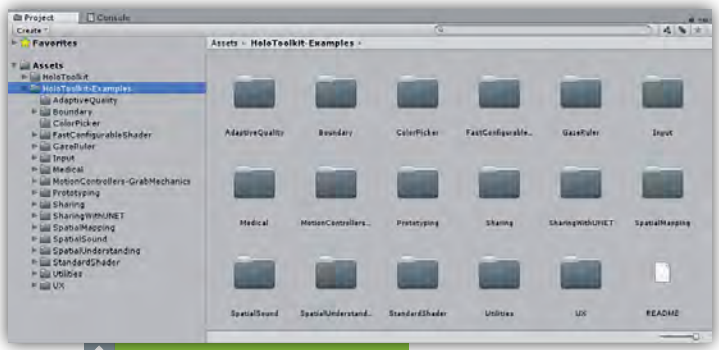
Zoals je kunt zien zijn de examples ook weer in diverse onderdelen bij elkaar gevoegd.

EEN EXAMPLE BEKIJKEN

In deze folder structuur ga je vervolgens op kijken wat je kunt gebruiken in je project. Als je bijvoorbeeld wil zien welke input mogelijkheden er zijn ga je naar de volgende map toe.

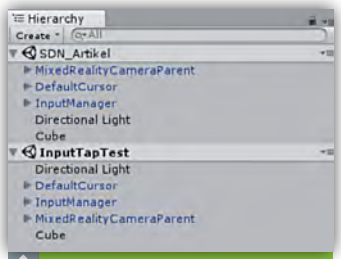


FIGUUR 3



FIGUUR 2

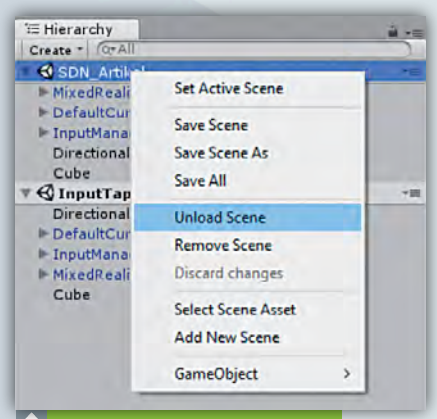
Je gaat binnen de examples altijd op zoek naar de zogenaamde Scenes. Dit zijn scenes die je direct in je Unity Hiërarchie kunt openen.



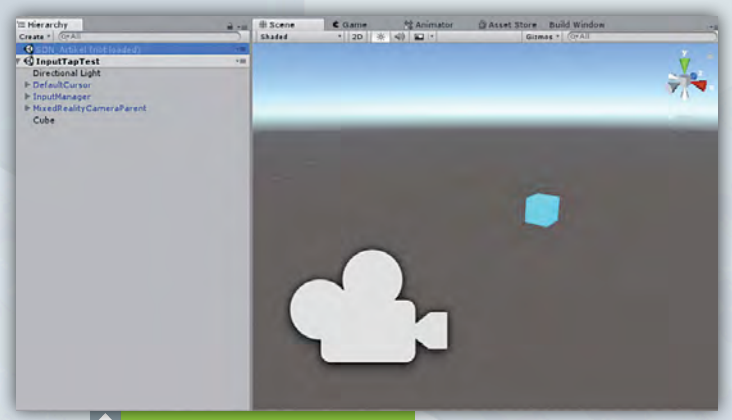
FIGUUR 4

Zoals je kunt zien heb ik hier de “InputTapTest” gekozen. Door deze naar mijn Hierarchie te slepen ziet deze er als volgt uit (figure 4).

Zoals je kunt zien zijn er nu twee scene’s actief aanwezig. Dat betekent dat ze ook beide in het “Scene” scherm actief zijn. Om hier weer wat meer inzicht te krijgen kun je ervoor kiezen de oude scene inactief te maken door via het context menu te kiezen voor “Unload Scene”. Op die manier krijg je wat meer inzicht in de example scene.

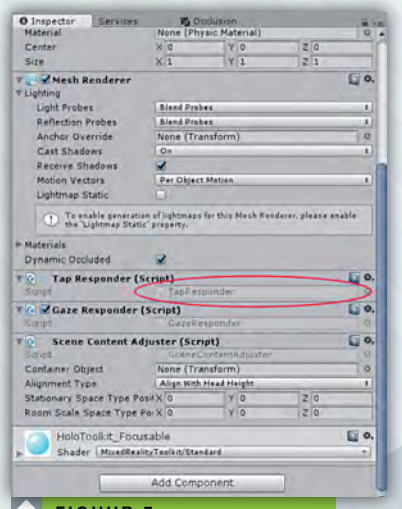


FIGUUR 5



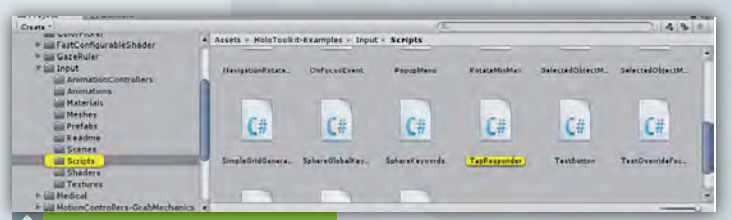
FIGUUR 6

Als je van dit voorbeeld wil leren klik je op de kubus die hier in de scene aanwezig is.



FIGUUR 7

In het “Inspector” window kun je vervolgens alle instellingen van dit object bekijken. Door op het bijbehorende script voor de “Tap Responder” te klikken zal in het script voor je opgezocht en aangeduid worden.

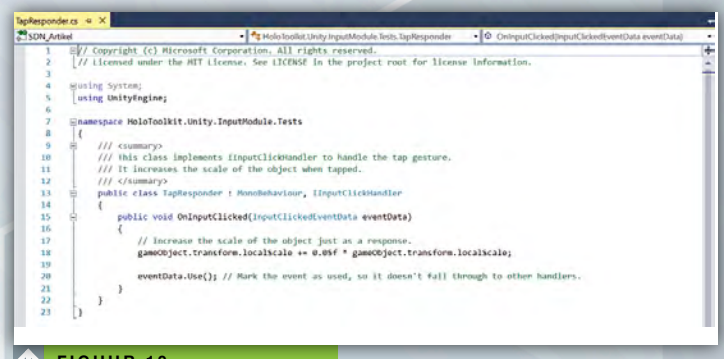


FIGUUR 8

Door hier het script aan te klikken zal de code in het inspector window zichtbaar zijn.

FIGUUR 9

Uiteraard kun je het script ook met een dubbelklik openen, dan zal het geopend worden in Visual Studio.



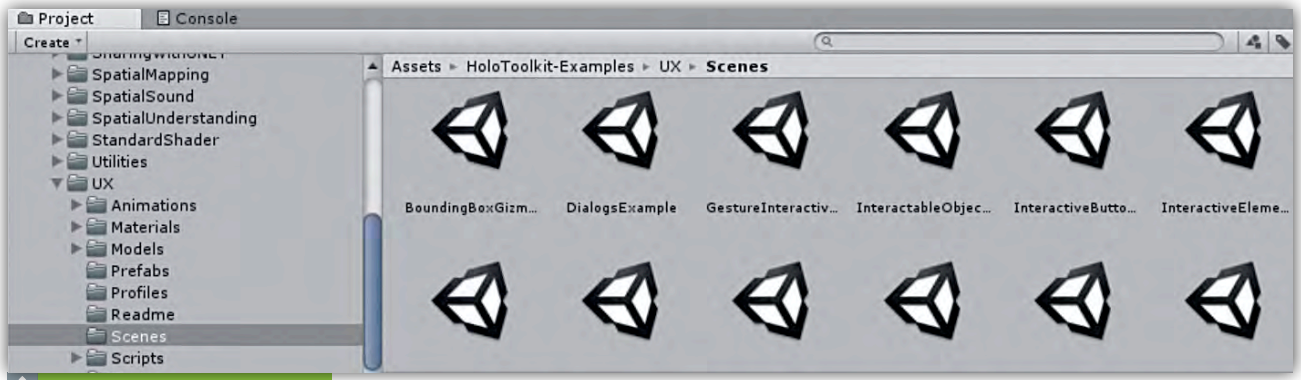
FIGUUR 10

Hier kun je dan uiteraard ook gelijk zelf wat code aanpassen. Zodra je deze opslaat zijn de wijzigingen ook direct in Unity doorgevoerd. Het is op dit moment ook de enige manier om de code aan te passen want dat is niet mogelijk vanuit Unity zelf.

MOST WANTED EXAMPLE

Het mooie is dat iedereen een ander example goed kan gebruiken om weer een stap verder te komen in een project. Toch denk ik dat voor de meeste developers de examples uit de “UX” folder heel leerzaam zijn. Voor HoloLens moet je op een andere manier naar je applicaties kijken en de meeste developers kijken dan meer naar de code dan de

user interface of user experience. In de “UX” folder zijn daarom een aantal mooie voorbeelden te vinden van user interface componenten die erg goed van pas komen in een HoloLens applicatie.



FIGUUR 11

WAT VALT ER NOG MEER TE LEREN

Dit tweede artikel heeft na de aankondiging van de HoloLens 2 wat mij betreft toch een andere wending gekregen. Want de HoloLens 2 is onderweg. Voor ons als developers is er echter nog niet veel bekend wat dat met zich meebrengt.

Ik heb gelezen dat voor het ontwikkelen hoogst waarschijnlijk gebruik gemaakt kan gaan worden van de MRTK - vNext die ik in het begin van dit artikel al even heb genoemd. Maar volledige duidelijkheid hierover is er op dit moment nog niet. Wel is het heel waarschijnlijk dat voor het ontwikkelen van HoloLens applicaties gebruik gemaakt word van Unity. Ervaring opdoen met Unity is dus op dit moment zeker aan te raden. Hopelijk kan ik in een volgend artikel al wat meer duidelijkheid geven over het ontwikkelen van HoloLens 2.

ROY JANSSEN

Roy Janssen is eigenaar van Semper IT Services en heeft al meer dan 19 jaar ervaring in IT. Hij vervult binnen kleine en grote projecten diverse rollen zoals onder andere technisch architect, lead developer en consultant. Verder is hij in te zetten als trainer, speaker en app developer. Zijn kennis in C#, WPF, XAML, NUI, PixelSense, HoloLens en andere deelt hij graag met de community.

> **ERVARING OPDOEN MET UNITY IS OP DIT MOMENT ZEKER AAN TE RADEN...**

Johan Smarius

INLEIDING IN

Blazor/Razor Components

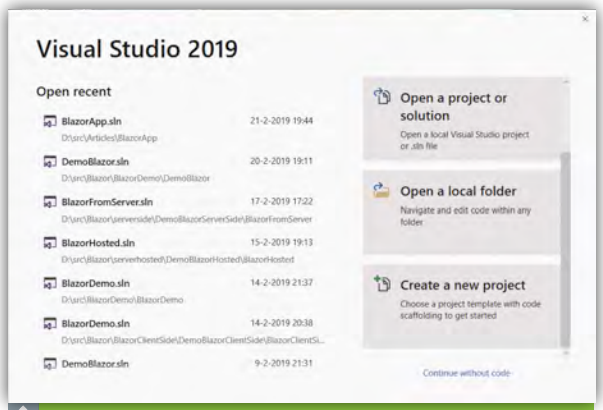
Blazor is een nieuwe experimentele technologie voor Single Page Applications van Microsoft om C# direct in de browser te kunnen draaien. Hiervoor wordt gebruik gemaakt van WebAssembly waarvoor ondersteuning bestaat in bijna alle moderne browsers. WebAssembly is een binair formaat dat uitgevoerd wordt door de browser in een sandbox die dezelfde security beperkingen heeft als JavaScript. Als je zeker wilt weten of WebAssembly ondersteund wordt door jouw browser, kun je dit altijd controleren op de site www.caniuse.com. Voor niet ondersteunde browsers kan Blazor gebruik maken van een fallback via JavaScript, maar het kan zijn dat je voor bepaalde JavaScript features aanvullende polyfills nodig hebt.

Maar waarom moeten we nu C# willen draaien in de browser? We hebben toch al een de facto standaard in de webwereld in de vorm van JavaScript. JavaScript is als taal de laatste jaren natuurlijk wel beter geworden, maar is nog lang niet zo krachtig als C#. Bovendien kan ik als Microsoft ontwikkelaar geen code delen tussen mijn ASP.NET backend en mijn SPA. Blazor maakt dit wel mogelijk. Het is natuurlijk ook fijn als ik binnen een project geen context-switch qua taal in mijn hoofd hoef te doen als ik als developer afwisselend op de backend en front-end bezig ben. Als laatste is het natuurlijk ook gewoon erg leuk om met nieuwe technologie bezig te zijn.
De code in dit artikel is gebaseerd om versie 0.8 van Blazor/Razor Components en Visual Studio 2019 Preview 3.0.

INSTALLATIE

Om gebruik te gaan maken van Blazor heb je een paar dingen nodig (zie ook https://blazor.net):

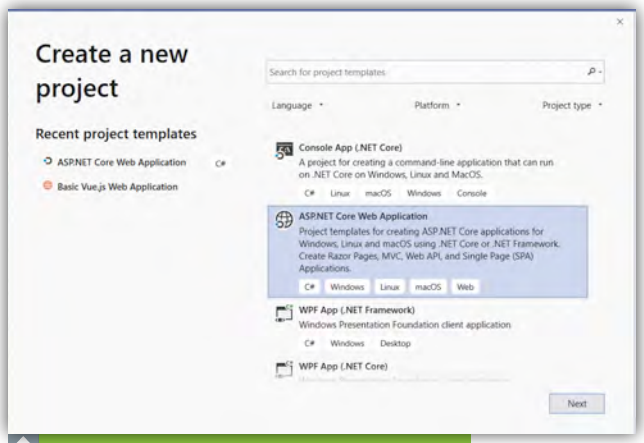
- Visual Studio 2019 Preview 2 of hoger (met de Web Development en ASP.NET Workload)
- .NET Core SDK 3.0 Preview
- De Visual Studio extensie: ASP.NET Core Blazor Language Services (https://marketplace.visualstudio.com/items?itemName=aspnet.blazor)
- Als je ook nog via de commandline wilt werken, dan moet je ook de Blazor templates installeren. Dit kun je doen door in PowerShell of de command prompt het volgende commando uit te voeren: `dotnet new -i Microsoft.AspNetCore.Blazor.Templates`



FIGUUR 1: PROJECT OPTIES OP HOMEPAGINA

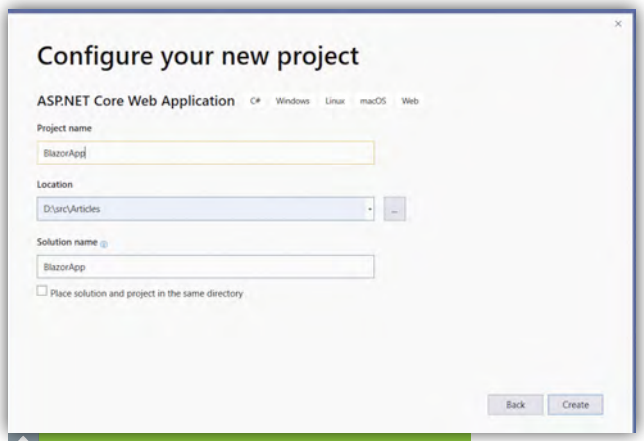
NIEUW PROJECT

Om een nieuw Blazor project aan te maken in Visual Studio, moet je de keuze maken om een nieuw project aan te maken (afbeelding 1) en te kiezen voor een ASP.NET Core Web Application te maken (afbeelding 2).

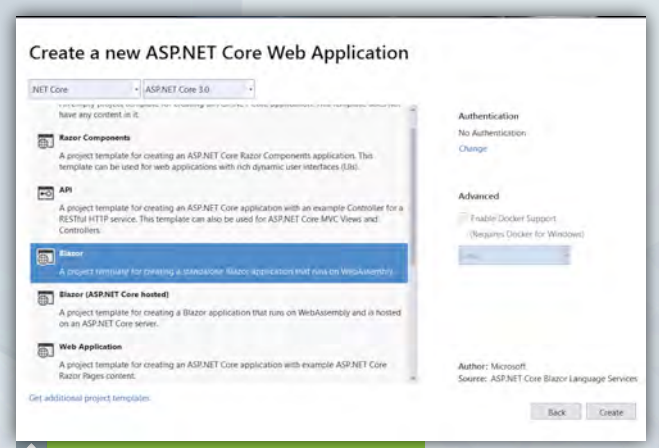


FIGUUR 2: BESCHIKBARE TEMPLATES

Je kunt nu je projectnaam en -paden opgeven (afbeelding 3)



FIGUUR 3: PROJECTINSTELLINGEN



FIGUUR 4: TEMPLATE OPTIES

Bij de template opties (afbeelding 4) heb je 3 keuzes voor Blazor:

- Blazor: volledig client side. Je hebt hierbij geen afhankelijkheid van ASP.NET Core op de server.
- Blazor: ASP.NET Core hosted. De Blazor App wordt nu gehost in ASP.NET Core, maar draait compleet lokaal in de browser.
- Razor Components: De Blazor app draait in principe volledig op de server, maar wordt door middel van SignalR en wat ondersteunend JavaScript in de browser getoond. Deze optie zal officieel, dus met ondersteuning, beschikbaar komen in .NET Core 3. Deze variant maakt geen gebruik van WebAssembly.

Voor de ontwikkelaars die liever via de command prompt werken, zijn ook templates beschikbaar voor het dotnet new commando (afbeelding 5)

```
Blazor (hosted in ASP.NET server)    blazorhosted        [C#]    Web/Blazor/Hosted
Blazor Library                       blazorlib           [C#]    Web/Blazor/Library
Blazor (Server-side in ASP.NET Core) blazorserverside    [C#]    Web/Blazor/ServerSide
Blazor (standalone)                  blazor              [C#]    Web/Blazor/Standalone
```

FIGUUR 5: TEMPLATE OPTIES

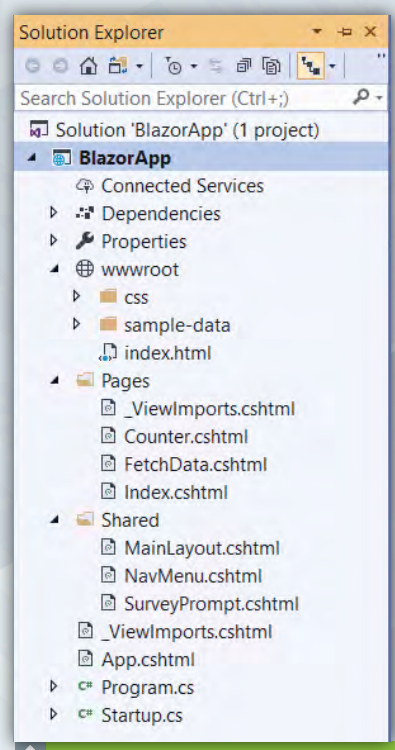
Via de command line heb je nog de extra optie “Blazor Library”. Deze kun je gebruiken om herbruikbare Blazor componenten te maken.
In de hosted variant kun je in de servercode meteen een WebAPI opnemen. Je kunt dan direct code delen tussen de WebAPI en de Blazorcode. Voor de Razor Components optie speelt dit niet echt omdat alle code echt draait op de webserver. Dit heeft natuurlijk wel tot gevolg dat je webserver veel zwaarder zal worden belast dan bij de andere opties en SignalR heeft ook zijn aandachtspunten bij het hosten.
Voor de rest van dit artikel gaan we werken met de standalone optie, want het client side deel ziet er voor alle opties hetzelfde uit. In een later artikel zal ik nog uitgebreider ingaan op de andere opties.

STRUCTUUR VAN EEN BLAZOR APP

Een Blazor app is opgebouwd uit een aantal onderdelen (afbeelding 6).

Globaal gezien hebben we hier 3 onderdelen:

- In de wwwroot zitten de statische onderdelen van de site. Door de standaard template worden geen JavaScript bestanden gegeneerd, maar deze horen ook in deze folder thuis.
- In pages zitten de Blazor componenten. De hele site wordt opgebouwd uit componenten (net zoals bij Angular en Vue.js). Elke component is verantwoordelijk voor een deel van de userinterface. In deze folder ben je dus heel vaak bezig.
- De Program en Startup zorgen voor de bootstrapping en configuratie van de applicatie analoog aan ASP.NET Core.



FIGUUR 6: STRUCTUUR BLAZOR PROJECT

COMPONENTEN

Binnen Blazor staan componenten centraal. Een component heeft (op dit moment nog) de extensie cshtml, dus net als een Razor-bestand. Dit is niet toevallig, want de naam Blazor heeft het team verzonnen omdat het een combinatie is van Browser en Razor. Door de standaardtemplate wordt een hele simpele Blazor component gemaakt: Counter.cshtml (afbeelding 7).

```
1  @page "/counter"
2
3  <h1>Counter</h1>
4
5  <p>Current count: @currentCount</p>
6
7  <button class="btn btn-primary" onclick="@IncrementCount">Click me</button>
8
9  @functions {
10     int currentCount = 0;
11
12     void IncrementCount()
13     {
14         currentCount++;
15     }
16 }
```

FIGUUR 7: COUNTER.CSHTML

In de code kun je duidelijk een paar onderdelen zien. De tag @page maakt onderdeel uit van de router van Blazor. Of anders gezegd, deze tag zorgt ervoor dat je deze component kunt oproepen in de browser door bijvoorbeeld naar http://localhost:<port>/counter te navigeren. In een component is de HTML-markup opgenomen en ook de code

voor deze component (in het @functions blok). Het is ook mogelijk om de code en de markup van elkaar los te trekken. In een vervolgartikel zal ik deze optie behandelen.

Net zoals alle andere SPA frameworks kan Blazor ook werken met databinding. Blazor ondersteunt 2 typen. De one-way binding wordt in de HTML-code aangegeven met @Variabelenaam. Natuurlijk kunnen we ook gebruik maken van 2-way binding door gebruik te maken van het bind attribuut (afbeelding 8).

```
<div>
    <label for="counter">Waarde</label>
    <input type="number" id="counter" bind="@currentCount"/>
</div>
```

FIGUUR 8: TWO-WAY BINDING

Je kunt binnen de code ook reageren op events. De meest simpele variant kun je zien in afbeelding 7. De methode IncrementCount wordt gebruikt om te reageren op het click-event van de button. De binding doe je door middel van de methodenaam achter het event te zetten zoals in de afbeelding wordt getoond. Naast de onclick heb je nog heel veel andere events tot je beschikking (afbeelding 9)

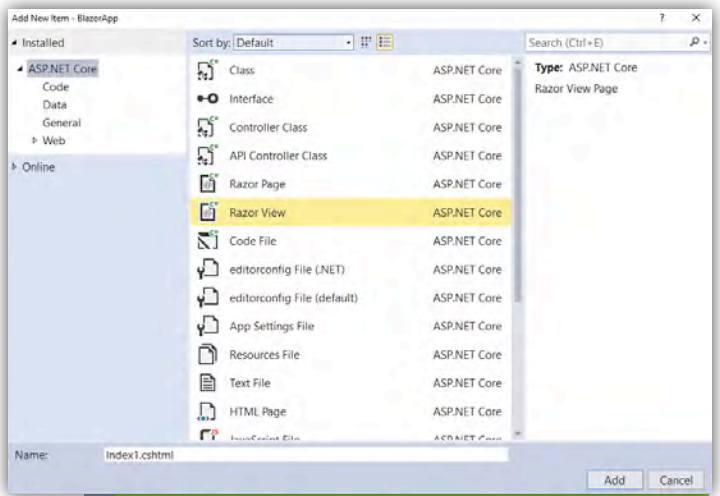
```
<button class="btn btn-primary" on>Click me</button>

@functions {
    int currentCount = 0;

    void IncrementCount()
    {
        currentCount++;
    }
}
```

onfocus, onblur, onclick, ondblclick, onfocusin, onfocusout, onkeydown, onkeypress, onkeyup, onmousedown, onmouseenter

Onfocus
Value: Script that handles event

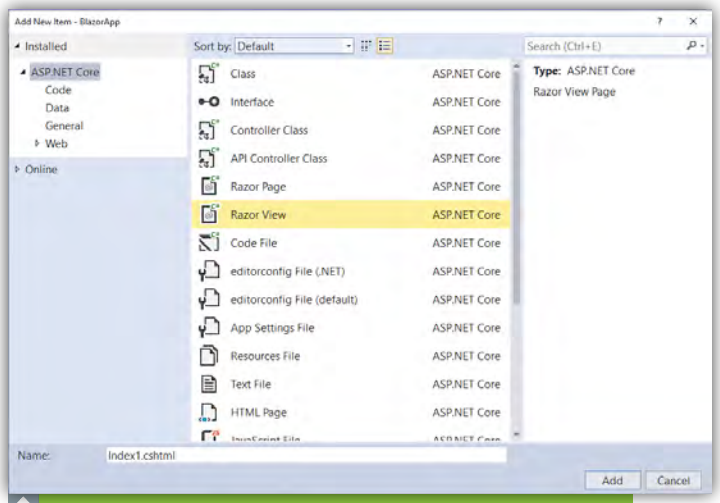
FIGUUR 9: BESCHIKBARE EVENTS

De code in afbeelding 10 geeft een voorbeeld dat events ook extra eventinformatie kunnen bevatten waar je gebruik van kunt maken. Bij de binding van het event hoef je dit niet op te geven. Je hoeft de parameter alleen bij de methode in het codeblok op te geven.



FIGUUR 10: EVENT MET EXTRA INFORMATIE

Nieuwe componenten kun je het beste aanmaken met de Razor View template (afbeelding 11) en dan de inhoud van het nieuwe bestand weggooien en vervangen door Blazor code. De verwachting is dat voor het aanmaken van Blazor componenten nog wel eigen template binnen Visual Studio beschikbaar komt, maar de bovenstaande werkwijze is op dit moment goed bruikbaar.



FIGUUR 11: AANMAKEN NIEUWE COMPONENT

Soms heb je in de HTML-code de behoefte om conditioneel een stukje HTML te tonen. Voor deze optie gebruikt Blazor gewoon Razor-syntax (afbeelding 12).

```
@if (SelectedUser != null)
{
    <div class="container">
        <div class="row">
            <label for="name">Name: </label>
            <input id="name" bind="@SelectedUser.Name"/>
        </div>
    </div>
}
```

FIGUUR 12: CONDITIES IN TEMPLATE

Voor lussen kun je gewoon gebruik maken van de Razor foreach constructie (afbeelding 13)

```
@foreach (var person in AvailableUsers)
{
    <PersonDetail person="person">
    </PersonDetail>

}
```

FIGUUR 13: LUS MET CHILDCOMPONENT

```
@functions
{
    [Parameter]
    User person { get; set; }
}
```

FIGUUR 14: PARAMETER OPNEMEN IN COMPONENT

In dit codefragment kun je ook meteen zien hoe je een andere component op kunt nemen in de layout van je component. De componentnaam kan als html-element opgenomen worden. De constructie person=”person” is nodig om een parameter door

te geven aan een component. Binnen je component kun je een parameter definiëren met de codeconstructie in afbeelding 14.

Deze parameter kun je weer gewoon als normale property in je component gebruiken.
Een laatste belangrijke optie binnen een component is dat je ook kunt reageren op een aantal lifecycle events via overrides van:

- OnInit: deze wordt aangeroepen als de component geïnitialiseerd is.
- OnParametersSet: deze wordt aangeroepen elke keer als de parameters veranderen.
- OnAfterRender: deze wordt aangeroepen als de rendering van de component met als zijn onderdelen klaar is.

Van al deze methoden zijn ook asynchrone varianten beschikbaar.

WERKEN MET WEBSERVICES

In een SPA hebben we veel interactie met restfull webservices. Binnen Blazor kunnen we hiervoor gebruik maken van de HttpClient klasse. Je kunt deze binnen je component beschikbaar krijgen door middel van dependency injection. Dependency injection is standaard beschikbaar in ASP.NET Core en dus ook in Blazor. Je moet hiervoor de code in afbeelding 15 opnemen boven in de component.

```
@inject HttpClient Http
```

FIGUUR 15: INJECT HTTPCLIENT

Binnen je methoden kun je nu gebruik maken van de property Http voor het ophalen van de resultaten van een WebAPI. In afbeelding 16 kun je zien dat de klasse het mappen van de JSON result naar een POCO-object automatisch af kan handelen.

```
protected override async Task OnInitAsync()
{
    AvailablePersons = await Http.GetJsonAsync<List<User>>("api/persons");
}
```

FIGUUR 16: AANROEPEN VAN WEBAPI

ROUTING

Een SPA framework is natuurlijk niet compleet als het geen ondersteuning beidt voor client side routing. Door de routing op de client af te handelen, kan een rond-trip naar de server voorkomen worden.
Het begin van routing heb je kunnen zien in afbeelding 7. Door gebruik te maken van de @page "/naam" kun je heel simpel een route maken. Nu heb je in de meeste single page applications de wens om tussen componenten te navigeren. Blazor biedt hier op een aantal vlakken ondersteuning voor. Als je simpelweg een link op wilt nemen in je component naar een andere component, kun je dit doet met de code constructie in afbeelding 17.

```
<NavLink class="nav-link" href="persons">
    <span class="oi oi-home" aria-hidden="true"></span> Persons
</NavLink>
```

FIGUUR 17: NAVIGATIE NAAR ANDERE COMPONENT

Dit codevoorbeeld komt uit de NavMenu component die gemaakt wordt door de standaard template. Deze code is functioneel equivalent aan Persons. Het grootte voordeel om toch NavLink te gebruiken, is dat deze laatste standaard de css-klasse active op het element plaats, als de router een match gevonden heeft bij een redirect. Binnen de links kun je gewoon gebruik maken van relatieve paden. Door de standaard template wordt standaard een HTML base element geplaatst in de index.cshtml in de folder wwwroot. Je kunt ook gebruik maken van parameters binnen de routes. Een voorbeeld van een route met een parameter is opgenomen in afbeelding 18. In deze route is ook meteen gebruik gemaakt van een route constraint om het type van de parameter expliciet te maken.

```
@page "/personedit/{id:int}"
```

FIGUUR 18: ROUTE MET PARAMETER

Vanuit je code kun je ook gebruik maken van routing om naar een andere component te navigeren. Hiervoor kun je de IUriHelper interface gebruiken. Deze kun je beschikbaar krijgen in je component door deze te injecten. In afbeelding 19 staat hier een codevoorbeeld voor.

```
@inject IUriHelper UriHelper
```

FIGUUR 19: INJECTEN IURIHELPER

Deze UriHelper kun je vervolgens gebruiken om te navigeren naar de component waarvan de route definitie in afbeelding 18 staat.

```
void Edit(User selectedUser)
{
    UriHelper.NavigateTo($"/PersonEdit/{selectedUser.Id}");
}
```

FIGUUR 20: NAVIGATIE IN DE CODE

SAMENVATTING

Blazor is nog niet volwassen, maar bevat al wel veel leuke mogelijkheden om op dit moment een simpele SPA te maken. De belangrijkste constructies om zelf met Blazor aan de slag te gaan, heb ik in dit artikel de revue laten passeren. Blazor ondersteunt echter nog veel meer. In een later artikel zal ik ingaan op wat meer geavanceerde mogelijkheden van Blazor. De ontwikkeling is nog steeds volop gaande. Bij een nieuwe versie kan het best zijn dat je bestaande code aan moet passen. Microsoft raadt op dit moment dan ook nog af om met de huidige versie van Blazor in productie te gaan. Met de komst van .NET Core 3.0 zullen we een ondersteunde versie van Razor Components gaan krijgen voor de server side, terwijl de ontwikkeling aan Blazor, de variant die afhankelijk is van WebAssembly, nog door zal gaan.
Op mijn github (https://github.com/JohanSmarius) heb ik een aantal Blazor demo projecten staan waarin de onderwerpen uit dit artikel plus nog wat meer features getoond worden. Een aantal screenshots in dit artikel zijn afkomstig uit deze code.

JOHAN SMARIUS

Johan Smarius is een ervaren (lead) software developer, architect, spreker en trainer op het Microsoft .NET vlak met meer dan 20 jaar ervaring in de IT. Hij heeft ervaring met .NET sinds versie 1.0. Johan werkt als docent informatica bij Avans Hogescholen in Breda. In zijn vrije tijd leert Johan kinderen programmeren en is instructeur EHBO. Twitter: @JohanSmarius
https://www.johansmarius.com

Sander Hoogendoorn

Truman Show Agile

LinkedIn is a nice indicator for the current state of agile. Recently, I've read posts that discuss whether it is mandatory to have a scrum master, or how to plan for unplanned work in sprints, such as defects or hotfixes. Posts are concerned with how to solve problems within the agile framework of choice, which is usually Scrum. The agile framework itself is hardly ever disputed or doubted, even though there are many cases where stepping out of the agile framework you use makes perfect sense.

Many organizations at some point decide to implement agile. The problem here is the word implement. To implement something, according to dictionaries, means to put something in effect. To implement something is a one-off process. Organizations and teams simply implement their agile framework of choice. Even though continuous learning and improvement is important in agile, organizations only seem to learn and improve within the boundaries of their agile framework, and do not retrospect on the framework itself. This is quite similar to the famous movie The Truman Show, where Jim Carrey lives in a seemingly perfect town, and has no idea about that there's life outside the movie set.

Over the past years, I have asked attendees at my talks about the agile framework they use in their organizations and teams, usually by a show of hands. Not surprisingly, the majority of attendees apply Scrum. My follow-up question to these attendees is if much has changed in the way they apply Scrum in the past two years. After thinking it over, most attendees conclude that they apply Scrum in exactly the same way as when they started with it. I find this remarkable.

Scrum is an excellent starting point for teams to move towards agile ways of working. Scrum has clear and precise documentation, and moreover, its concepts such as product owner, sprint, daily scrum, sprint demo and retrospective have become the de facto vocabulary when reasoning about agile. But once people are inside this seemingly perfect town of Scrum, they stick within its boundaries, and never retrospect on the use of the framework itself.

Nor agile, nor Scrum are intended to be implemented one-off. The Agile Manifesto states that "at regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly." The ways of working of teams is subject of evaluation and improvement. Preferably regularly. In Scrum for instance, this evaluation is done during retrospectives at the end of each sprint. According to the authoritative website scrum.org, the retrospective is "the opportunity for the Scrum team to improve." During a retrospective the Scrum master encourages the team to improve its way of working to make it more effective and enjoyable for the next sprint.

Most teams I've worked with in the past two decades indeed improve their ways of working regularly. Having retrospectives pays off as the quality of work improves. However, I've always found it intriguing that although teams evolve their ways of working in many ways, they hardly ever retrospect on the effectiveness of their agile framework.

Truman Show Agile

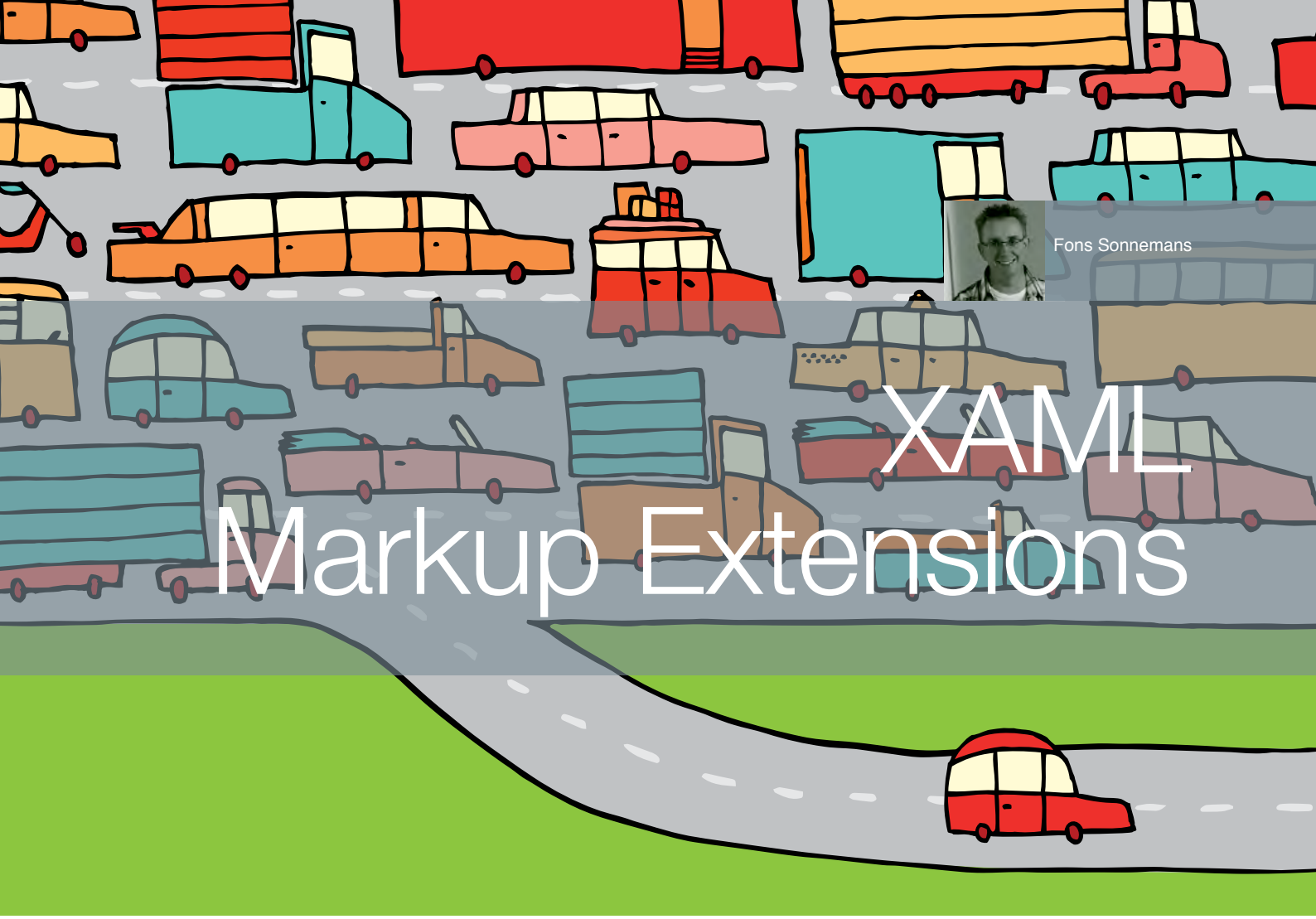
Teams don't step outside their chosen boundaries. They don't go beyond sprints, refinements and retrospectives, even though moving beyond these typical traditional agile techniques can be highly beneficial.

It is unlikely that teams improve in almost everything they do, but do not have any need for altering their agile framework, even though these frameworks leave room for improvement. Teams seem to think from inside the box, and rarely look at the box itself. It stays intact. Teams do not discuss the usefulness of the role of product owner, or of having stand-ups, or sprints. Being agile implies continuous learning and improvement. In my opinion, this includes the agile framework you are using. Once you consider that there's improvement beyond traditional agile, it does feel a lot like Jim Carrey in the closing scene of the Truman Show, where he's about to open the door between him and the rest of the world. A bit scary, but liberating.

SANDER HOOGENDOORN

Sander is an independent consultant; as a mentor, trainer, coach, software architect, developer, writer,speaker, for his company ditisagile.nl (after the Dutch title of my latest book This Is Agile). Over the past decade, in my roles as Principal Technology Officer and Global Agile Thoughtleader at Capgemini. During this period I've learned a great deal and have contributed to the innovation of software development at Capgemini and its many international clients.

TEAMS SEEM TO THINK FROM INSIDE THE BOX, AND RARELY LOOK AT THE BOX ITSELF.



Fons Sonnemans

XAML Markup Extensions

The possibility to create your own Markup Extensions where added to UWP in the Windows Falls Creators Update SDK (16299, 1709). I only used it once. I created an OnDevice Markup Extensions[1] which I added to the Windows Community Toolkit[2]. Last month Pedro Lamas[3] wrote the blog post Making the case for XAML Markup Extensions[4]. He demonstrates a FontIconExtension which he uses to shorten the XAML of an AppBarButton. I love his solution. His blog post inspired me to write some new Markup Extensions with the same goal, short XAML.

DEMO PROJECT

In the following page I have 2 Rectanges, 2 Borders and 2 Buttons inside a vertical StackPanel. The first Rectangle, Border and Button use the traditional (long) XAML syntax. The second ones use the Markup Extension which I created. The XAML is much shorter which makes it for me easier to write and read.

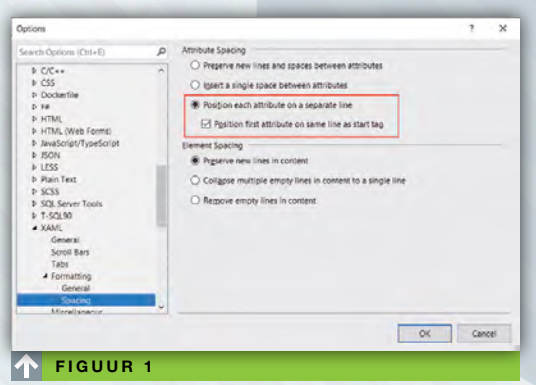
```
<Page x:Class="XamlMarkupExtensionsDemo.MainPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:local="using:XamlMarkupExtensionsDemo"
      xmlns:me="using:XamlMarkupExtensionsDemo.MarkupExtensions"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      mc:Ignorable="d"
      Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

          <Grid>
              <StackPanel Spacing="16"
                          Margin="16">

                  <Rectangle Height="80">
                      <Rectangle.Fill>
                          <ImageBrush ImageSource="https://png.pngtree.com/thumb_back/fw800/back_pic/03/70/72/5257b6c12d89875.jpg" />
                      </Rectangle.Fill>
                  </Rectangle>

                  <Rectangle Height="80"
                             Fill="{me:ImageBrush Source='https://png.pngtree.com/thumb_back/fw800/back_pic/03/70/72/5257b6c12d89875.jpg' }" />

                  <Border Height="80">
                      <Border.Background>
                          <LinearGradientBrush EndPoint="0.5,1"
                                               StartPoint="0.5,0">
                              <GradientStop Color="Red" />
                              <GradientStop Color="Green"
```

XAML Markup Extensions

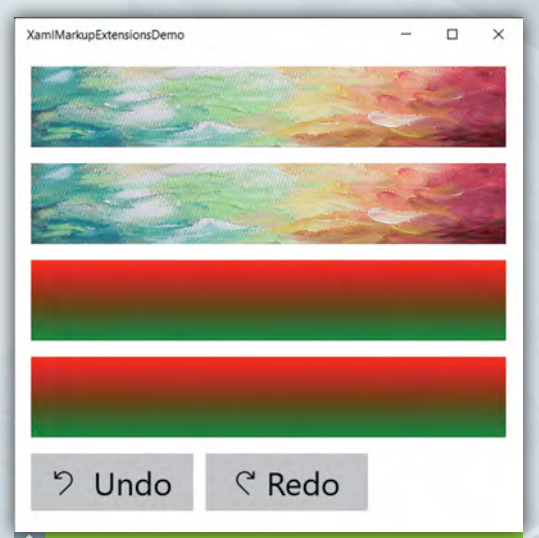
```
30.                                         Offset="1" />
31.                         </LinearGradientBrush>
32.                     </Border.Background>
33.                 </Border>
34.
35.                 <Border Height="80"
36.                         Background="{me:GradientBrush
  Start=Red, Stop=Green}" />
37.
38.                 <StackPanel Orientation="Horizontal"
39.                             Spacing="12">
40.
41.                     <Button Click="ButtonUndo_Click"
42.                             Width="160"
43.                             FontSize="32">
44.                         <Button.Content>
45.                             <TextBlock>
46.                                 <Run FontFamily="Segoe
  MDL2 Assets"
47.                                      FontSize="24"
48.                                      Text="&#xE7A7;" />
49.                                 <Run Text=" Undo" />
50.                             </TextBlock>
51.                         </Button.Content>
52.                         <Button.KeyboardAccelerators>
53.                             <KeyboardAccelerator
  Key="Z"
54.
  Modifiers="Control" />
55.                         </Button.KeyboardAccelerators>
56.                     </Button>
57.
58.                     <Button Content="{me:GlyphAndText
  Glyph='&#xE7A6;', GlyphFontSize=24, Text='Redo'}"
59.
  me:KeyboardAcceleratorExtension.KeyboardAccelerator="{me:K
  eyboardAccelerator Key=Y, Modifiers=Control}"
60.                             Click="ButtonRedo_Click"
61.                             Width="160"
62.                             FontSize="32" />
63.                 </StackPanel>
64.             </StackPanel>
65.         </Grid>
66.     </Page>
```

The Rectangle uses an ImageBrush Markup Extension which reduces 3 lines of XAML markup. The Border uses a GradientBrush Markup Extension which reduces 8 lines of XAML markup. The Button uses a GlyphAndText and a KeyboardAccelerator Markup Extensions which reduces 10 lines of XAML markup. I have set up my Visual Studio/ Blend Text Editor to 'position each attribute on a separate line'.

This page renders like this:



FIGUUR 1



FIGUUR 2

IMAGEBRUSH MARKUP EXTENSION

The ImageBrushExtension can be used to create an ImageBrush with a Source, Stretch and Opacity.

```
1. [MarkupExtensionReturnType(ReturnType =
  typeof(ImageBrush))]
2. public class ImageBrushExtension : MarkupExtension {
3.
4.     public string Source { get; set; }
5.     public Stretch Stretch { get; set; } = Stretch.Fill;
6.     public double Opacity { get; set; } = 1.0;
7.
8.     protected override object ProvideValue() {
9.         return new ImageBrush() {
10.                 Stretch = this.Stretch,
11.                 Opacity = this.Opacity,
12.                 ImageSource = new BitmapImage {
13.                     UriSource = new Uri(Source),
14.                 }
15.             };
16.         }
17.     }
```

GRADIENTBRUSH MARKUP EXTENSION

The GradientBrushExtension can be used to create a GradientBrush with Start and Stop colors and points.

```
1. [MarkupExtensionReturnType(ReturnType =
  typeof(GradientBrush))]
2. public class GradientBrushExtension : MarkupExtension {
3.
4.     public Color Start { get; set; }
5.     public Color Stop { get; set; }
6.     public Point StartPoint { get; set; } = new Point(0.5,
  0);
7.     public Point EndPoint { get; set; } = new Point(0.5,
  1);
8.
9.     protected override object ProvideValue() {
10.             var gb = new LinearGradientBrush() {
11.                 StartPoint = this.StartPoint,
12.                 EndPoint = this.EndPoint,
13.             };
14.             gb.GradientStops.Add(new GradientStop() {
15.                 Color = Start,
16.             });
17.             gb.GradientStops.Add(new GradientStop() {
18.                 Color = Stop,
19.                 Offset = 1,
20.             });
21.             return gb;
22.         }
23.     }
```

GLYPHANDTEXT MARKUP EXTENSION

The GlyphAndTextExtension can be used to create a TextBlock with a Glyph and a Text.

```
1. [MarkupExtensionReturnType(ReturnType = typeof(TextBlock))]
2. public class GlyphAndTextExtension : MarkupExtension {
3.
4.     public string Glyph { get; set; }
5.     public double GlyphFontSize { get; set; } = 14;
6.     public string Text { get; set; }
7.     public FontFamily FontFamily { get; set; } = new
   FontFamily("Segoe MDL2 Assets");
8.
9.     protected override object ProvideValue() {
10.             var tb = new TextBlock();
11.             tb.Inlines.Add(new Run {
12.                 Text = this.Glyph,
13.                 FontSize = this.GlyphFontSize,
14.                 FontFamily = this.FontFamily,
15.             });
16.             tb.Inlines.Add(new Run { Text = " " + this.
   Text });
17.             return tb;
18.         }
19.     }
```

KEYBOARDACCELERATOR MARKUP EXTENSION

The KeyboardAcceleratorExtension can be used to a create KeyboardAccelerator with a Key and Modifiers. It also contains a KeyboardAccelerator attached property which allows you to add it to the KeyboardAccelerators collection of a UIElement.

```
1. [MarkupExtensionReturnType(ReturnType =
   typeof(KeyboardAccelerator))]
2. public class KeyboardAcceleratorExtension :
   MarkupExtension {
3.
4.     /// <summary>
5.     /// Identifies the KeyboardAccelerator attachted
   property. This enables animation, styling, binding, etc...
6.     /// </summary>
7.     public static readonly DependencyProperty
   KeyboardAcceleratorProperty =
8.         DependencyProperty.RegisterAttached("KeyboardAccel
   erator",
9.
   typeof(UIElement),
10.                                                 typeof(
   KeyboardAcceleratorExtension),
11.                                                 new
   PropertyMetadata(default(object),
   OnKeyboardAcceleratorChanged));
12.
13.         /// <summary>
14.         /// Gets or sets the virtual key (used in
   conjunction with one or more modifier keys)
15.         /// for a keyboard shortcut (accelerator).
16.         /// </summary>
17.         public VirtualKey Key { get; set; }
18.
19.         /// <summary>
20.         /// Gets or sets the virtual key used to modify
   another keypress for a keyboard shortcut
21.         /// (accelerator).
22.         /// </summary>
23.         public VirtualKeyModifiers Modifiers { get;
   set; }
24.
25.         /// <summary>
26.         /// Gets the value of the KeyboardAccelerator
   attached property from the specified FrameworkElement.
27.         /// </summary>
28.         public static object GetKeyboardAccelerator(Dep
   endencyObject obj) {
29.             return (object)obj.GetValue(KeyboardAccelera
   torProperty);
30.         }
31.
32.         /// <summary>
33.         /// KeyboardAccelerator changed handler.
34.         /// </summary>
35.         /// <param name="d">FrameworkElement that changed
   its KeyboardAccelerator attached property.</param>
36.         /// <param name="e">DependencyPropertyChangedEven
   tArgs with the new and old value.</param>
37.         private static void OnKeyboardAcceleratorChanged(
   DependencyObject d, DependencyPropertyChangedEventArgs e) {
38.             if (d is FrameworkElement source) {
39.                 var value = (KeyboardAccelerator)
   e.NewValue;
40.                 source.KeyboardAccelerators.Add(value);
41.             }
42.         }
43.
44.         /// <summary>
45.         /// Returns a KeyboardAccelerator with the Key
   and Modifiers set
46.         /// </summary>
47.         /// <returns></returns>
48.         protected override object ProvideValue() {
49.             return new KeyboardAccelerator() {
50.                 Key = this.Key,
51.                 Modifiers = this.Modifiers,
52.             };
53.         }
54.
55.         /// <summary>
56.         /// Sets the value of the KeyboardAccelerator
   attached property to the specified FrameworkElement.
57.         /// </summary>
58.         /// <param name="obj">The object on which to set
   the KeyboardAccelerator attached property.</param>
59.         /// <param name="value">The property value to
   set.</param>
60.         public static void SetKeyboardAccelerator(Depende
   ncyObject obj, object value) {
61.             obj.SetValue(KeyboardAcceleratorProperty,
   value);
62.         }
63.     }
```

I have published the code on GitHub[5]. I hope you like it. Maybe it will inspire you too to create Markup Extensions.

LINKS

1) https://docs.microsoft.com/en-us/windows/communitytoolkit/extensions/ondevicemarkup
2) https://github.com/windows-toolkit/WindowsCommunityToolkit
3) https://www.pedrolamas.com
4) https://www.pedrolamas.com/2019/03/31/making-the-case-for-xaml-markup-extensions
5) https://github.com/sonnemaf/XamlMarkupExtensionsDemo

FONS SONNEMANS

Fons is a well-known trainer & speaker, he is known for his up-to-date and expert knowledge of the latest Microsoft technologies and therefore rewarded by Microsoft as MVP. He has been using Microsoft .NET technology intensively since 2001, because it allows him to apply his knowledge and experience to business-critical situations. As a trainer, speaker, coach and developer at Reflection IT, he likes to share this knowledge and experience on a daily basis. Fons also has a passion for developing Windows apps. He has created numerous apps, some of them are very successful.