Ontwerpdocument Inside Airbnb

Datagedreven webapplicatie
getest voor high performance en security

Rowan Paul Flynn (637588)
ITA-NotS-A-f 2021
Course NotS WAPP

Marcel Verheij

v1.0
9 juni 2022

Inhoudsopgave

1 Inleiding

Dit document is een ontwerpdocument voor de webapplicatie Inside Airbnb. Met deze webapplicatie kan inzicht in het gebruik van Airbnb locaties in Amsterdam gegeven worden. Er kan aan de hand van een map de spreiding van alle listings bekeken worden met mogelijkheid om te filteren. Ook zijn er een aantal afgeschermde statistieken die bereikbaar zijn voor administrators. De applicatie is getest op performance en security.

2 Functioneel ontwerp

Er zijn een aantal use cases opgegeven door de opdrachtgever:

Use case	Prioriteit
Registreren en inloggen	Must
Filter op prijs	Must
Filter op buurt	Must
Kaart is clickable, details rechts op pagina, maakt gebruik van de mapbox API	Must
Details per item waarop gefilterd is: #overnachtingen, #opbrengst in de maand	Must
Er moeten rollen toegevoegd en toegekend worden aan geregistreerde gebruikers	Must
Resultaten zoals trends, totalen, gemiddelden, etc. worden weergegeven in charts, alleen te bekijken voor ADMINS. Denk daarbij aan bv. Gemiddelde beschikbaarheid per maand, gemiddelde beschikbaarheid per buurt, overzicht van gemiddelde huurprijs per buurt. Andere managementoverzichten zijn ook mogelijk, ga daarvoor op zoek naar online voorbeelden	Must
Locaties van zoekresultaten zichtbaar op kaart	Could
Layout idem als insideairbnb.com	Could

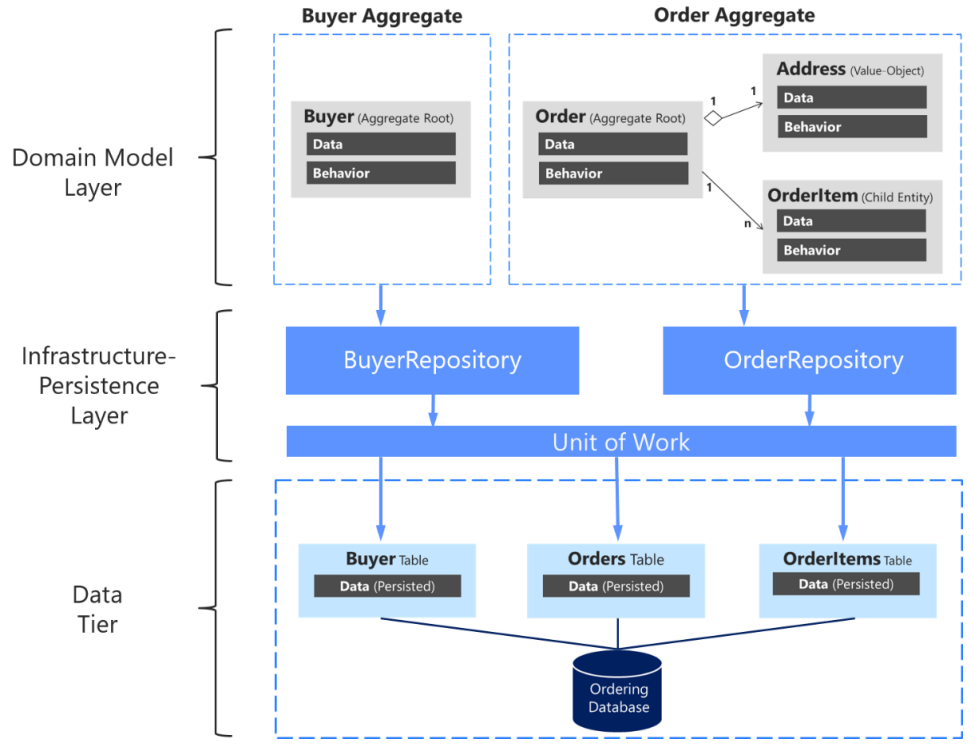
Tabel 1 Use cases

3 Technisch ontwerp

In dit hoofdstuk bespreek ik het technisch ontwerp van de webapplicatie.

De webapplicatie is een C# .NET 6.0 applicatie die gebruik maakt van Blazor, een framework voor .NET om websites te maken. Specifiek is de applicatie een *Blazor Web Assembly Hosted* applicatie. Dit betekend dat er een server is die de frontend Blazor WASM client served. Ook is er een *Class Library* die modellen bevat die gebruikt worden door zowel de client als server.

De server is gemaakt volgens het *repository pattern*, dit betekend dat er een laag zit tussen de domein modellen en de database. In dit geval is er een controller die een repository aanroept die vervolgens een database aanroept. Door gebruik te maken van het repository pattern is het makkelijk om de database te veranderen doordat deze niet direct aangeroepen wordt in de controller.



Figuur 1: Het repository pattern

Voor authenticatie is er gebruik gemaakt van Microsoft Azure Active Directory. Met behulp van de Microsoft.Authentication.WebAssembly.Msal kan er worden ingelogd op de frontend. Voor authenticated routes op de API stuurt de frontend een bearer token mee die de backend controleert.

3.1 Packages

Er zijn verschillende *nuget packages* gebruikt die in de tabel hieronder te zien zijn, inclusief gebruikte versie en waarvoor het gebruikt is.

Package naam	Versie	Project	Waarvoor
Microsoft.AspNetCore.Components.WebAssembly	6.0.5	Client	Onderdeel van Blazor
Microsoft.AspNetCore.Components.WebAssembly.DevServer	6.0.5	Client	Onderdeel van Blazor
Microsoft.Authentication.WebAssembly.Msal	6.0.5	Client	Authenticeren bij Azure AD
Microsoft.Extensions.Http	6.0.0	Client	Onderdeel van Blazor
GeoJSON.Text	1.0.0	Server	Maken van GeoJson voor mapbox
Microsoft.AspNetCore.Components.WebAssembly.Server	6.0.5	Server	Onderdeel van Blazor
Microsoft.EntityFrameworkCore.Design	6.0.5	Server	Database model maken
Microsoft.EntityFrameworkCore.SqlServer	6.0.5	Server	Ef core met SQL server
Microsoft.EntityFrameworkCore.Tools	6.0.5	Server	Database aanroepen
Microsoft.Extensions.Caching.StackExchangeRedis	6.0.5	Server	Redis (cache)
Microsoft.VisualStudio.Web.CodeGeneration.Design	6.0.5	Server	Genereren van code
Swashbuckle.AspNetCore	6.3.1	Server	Swagger

Tabel 2 Packages

Ook wordt er gebruik gemaakt van twee javascript scripts; namelijk mapbox voor de interactieve kaart en chart.js voor de interactieve kaarten op de admin pagina. Deze worden aangesproken via JSinterop in de frontend.

4 Performance

In dit hoofdstuk bespreek ik de verschillende performance optimalisaties die zijn uitgevoerd op de webapplicatie.

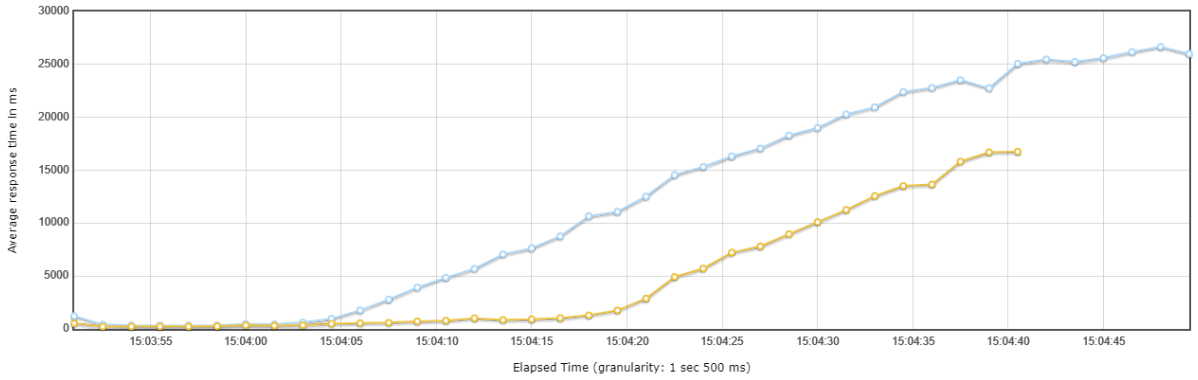
4.1 Baseline

Als eerste heb ik een baseline test gedaan om te kijken hoe goed de applicatie is zonder performance verbeteringen. De tests heb ik gedaan met Jmeter met de configuratie die te zien is in de tabel hieronder.

Setting	Value
Target rate	10
Ramp up time	30
Ramp up step count	10
Hold target rate	5

Tabel 3 Instellingen Jmeter

De resultaten van de eerste test zijn als volgt:



Figuur 2 De response times over tijd

Route	Gemiddelde response time (ms)	Median response time (ms)
/api/listings?geojson=true	16999.51	22593.00
/api/statistics/neighbourhoods	4349.63	923.00

Tabel 4 De response times van de base test

De aangesproken endpoints zijn:

- GET /api/listings?geojson=true

- GET /api/statistics/neighbourhoods

De listings route geeft alle listings in geojson formaat terug en de statistics route geeft in chart.js formaat de hoeveelheid listings per neighbourhood.

Zoals te zien is de operational ceiling bereikt na ongeveer 20 seconden en gaat de response time dan omhoog naar 20.000 ms. De response time van de listings route is duidelijk een stuk hoger wat ook logisch is aangezien deze ~5000 listings ophaalt vergeleken met maar een tiental resultaten bij neighbourhoods.

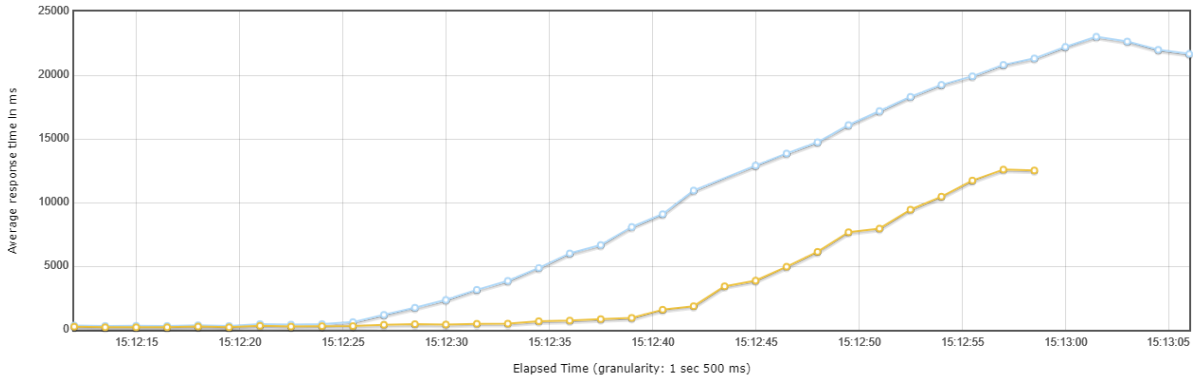
4.2 Indexes

Door indexes (zoals primary keys) toe te voegen aan de database hoeft de database minder I/O operaties uit te voeren bij bijvoorbeeld een SELECT query (Microsoft Docs Contributors, 2022). Om dit te proberen heb ik primary keys toegevoegd aan elke tabel die gebruikt wordt in de applicatie. Als er een query wordt uitgevoerd direct op de database dan is er bij de meeste tabellen wel een verschil te zien:

	Zonder indexes	Met indexes
Listings	25ms	22ms
Neighbourhoods	2ms	2ms
Reviews	6,5s	5,9s

Tabel 5

Uit deze test kwam het volgende:



Figuur 3 Met caching

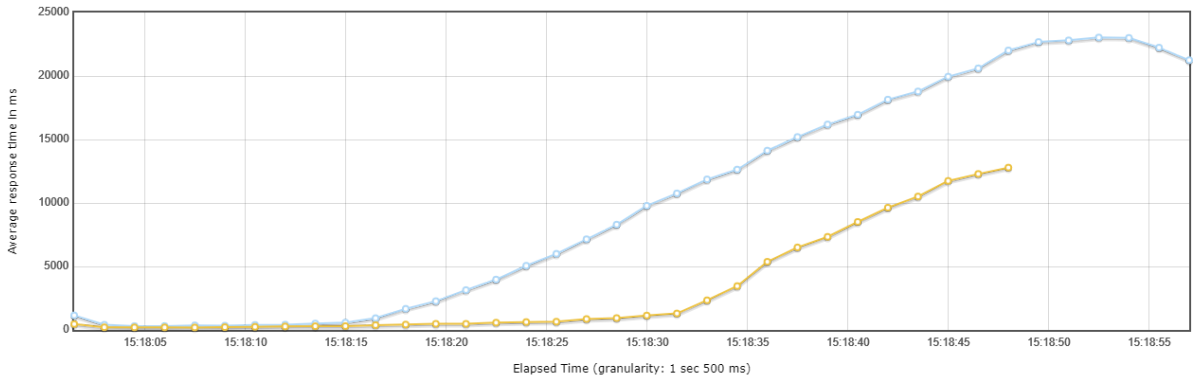
Route	Gemiddelde response time (ms)	Median response time (ms)
/api/listings?geojson=true	13697.71	19409.00
/api/statistics/neighbourhoods	2845.24	735.00

Tabel 6 De response times met caching

Zoals te zien is is de response time iets beter dan de baseline, maar wordt de operational ceiling nog steeds snel bereikt. Het toevoegen van indexes heeft dus wel iets geholpen, maar niet enorm.

4.3 AsNoTracking

Standaard worden de query's de return entity types bijgehouden zodat als je een verandering maakt deze opgeslagen kunnen worden in de database. Maar als je allen read operaties uitvoert zoals bij deze applicatie hoeft dat dus niet.



Figuur 4 De response times over time

Route	Gemiddelde response time (ms)	Median response time (ms)
/api/listings?geojson=true	13658.88	18398.00
/api/statistics/neighbourhoods	2571.19	647.00

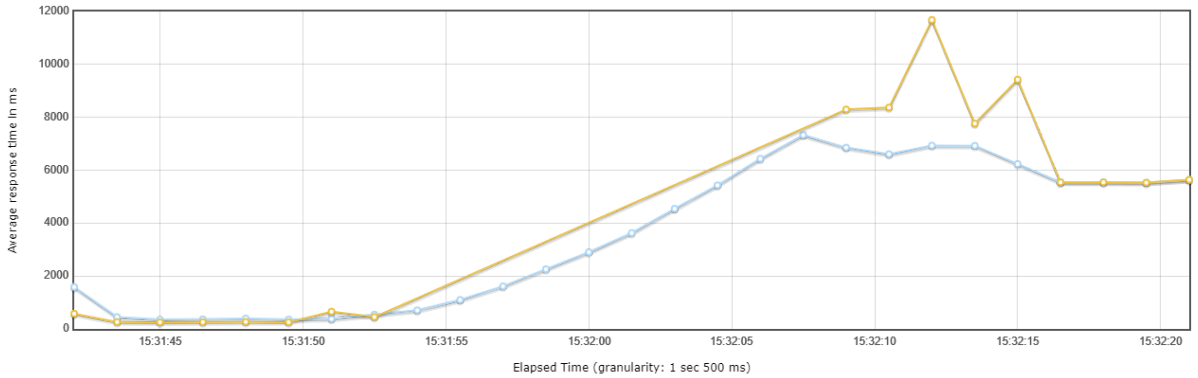
Tabel 7 Response times in milliseconden

De responstijd is bij een paar honderd milliseconden omlaag gegaan en bij de median response tijd zelfs met 200 ms omlaag gegaan. Ook is te zien in figuur 4 dat de operational ceiling een stuk later is dan voordat er asNoTracking werd toegepast.

4.4 Caching

Nu moet de applicatie bij elke request de SQL database aanspreken. Dit kan ook anders door gebruik te maken van caching. Hierdoor wordt er maar één keer een request gedaan naar de SQL database en vervolgens wordt er een snellere database gebruikt die de data van die specifieke request heeft.

Ik heb gebruik gemaakt van Redis, deze laat webapplicaties snel data bewaren in het geheugen van een computer. Dit is vele malen sneller dan data halen van een harde schijf of SSD en kan dus behoorlijk veel performance verbetering opleveren.



Figuur 5 Met caching

Route	Gemiddelde response time (ms)	Median response time (ms)
/api/listings?geojson=true	4791.45	5393.00
/api/statistics/neighbourhoods	7646.60	5741.00

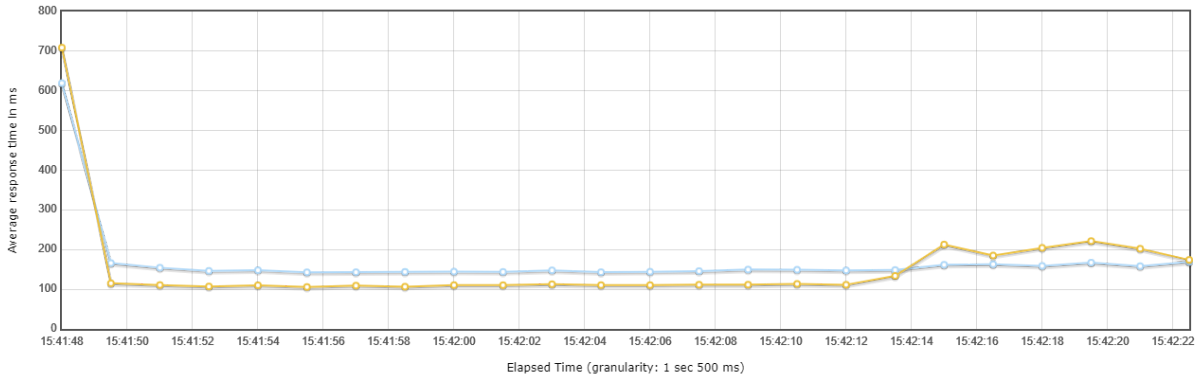
Tabel 8 De response times met caching

Bij het toevoegen cache wordt de operational ceiling een stuk lager dan bij de vorige meting. van ongeveer 25 naar 15 seconden. Dit komt omdat ongeveer 70% van de requests falen met een error. Dit komt omdat de Redis database een timeout exception geeft. Wel gaat de gemiddelde response time naar beneden met bijna de helft voor de listings api, terwijl deze bij de andere url juist omhoog gaat. Dit komt omdat deze request vaker tegen de timeout is aangelopen en dus vaker een error gaf.

Als je kijkt naar de response time van een individuele request, bijvoorbeeld de /api/neighbourhoods route gaat die van 290 ms naar 23 ms na de eerste keer laden. De individuele requests zijn dus wel een stuk sneller geworden met caching maar door de redis connection timeout wordt er bij caching langer over gedaan.

4.3 Gericht ophalen kolommen

Op dit moment worden bij requests alle kolommen opgehaald, maar dit is niet nodig. Door alleen de kolommen uit de database op te halen die ook echt nodig zijn hoeft er minder data verstuurd te worden en dus kan dit een performance verbetering opleveren.



Figuur 6 De response times over time

Route	Gemiddelde response time (ms)	Median response time (ms)
/api/listings?geojson=true	157.00	148.00
/api/statistics/neighbourhoods	152.29	112.00

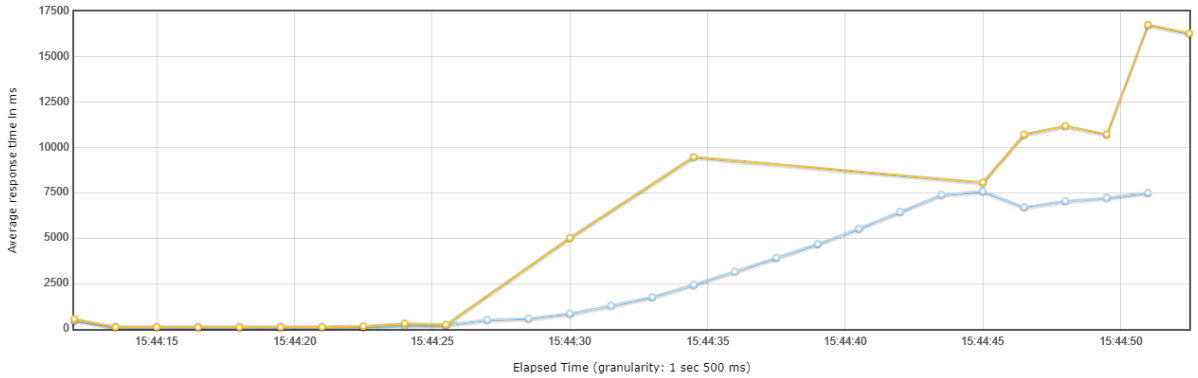
Tabel 9 Response times in milliseconden

Zoals te zien is in figuur 3 wordt de operational ceiling niet bereikt en kijkend naar de response time is deze gemiddeld meer dan 200% verbeterd. Ook krijg ik niet meer de Redis connection timeout en slagen hierdoor alle requests. Hierdoor heb ik moeten kijken naar nieuwe instellingen om te testen zodat de applicatie weer zijn operational ceiling bereikt.

Setting	Value
Target rate	20
Ramp up time	30
Ramp up step count	10
Hold target rate	5

Tabel 5

Door de target rate naar 20 te zetten wordt na ongeveer 20 seconden weer de operational ceiling bereikt en kunnen we weer verder testen. De grafieken en average response tijden zien er nu zo uit:



Figuur 7 De response times over time

Route	Gemiddelde response time (ms)	Median response time (ms)
/api/listings?geojson=true	4065.01	4885.50
/api/statistics/neighbourhoods	9657.92	9440.50

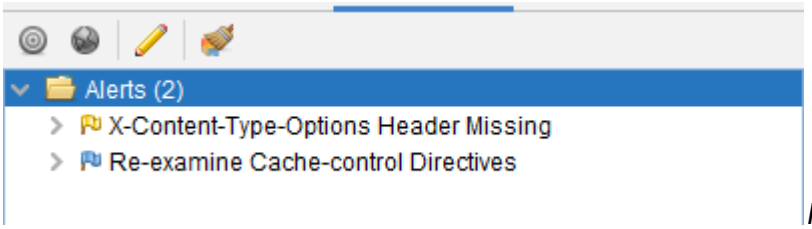
Tabel 10 Response times in milliseconden

5 Security

In dit hoofdstuk worden de security issues die gevonden zijn met behulp van OWASP ZAP uitgelegd.

5.1 Server

Als eerste heb ik de API gescand met OWASP ZAP, hieruit kwamen twee dingen naar voren: X-Content-Type-Options Header Missing en Re-examine Cache-control Directives.



Figuur 8

Bij X-Content-Type-Options Header Missing kunnen oudere versies van Internet Explorer en Chrome MIME-sniffing uitvoeren op de response body als je deze header niet zet naar ‘no-sniff’ (MDN contributors, 2022a). Dit kan resulteren in dat de response body wordt weergegeven als het verkeerde type content en dat willen we natuurlijk niet.

De cache-control header was niet gezet dus kan de browser zelf ervoor kiezen om dingen te caching. Dit zou voor bestanden zoals css bestanden of javascript bestanden niet erg zijn, maar voor dingen waarvoor je authenticated moet zijn is dat niet zo handig.

Ik heb beide problemen opgelost door middleware in mijn Program.cs bestand van de server te zetten met de volgende code:

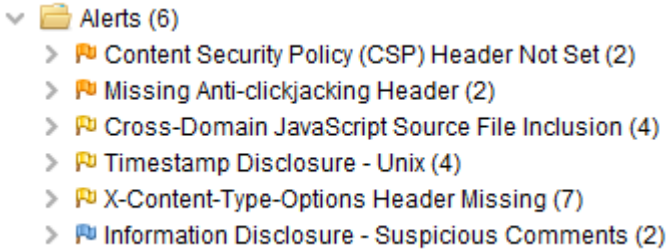
```
app.Use(async (context, next) =>
{
    context.Response.Headers.Add"X-Content-Type-Options", "nosniff");
    context.Response.Headers.Add("Cache-Control", "no-cache, no-store, must-revalidate");

    await next();
};
```

Figuur 9

5.2 Cliënt

Ook heb ik de cliënt gescand met OWASP ZAP, hieruit kwamen 5 verschillende issues naar voren. Twee hiervan, timestamp disclosure en information disclosure waren een false positive. De timestamp was namelijk het id van een listing en de comments zitten in de webassembly packages die mee worden geleverd met Blazor.



Figuur 10

Content Security Policy (CSP) Header Not Set was de eerste issue die opkwam. De CSP helpt om bepaalde soorten aanvallen te detecteren zoals Cross Site Scripting en data injection (MDN contributors, 2022). CSP heeft een aantal standaard headers waarmee je kan bepalen wat voor bronnen op een website geladen mogen worden. Ik heb de CSP ingesteld voor alleen de origins die benodigd zijn voor goede werking van de site.

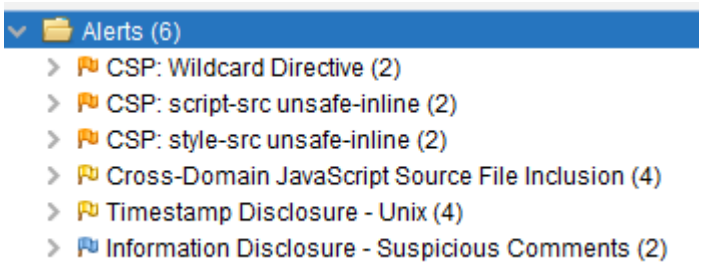
De tweede issue is Missing Anti-clickjacking Header oftewel de X-Frame-Options Header is niet gezet is. Deze header zorgt ervoor dat de browser weet wat wel en wat niet toegestaan is om te laten zien in een X-Frame (Jackson, 2019). Ik heb deze op SAMEORIGIN gezet.

```
app.Use(async (context, next) =>
{
    context.Response.Headers.Add("X-Frame-Options", "SAMEORIGIN");
    context.Response.Headers.Add("Content-Security-Policy",
        "default-src 'self' 'unsafe-inline' 'unsafe-eval' https://api.mapbox.com https://events.mapbox.com "
        + "https://login.microsoftonline.com; script-src 'unsafe-inline' 'unsafe-eval' 'self' https://api.mapbox.com
"
        + "https://code.jquery.com https://cdn.jsdelivr.net blob: data:; style-src 'unsafe-inline' 'self' " +
"https://api.mapbox.com; img-src 'self' blob: data:;");

    await next();
});
```

Figuur 11

De Cross-Domain Javascript Source File Inclusion wordt al deels opgelost door CSP, want die zorgt dat de javascript alleen van de opgegeven urls geladen kan worden. De laatste issue is die één van twee die ook bij server kwam. Dit kwam omdat deze files niet opnieuw gebuild waren en nadat ik de bin en obj folder verwijderd had en de scan opnieuw deed kwamen deze niet meer voor.



Figuur 12

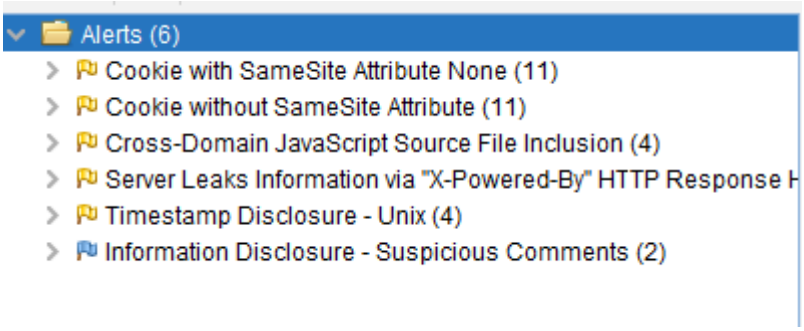
Zoals hierboven gezegd heb ik de scan nog een keer gedaan en nu kwamen er 3 nieuwe dingen uit over de CSP. De eerste is dat “frame-ancestors” en “form-actions”. Aangezien ik deze niet gebruik heb ik die op “none” gezet.

De andere issue is dat er bij script-src en style-src “unsafe-inline” gebruikt wordt om inline javascript en css te laden. Dit is ongewenst want zorgt eigenlijk ervoor dat de CSP gemakkelijk kunnen worden omzeild.

Omdat Blazor WASM gebruik maakt van inline scripts heb ik een hash toegevoegd en “unsafe-eval” toegevoegd. Dit volgens de documentatie van Microsoft.Ook heb ik de setDotnetHelper voor de JSinterop verzet naar de mapbox.js bestand zodat deze geen inline script meer heeft.

Voor de inline css heb ik deze allemaal vervangen door classes in de app.css bestand. Hierdoor kon ‘unsafe-inline’ verwijderd worden van style-src. Na het veranderen van deze dingen zijn deze twee issues ook opgelost.

5.3 Azure



Figuur 13

Tijdens het deployen op Azure worden er een aantal headers en dingen standaard toegevoegd die niet altijd gewenst zijn. Daarom heb ik toen ik de site gedeployed had op Azure nog een keer gescand met de OWASP scanner. Hieruit kwamen drie nieuwe issues.

Als eerste kwamen er twee issues over cookies: cookie with SameSite Attribute None en Cookie without SameSite Attribute. Dit zorgt ervoor dat de cookie technisch gezien kan worden aangemaakt door een cross-site request waar jij geen controle over hebt. Dit is gemakkelijk op te lossen door de SameSite attribute naar strict of lax te zetten.

De volgende issue is dat azure een X-Powered-By header toevoegt en dit kan leiden dat hackers gemakkelijk kunnen zien wat je gebruikt en of hiervoor ook een vulnerability beschikbaar voor is. Dit kan je oplossen door de header weg te halen.

Deze issues heb ik opgelost door in de web.config in de server de volgende code toe te voegen:

Figuur 14

Voor de cookies heb ik ook de enigste cookie die gezet werd, “arraffinity”, uitgezet. Deze cookie wordt gezet door Azure bij de app service platform voor backwards compatibility, maar aangezien ik dat niet nodig heb kan ik deze gewoon uitzetten.

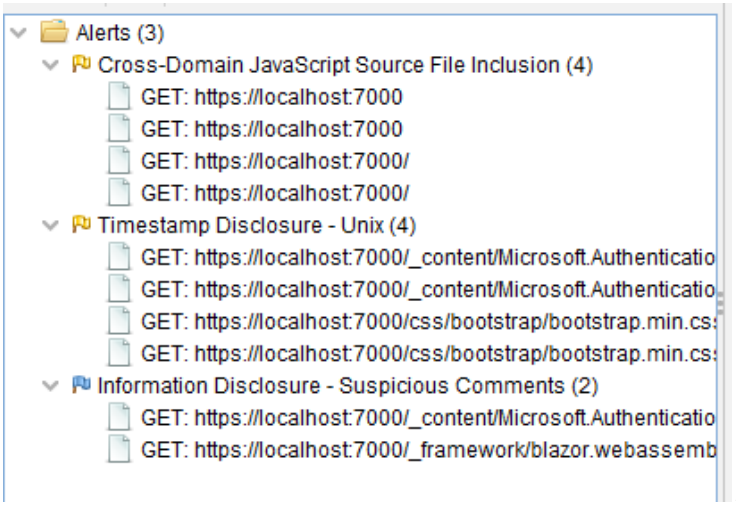
5.3 Conclusie

```
app.Use(async (context, next) =>
{
    context.Response.Headers.Add("X-Content-Type-Options", "nosniff");
    context.Response.Headers.Add("Cache-Control", "no-cache, no-store, must-revalidate");
    context.Response.Headers.Add("X-Frame-Options", "SAMEORIGIN");
    context.Response.Headers.Add("Content-Security-Policy",
        "default-src 'self' 'unsafe-inline' 'unsafe-eval' https://api.mapbox.com https://events.mapbox.com
https://login.microsoftonline.com; " + "script-src 'self' 'sha256-v8v3RKRPmN4odZ1CWM5gw80QKPCCWMcpNeOmimNL2AA='
'unsafe-eval' https://api.mapbox.com https://code.jquery.com https://cdn.jsdelivr.net blob: data:; "
        + "style-src 'self' https://api.mapbox.com; "
        + "img-src 'self' blob: data:; frame-ancestors 'none'; form-action 'none';" );

    await next();
});
```

Figuur 15

Van de negen issues die OWASP ZAP aan heeft gegeven heb ik er drie opgelost en waren er twee false positives. In figuur 15 staan alle headers die ik heb toegevoegd om de issues op te lossen. De ene die overblijft, het laden van externe javascript bestanden is grotendeels ook opgevangen door de CSP policy die alleen van specifieke URLs toestaat. Zie ook figuur 16 voor de overgebleven issues.



Figuur 16

Literatuurlijst

1. Jackson, B. (2019, 16 augustus). X-Frame-Options - How to Combat Clickjacking. KeyCDN. Geraadpleegd op 1 juni 2022, van https://www.keycdn.com/blog/x-frame-options
2. MDN contributors. (2022a, april 22). X-Content-Type-Options - HTTP | MDN. MDN Web Docs. Geraadpleegd op 1 juni 2022, van https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options
3. MDN contributors. (2022b, mei 28). Content Security Policy (CSP) - HTTP | MDN. MDN Web Hub. Geraadpleegd op 1 juni 2022, van https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP
4. Microsoft Docs Contributors. (2022, 29 januari). Clustered and nonclustered indexes described - SQL Server. Microsoft Docs. Geraadpleegd op 7 juni 2022, van https://docs.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-ver16