

Inside AirBNB Amsterdam

Ontwerp Document



Datagedreven webapplicatie getest voor high performance en security

AUTEUR	Iliass El Kaddouri // 658025
COURSE	NOTS WAPP
DOCENTEN	M. Verheij
DATUM	03-06-2022
VERSIE	1.0

Inhoudsopgave

1. Inleiding	3
2 Functioneel ontwerp	4
2.1 Use cases brief format	4
2.1.1 Registreren en inloggen account	4
2.1.2 Inloggen account	4
2.1.3 Bekijk overzicht AirBNB listings	4
2.1.4 Filter listings	4
2.1.5 Bekijk listing details	4
2.1.6 Bekijk overzicht statistieken	4
2.2 use case diagram	5
3 Technisch ontwerp	6
3.1 Architectuur	6
3.2 Overzicht frameworks	8
3.3 Overzicht packages	8
3.3.1 Client	9
3.3.2 API	10
3.3.3 Tooling	11
4 Performance	12
4.1 AsNoTracking	14
4.1.1 Implementatie	14
4.1.2 Resultaten	15
4.2 DTO's	16
4.2.1 Implementatie	16
4.2.2 Resultaten	17
4.3 Async	18
4.3.1 Implementatie	19
4.3.2 Resultaten	20
4.4 Caching	21
4.4.1 Implementatie	21
4.4.2 Resultaten	22
4.5 Conclusie	24
5 Security	25
5.1 OWASP ZAP	25
5.1.1 Content Security Policy (CSP) Header Not Set	25
5.1.2 Missing Anti-clickjacking Header	27
5.1.3 Timestamp Disclosure - Unix	29
5.1.4 X-Content-Type-Options Header Missing	29
5.1.5 Information Disclosure - Suspicious Comments	29
5.2 Eigen maatregelen	30
5.2.1 Endpoint protection	30

5.3 Conclusie	32
Bijlagen	33
Performance tests	33
Baseline	33
Optimalisatie 4.1 AsNoTracking()	34
Optimalisatie 4.2 DTO's	35
Optimalisatie 4.3 Async	36
Optimalisatie 4.4 Redis cache	37
Optimalisatie 4.4 Redis cache synchroon	38
Literatuurlijst	39

1. Inleiding

Het management van InsideAirBNB heeft behoefte aan inzicht in het gebruik van AirBNB locaties in Amsterdam. De wensen zijn overzichten van verschillende statistieken zoals onder andere het gemiddelde aantal overnachtingen per maand, gemiddelde opbrengsten per buurt en de gemiddelde reviews per buurt. De applicatie zal door medewerkers in gebruik worden genomen. De wens is dat de look-and-feel van de applicatie hetzelfde is als voor derde gebruikers.

2 Functioneel ontwerp

2.1 Use cases brief format

2.1.1 Registreren en inloggen account

Een medewerker registreert een account. Elk account heeft een toegekende rol, zoals administrator. Op basis van deze rollen krijgt een medewerker verschillende privileges.

2.1.2 Inloggen account

Een medewerker kan inloggen met zijn account om gebruik te maken van de toegekende privileges.

2.1.3 Bekijk overzicht AirBNB listings

Een gebruiker of medewerker kan een overzicht met alle AirBNB listings in Amsterdam bekijken. Dit overzicht wordt visueel weergegeven in een kaart.

2.1.4 Filter listings

De listings in het overzicht, genoemd in 2.1.3, kunnen gefilterd worden op prijs, buurt en aantal reviews.

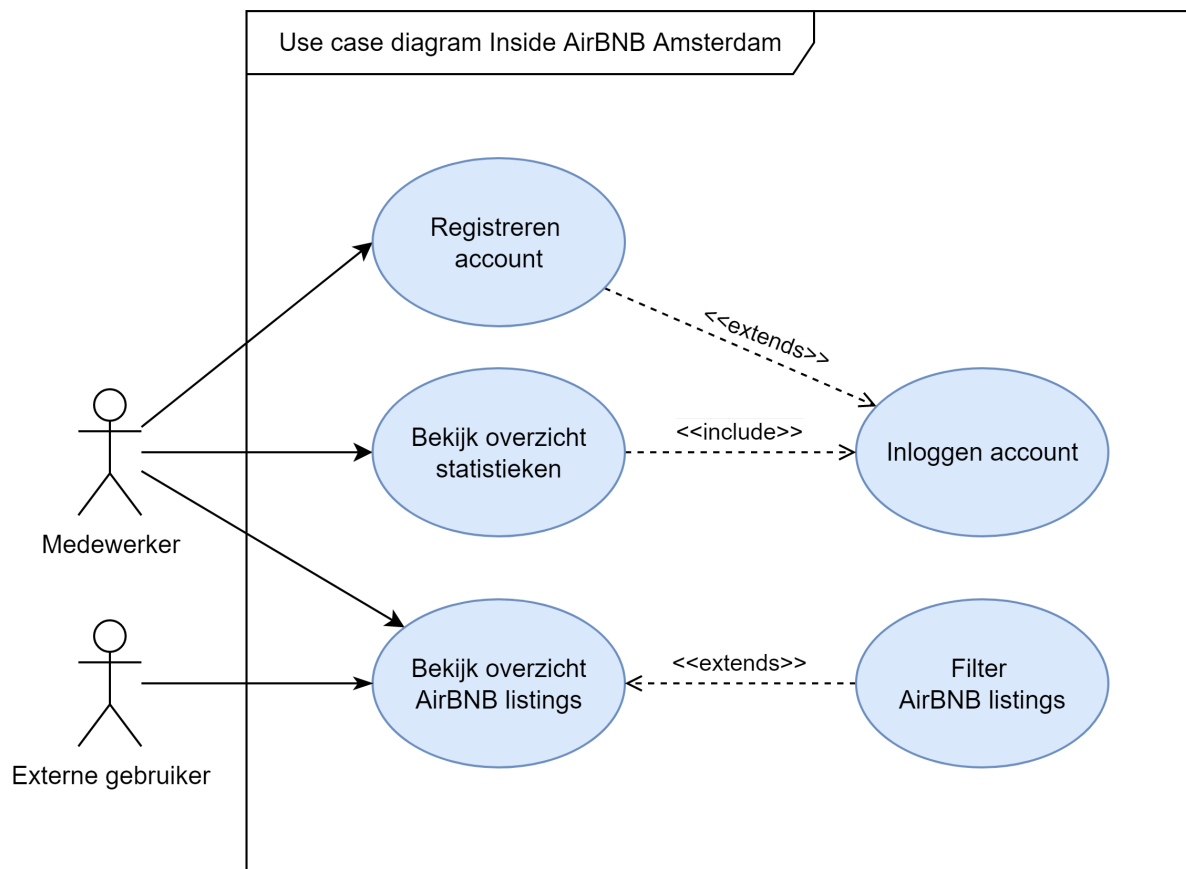
2.1.5 Bekijk listing details

Er is een overzicht van details voor een geselecteerde listing. Een gebruiker kiest een listing waarna de details van deze listing worden weergegeven.

2.1.6 Bekijk overzicht statistieken

Een medewerker kan een overzicht opvragen van statistieken. Het overzicht bevat relevante statistieken van alle listings, zoals het aantal listings per buurt, gemiddelde opbrengst per buurt, gemiddelde beschikbaarheid per buurt en het gemiddelde aantal bedden per buurt.

2.2 use case diagram



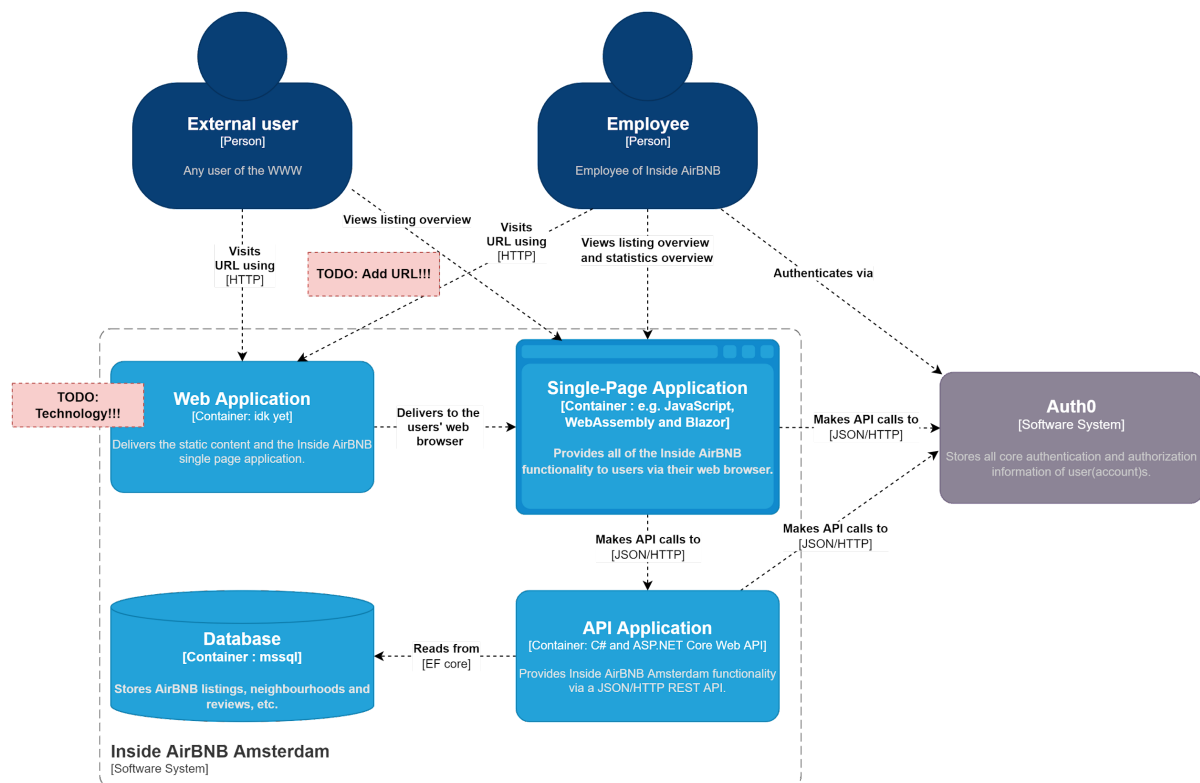
Figuur 1: Use case diagram Inside AirBNB Amsterdam.

3 Technisch ontwerp

In dit hoofdstuk wordt er ingegaan over de technische ontwerp keuzes voor dit project.

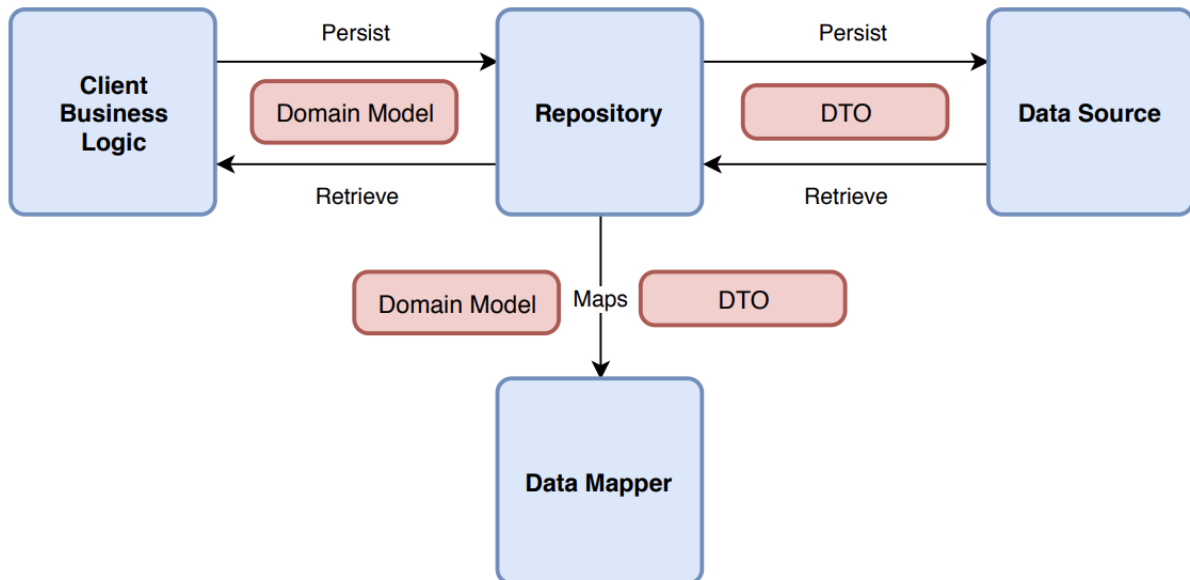
3.1 Architectuur

Om een high level overview te laten zien van de applicatie is het volgende C4 Container diagram opgesteld.



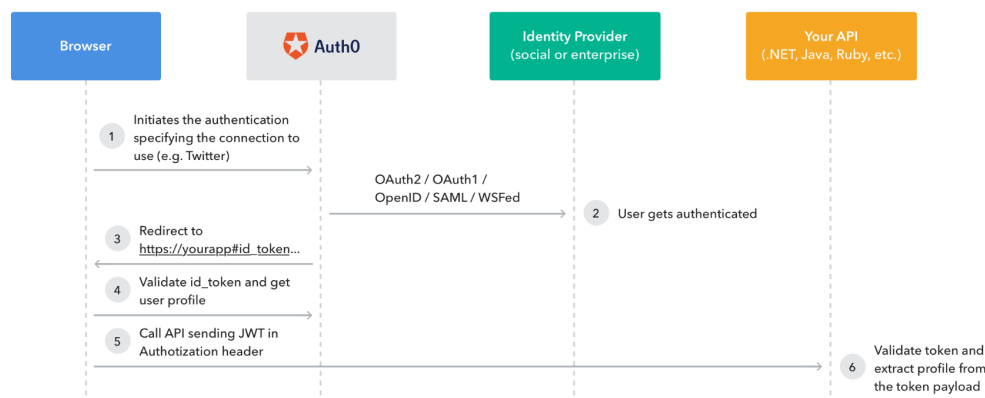
Figuur 2: C4 Container diagram voor Inside AirBNB Amsterdam. Dit diagram geeft een weergave van de gebruikte architectuur.

Verder is er gebruik gemaakt van het Repository pattern¹ om de API in een overzichtelijke en nette manier in te richten. De repository objecten in dit project heten services. Zo blijft de code gescheiden en leesbaar en zijn de services verantwoordelijk voor het aanspreken van de SQL database.



Figuur 3: Overzicht repository pattern. Bron: Dennis Brandi, zie voetnoot 1.

Om authenticatie en autorisatie met rollen mogelijk te maken is er gebruik gemaakt van Auth0². Auth0 neemt al het werk betreft authenticatie en autorisatie uit handen. Hierdoor hoeft een ontwikkelaar zich minder zorgen te maken over de veiligheid en complexiteit van dit proces. Tevens biedt het een simpele oplossing om endpoints te beveiligen met rollen. Hieronder staat een overzicht van de authenticatie flow die door Auth0 gebruikt wordt.



Figuur 4: Overzicht Token based authenticatie en autorisatie flow. Bron: auth0.com³

¹ <https://proandroiddev.com/the-real-repository-pattern-in-android-efba8662b754>

² <https://auth0.com/docs/>

³ <https://auth0.com/learn/how-auth0-uses-identity-industry-standards/>

3.2 Overzicht frameworks

1. Client: ASP.NET Blazor (WASM)⁴
2. API: ASP.NET Web API⁵
3. Database: Microsoft/Azure SQL Server⁶
4. Cache (In-memory database): Redis⁷

3.3 Overzicht packages

Alle genoemde packages zijn geïnstalleerd m.b.v. NuGet. Alleen achteraf geïnstalleerde packages zijn beschreven. De standaard packages die geïnstalleerd worden tijdens het scaffolding (met het `dotnet new` commando) van de apps zijn niet beschreven.

⁴ <https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-6.0>

⁵ <https://dotnet.microsoft.com/en-us/apps/aspnet/apis>

⁶ <https://docs.microsoft.com/en-us/sql/?view=sql-server-ver16>

⁷ <https://redis.com/solutions/use-cases/caching/>

3.3.1 Client

Package	Versie	Comments
ChartJs Blazor ⁸	2.0.2	<ul style="list-style-type: none"> • Geeft de mogelijkheid om ChartJs⁹ te gebruiken m.b.v. C# in de ASP.NET omgeving.
Mapbox GL JS ¹⁰	2.8.2	<ul style="list-style-type: none"> • Geeft de mogelijkheid om Mapbox te gebruiken in een webapp. • Is niet speciaal voor Blazor gemaakt. Hierdoor moest er gebruik worden gemaakt van IJRuntime¹¹ en DotNetObjectReference¹² om javascript functies vanuit .NET aan te roepen en vice versa.
Microsoft.AspNetCore.Components.WebAssembly.Authentication ¹³	6.0.5	<ul style="list-style-type: none"> • Deze package is nodig om Auth0 Token based authenticatie mogelijk te maken. • Het gebruik ervan staat beschreven in een blog post van Auth0¹⁴.
Microsoft.Extensions.Http ¹⁵	6.0.0	<ul style="list-style-type: none"> • Geeft de mogelijkheid om 'named' HTTP clients te maken en het gedrag van de client aan te passen. Dit is nodig om de Autorisatie Token automatisch aan een request te plakken. • Het gebruik ervan staat beschreven in deze blog post van Auth0¹⁶.

Tabel 1: Overzicht packages gebruikt in de client.

⁸ <https://github.com/mariusmuntean/ChartJs.Blazor>

⁹ <https://www.chartjs.org/>

¹⁰ <https://docs.mapbox.com/mapbox-gl-js/guides/>

¹¹

<https://docs.microsoft.com/en-us/aspnet/core/blazor/javascript-interoperability/call-javascript-from-dotnet?view=aspnetcore-6.0>

¹²

<https://docs.microsoft.com/en-us/aspnet/core/blazor/javascript-interoperability/call-dotnet-from-javascript?view=aspnetcore-6.0>

¹³

<https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.components.webassembly.authentication?view=aspnetcore-6.0>

¹⁴ <https://auth0.com/blog/securing-blazor-webassembly-apps/>

¹⁵ <https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.http?view=dotnet-plat-ext-6.0>

¹⁶ <https://auth0.com/blog/securing-blazor-webassembly-apps/>

3.3.2 API

Package	Versie	Comments
Microsoft.EntityFrameworkCore.Design ¹⁷	6.0.4	<ul style="list-style-type: none">• Nodig voor het reverse engineeren van de Database. Hierbij worden er automatisch Models gescaffold op basis van de tabellen.• Het gebruik ervan is beschreven in een post van Microsoft¹⁸.
Microsoft.EntityFrameworkCore.SqlServer ¹⁹	6.0.4	<ul style="list-style-type: none">• Maakt het mogelijk om te verbinden met een MSSQL Server.
Microsoft.AspNetCore.Authentication.JwtBearer ²⁰	6.0.5	<ul style="list-style-type: none">• Maakt het mogelijk om endpoints te beveiligen d.m.v. Token based authenticatie en autorisatie van Auth0.• Het gebruik ervan is beschreven in twee blog posts van Auth0^{21 22}.
Microsoft.Extensions.Caching.StackExchangeRedis ²³	6.0.5	<ul style="list-style-type: none">• Maakt het mogelijk om gebruik te maken van Redis cache.

Tabel 2: Overzicht packages gebruikt in de API.

¹⁷ <https://docs.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.design?view=efcore-6.0>

¹⁸ <https://docs.microsoft.com/en-us/ef/core/managing-schemas/scaffolding?tabs=dotnet-core-cli>

¹⁹ <https://docs.microsoft.com/en-us/ef/core/providers/sql-server/?tabs=dotnet-core-cli>

²⁰ <https://auth0.com/blog/securing-blazor-webassembly-apps/>

²¹ <https://auth0.com/blog/role-based-authorization-for-aspnet-webapi/>

²² <https://auth0.com/blog/securing-blazor-webassembly-apps/>

²³

<https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.caching.stackexchangeredis.redisocache?view=dotnet-plat-ext-3.1>

3.3.3 Tooling

Package	Versie	Comments
Entity Framework Core tools (.NET Core CLI) ²⁴	6.0.4	<ul style="list-style-type: none">• Deze package is global geïnstalleerd en bevat CLI tools voor EF Core.• Nodig voor het scaffolding van Modellen zoals beschreven in 3.2.2.
Docker ²⁵ Docker Compose ²⁶	20.10.16 2.6.0	<ul style="list-style-type: none">• Maakt het mogelijk om lokaal eenvoudig een Azure MSSQL database en Redis Cache te draaien.• Zie docker-compose.yml²⁷.

Tabel 3: Overzicht van CLI tools.

²⁴ <https://docs.microsoft.com/en-us/ef/core/cli/dotnet>

²⁵ <https://docs.docker.com/engine/>

²⁶ <https://docs.docker.com/compose/>

²⁷ <https://github.com/ilivss/inside-airbnb-amsterdam/blob/main/docker-compose.yml>

4 Performance

In dit hoofdstuk wordt er ingegaan op de verschillende prestatie optimalisaties die zijn uitgevoerd in deze applicatie. Als eerst wordt er ingegaan op prestatie verbetering door middel van code veranderingen, later wordt er uitleg gegeven over de verbeteringen door middel van memory caching.

Om de optimalisaties aan te tonen wordt er gebruik gemaakt van Locust²⁸. Om de werkelijkheid na te bootsen is er gebruik gemaakt van een custom Locust script²⁹ (zie figuur 5) welke de load stapsgewijs verzaard. Als eerst is er een nulmeting uitgevoerd waarvan de resultaten in onderstaande figuren 6, 7, 8 en 9 te zien zijn.

```
// locustfile.py
import math
from locust import HttpUser, TaskSet, task, constant
from locust import LoadTestShape

class UserTasks(TaskSet):
    def on_start(self):
        self.client.verify = False

    @task
    def get_listings(self):
        self.client.get("/listing")
        self.client.get("/neighbourhood")

class WebsiteUser(HttpUser):
    wait_time = constant(0.5)
    tasks = [UserTasks]

class StepLoadShape(LoadTestShape):
    # step_time -- Time between steps
    # step_load -- User increase amount at each step
    # spawn_rate -- Users to stop/start per second at every step
    # time_limit -- Time limit in seconds

    step_time = 15
    step_load = 10
    spawn_rate = 10
    time_limit = 150

    def tick(self):
```

²⁸ <https://locust.io/>

²⁹ <https://docs.locust.io/en/stable/writing-a-locustfile.html>

```

run_time = self.get_run_time()

if run_time > self.time_limit:
    return None

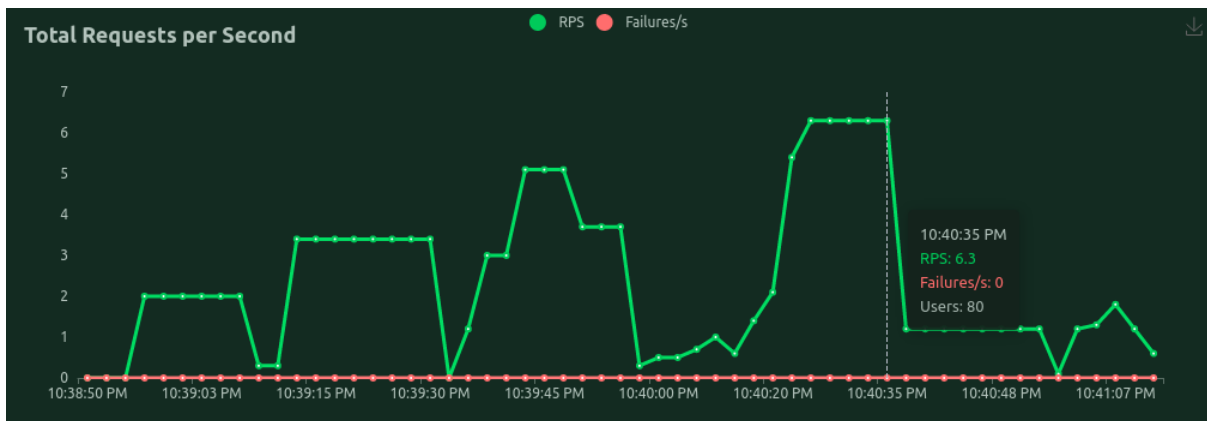
current_step = math.floor(run_time / self.step_time) + 1
return (current_step * self.step_load, self.spawn_rate)

```

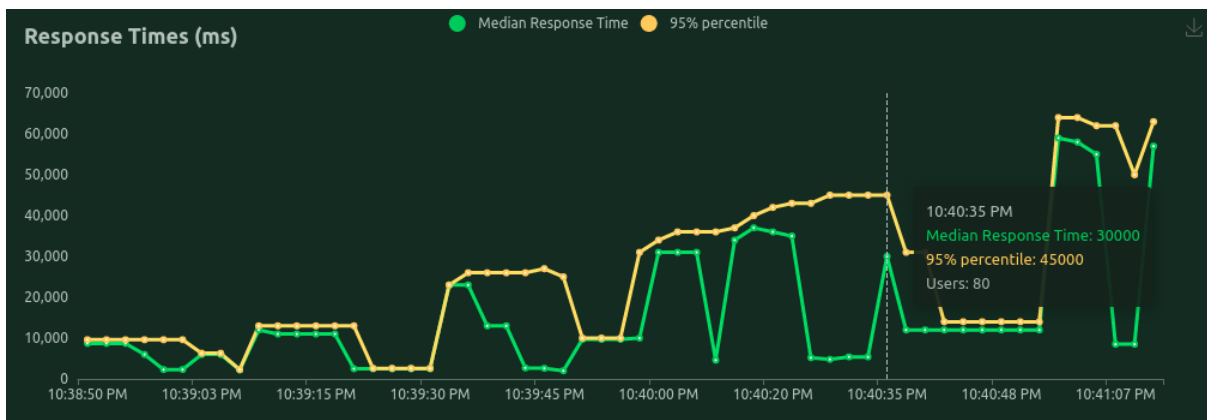
Figuur 5: Locust stress testing script. Zie comments voor uitleg van waarden.

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/listing	143	0	26000	50000	63000	28459	5984	64038	107684529	0.1	0
GET	/neighbourhood	141	0	2600	9700	14000	3691	83	13679	2391	0.5	0
Aggregated		284	0	9400	41000	62000	16162	83	64038	54222622	0.6	0

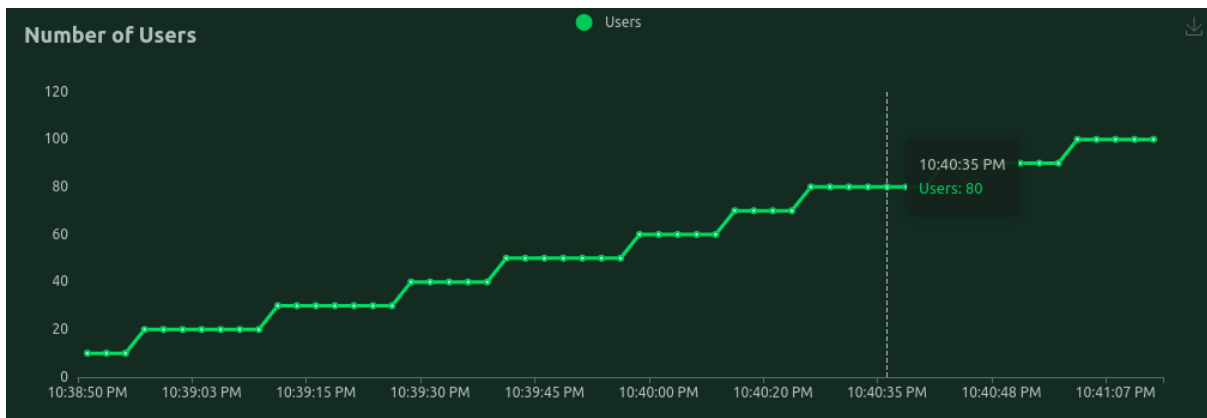
Figuur 6: Tabel met overzicht van de nulmeting.



Figuur 7: Nulmeting van het aantal requests per seconde.



Figuur 8: Nulmeting van de responstijden.



Figuur 9: Nulmeting van aantal gebruikers.

De aangesproken endpoints voor het bepalen van prestatieverbeteringen zijn:

- [GET] /listing
- [GET] /neighbourhood

4.1 AsNoTracking

Microsofts' Entity Framework Core biedt een gemakkelijke optie voor het verbeteren van prestaties. De optie *AsNoTracking()*³⁰ geeft aan dat EF Core de resultaten van een query niet moet cachen. Wat resulteert in een lagere overhead, omdat EF Core geen aanvullende verwerking en opslag uitvoert van de geretourneerde waarden van een query.

4.1.1 Implementatie

De implementatie van deze optimalisatie is minimaal. Een voorbeeld is in onderstaand figuur weergegeven.

```
// ListingController.cs
public IEnumerable<Listing> Get(int? minPrice, int? maxPrice, string?
    neighbourhood, int? minNrOfReviews, int? maxNrOfReviews)
{
    return _context.Listings
+     .AsNoTracking()
        .Where(l =>
            (minPrice == null || l.Price > minPrice) &&
            (maxPrice == null || l.Price < maxPrice) &&
            (neighbourhood == null ||
                l.Neighbourhood.ToLower().Equals(neighbourhood.ToLower())) &&
            (minNrOfReviews == null || l.NumberOfReviews > minNrOfReviews) &&
            (maxNrOfReviews == null || l.NumberOfReviews < maxNrOfReviews)
        )
}
```

³⁰

<https://docs.microsoft.com/en-us/dotnet/api/system.data.entity.dbextensions.asnotracking?view=entity-framework-5.0.0>

```

        .ToList();
    }

```

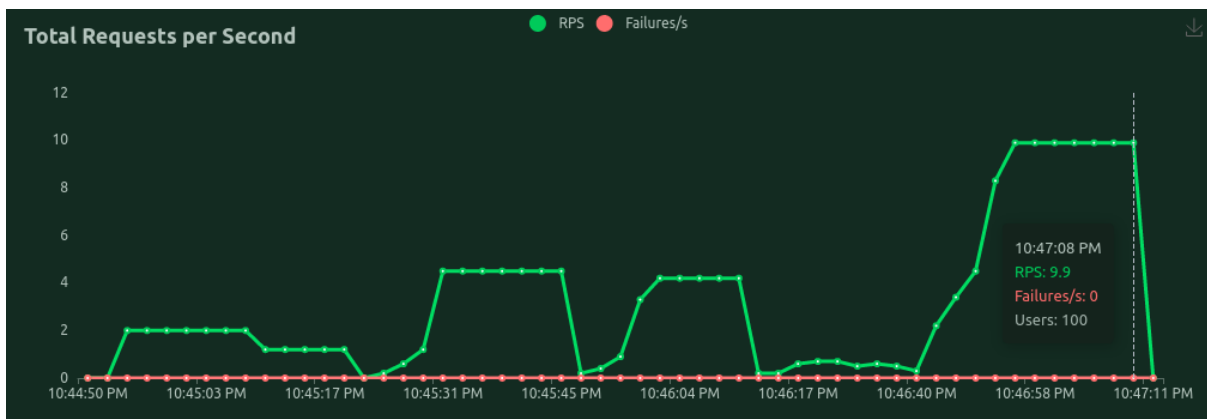
Figuur 10: Voorbeeld implementatie van optimalisatie 4.1 in ListingController.cs.

4.1.2 Resultaten

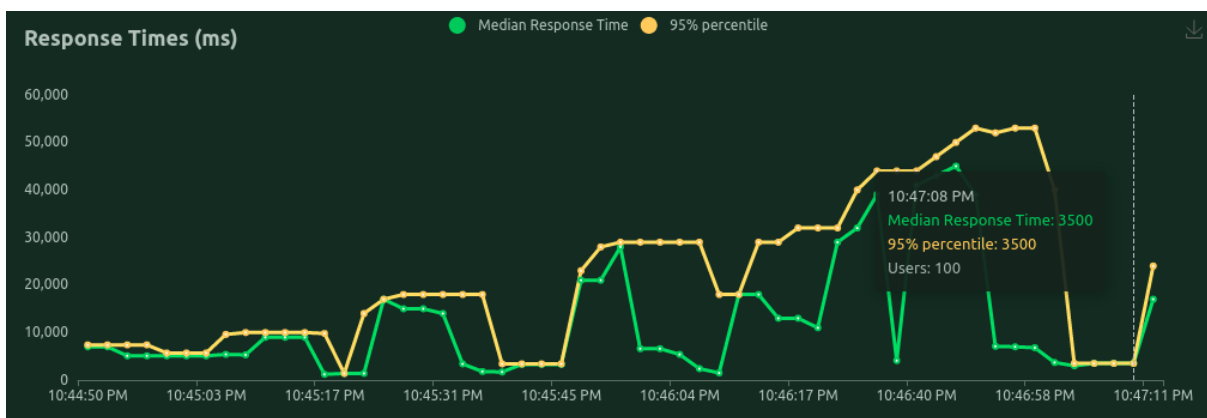
De resultaten zijn ten opzichte van de nulmeting gestegen. Zo scheelt de gemiddelde responstijd met ongeveer 600 ms. Het aantal requests per seconde en users zijn wel erg gestegen: er is een voordeel behaald van 3.6 RPS en 20 users.

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/listing	189	1	28000	47000	53000	27625	5076	52838	107114770	0	0
GET	/neighbourhood	188	0	2400	7000	17000	3454	83	17089	2391	0	0
Aggregated		377	1	7200	43000	53000	15571	83	52838	53700639	0	0

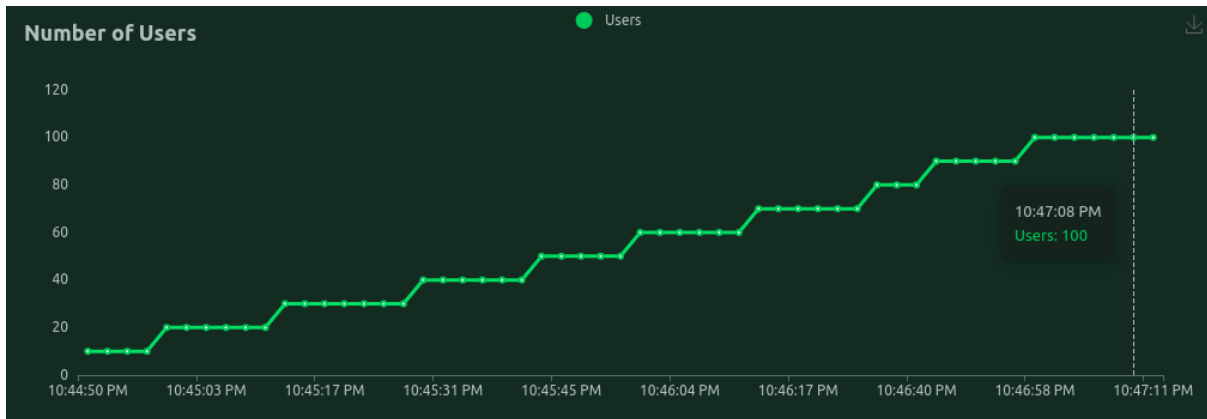
Figuur 11: Tabel met overzicht van optimalisatie 4.1.



Figuur 12: Meting van het aantal requests per seconde over tijd van optimalisatie 4.1.



Figuur 13: Meting van de responstijd over tijd van optimalisatie 4.1.



Figuur 14: Meting van het aantal users over tijd van optimalisatie 4.1.

4.2 DTO's

Een andere manier van optimalisatie is door simpelweg minder data te versturen om zo de response size te verkleinen. Zo heeft bijvoorbeeld een Listing object heeft erg veel fields waarvan er eigenlijk maar een aantal gebruikt worden voor de [GET] /listing endpoint. Hetzelfde geldt voor [GET] /neighbourhood waarvan maar een enkel veld wordt gebruikt.

4.2.1 Implementatie

```
// Listing.cs
public partial class ListingDTO
{
    public int Id { get; set; }
    public string Name { get; set; } = null!;
    public double? Latitude { get; set; }
    public double? Longitude { get; set; }
    public string? RoomType { get; set; }
}
```

Figuur 15: Voorbeeld implementatie van optimalisatie 4.2 op Listing.cs. De originele Listing model had 97 fields.

```
// ListingService.cs
M public IEnumerable<ListingDTO> Get(int? minPrice, int? maxPrice,
    string? neighbourhood, int? minNrOfReviews, int? maxNrOfReviews)
{
    return _context.Listings
        .AsNoTracking()
        .Where(l =>
            (minPrice == null || l.Price > minPrice) &&
            (maxPrice == null || l.Price < maxPrice) &&
            (neighbourhood == null ||
                l.Neighbourhood.ToLower().Equals(neighbourhood.ToLower())) &&
            (minNrOfReviews == null || l.NumberOfReviews > minNrOfReviews) &&
```

```

        (maxNrOfReviews == null || l.NumberOfReviews < maxNrOfReviews)
    )
+   .Select(l => new ListingDTO
+   {
+       Id = l.Id,
+       Name = l.Name,
+       Latitude = l.Latitude,
+       Longitude = l.Longitude,
+       RoomType = l.RoomType,
+   })
    .ToList();
}

```

Figuur 16: Voorbeeld implementatie van optimalisatie 4.2 in ListingService.cs.

```

// ListingController.cs
[HttpGet]
public IActionResult Get(int? minPrice, int? maxPrice, string? neighbourhood,
    int? minNrOfReviews, int? maxNrOfReviews)
{
M    var listingDTOs = _listingService.Get(minPrice, maxPrice, neighbourhood,
    minNrOfReviews, maxNrOfReviews);

M    return Ok(listingDTOs);
}

```

Figuur 17: Voorbeeld implementatie van optimalisatie 4.2 in ListingController.cs.

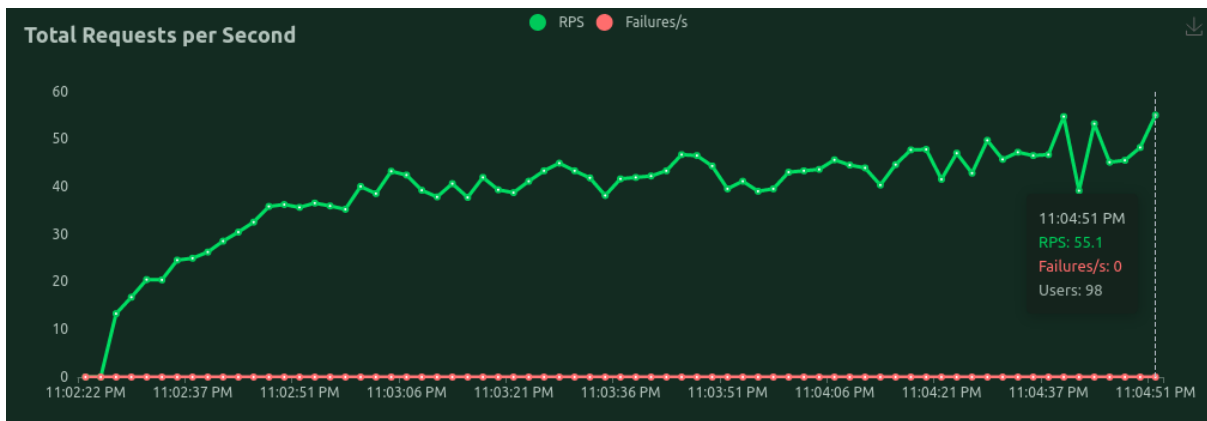
4.2.2 Resultaten

Het gebruik van DTO's heeft de performance significant laten stijgen. Zo wordt de load roof niet bereikt met de gekozen test configuratie. De responstijd heeft een voordeel behaald van 14.1 seconden ten opzichte van de laatste test. Het aantal RPS is gestegen met maar liefst 45.2 ten opzichte van de laatste meting. Het maximaal aantal users is gelijk gebleven.

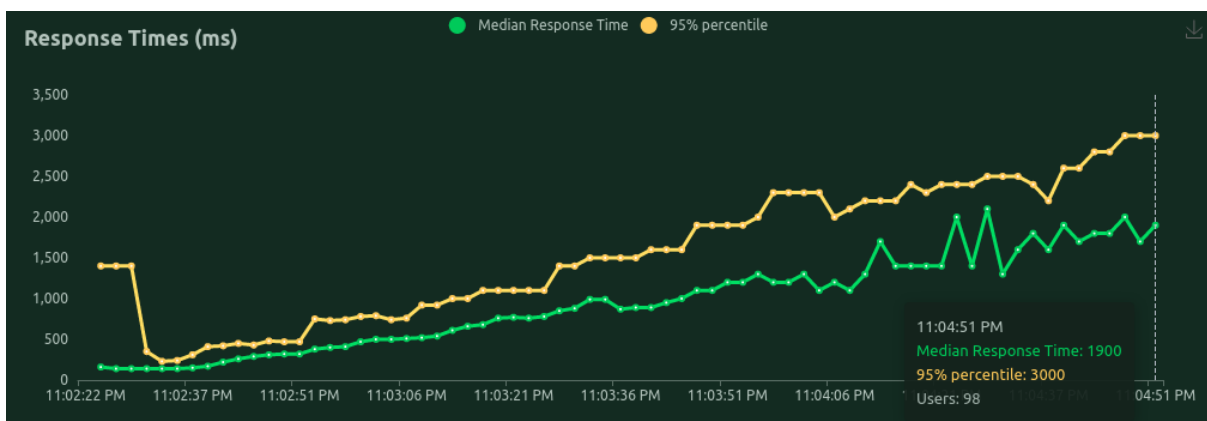
Wat nog meer interessant is, is dat de grafieken relatief stabiel zijn ten opzichte van de vorige meting, waar de grafieken vluchtig zijn.

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/listing	3115	0	1400	2300	2800	1341	44	4228	2776942	28.6	0
GET	/neighbourhood	3040	0	780	1400	2600	822	19	3009	1221	26.5	0
Aggregated		6155	0	970	2100	2800	1084	19	4228	1405993	55.1	0

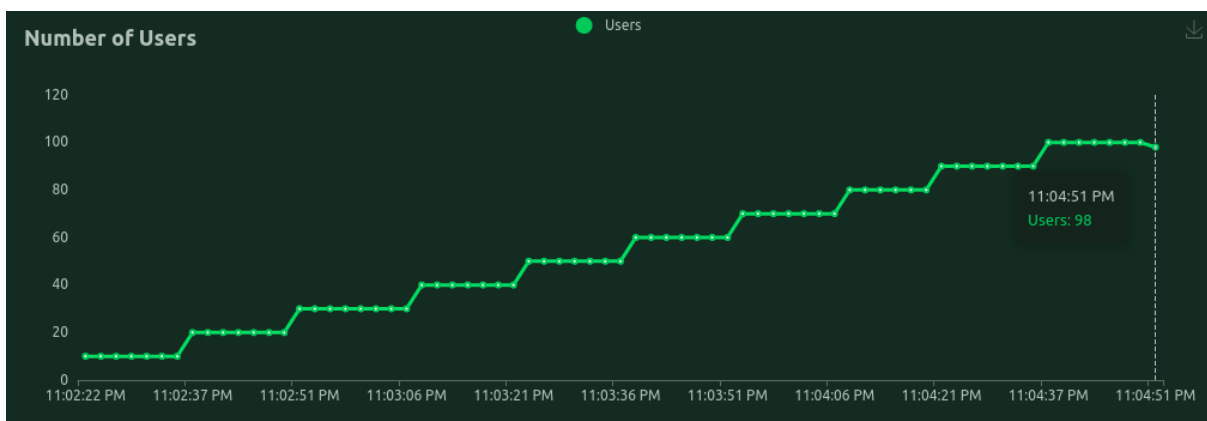
Figuur 18: Tabel met overzicht van optimalisatie 4.2.



Figuur 19: Meting van het aantal requests per seconde over tijd van optimalisatie 4.2.



Figuur 20: Meting van de responstijd over tijd van optimalisatie 4.2.



Figuur 21: Meting van het aantal users over tijd van optimalisatie 4.2.

4.3 Async

De endpoints worden nu aangeroepen door middel van synchrone functies, wat betekent dat de applicatie blijft wachten op deze synchrone methoden. De responstijden kunnen worden verlaagd door gebruik te maken van asynchrone functies.

4.3.1 Implementatie

In figuren 22 en 23 zijn code voorbeelden van veranderingen van *ListingController* en *ListingService*. Deze wijzigingen is op dezelfde op elke endpoint doorgevoerd.

```
// ListingController.cs
[HttpGet]
M public async Task<IActionResult> Get(int? minPrice, int? maxPrice, string?
    neighbourhood, int? minNrOfReviews, int? maxNrOfReviews)
{
M     var listingDTOs = await _listingService.Get(minPrice, maxPrice,
        neighbourhood, minNrOfReviews, maxNrOfReviews);

    return Ok(listingDTOs);
}
```

Figuur 22: Implementatie van optimalisatie 4.3 in ListingController.cs.

```
// ListingService.cs
M public async Task<IEnumerable<Listing>> Get(int? minPrice, int? maxPrice,
    string? neighbourhood, int? minNrOfReviews, int? maxNrOfReviews)
{
M     return await _context.Listings
        .AsNoTracking()
        .Where(l =>
            (minPrice == null || l.Price > minPrice) &&
            (maxPrice == null || l.Price < maxPrice) &&
            (neighbourhood == null ||
                l.Neighbourhood.ToLower().Equals(neighbourhood.ToLower())) &&
            (minNrOfReviews == null || l.NumberOfReviews > minNrOfReviews) &&
            (maxNrOfReviews == null || l.NumberOfReviews < maxNrOfReviews)
        )
        .Select(l => new ListingDTO
        {
            Id = l.Id,
            Name = l.Name,
            Latitude = l.Latitude,
            Longitude = l.Longitude,
            RoomType = l.RoomType,
        })
M     .ToListAsync();
}
```

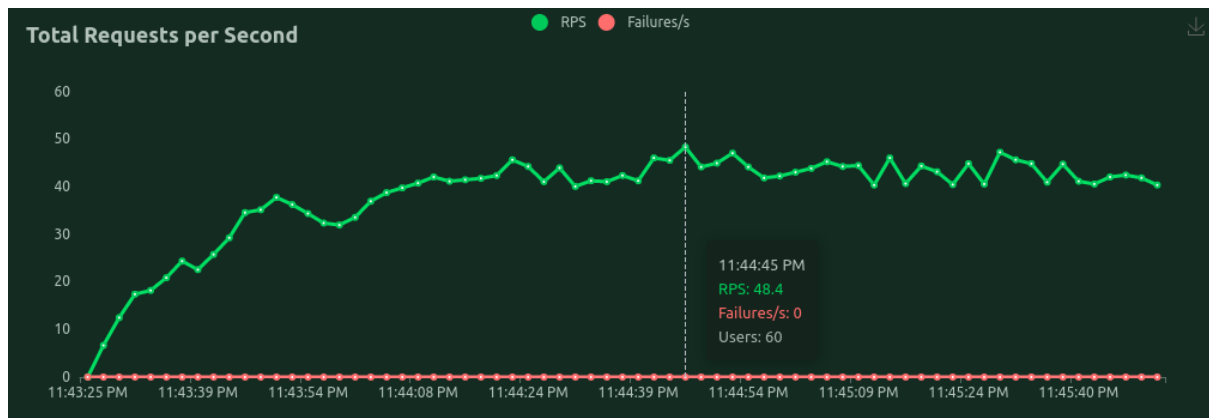
Figuur 23: Implementatie van optimalisatie 4.3 in ListingService.cs.

4.3.2 Resultaten

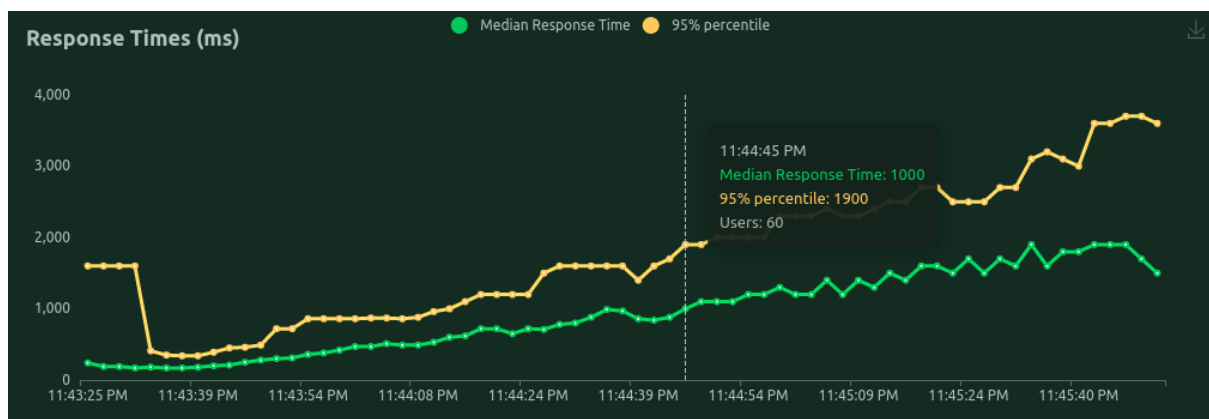
De resultaten van deze optimalisatie vallen erg tegen. De responstijden zijn vrijwel gelijk gebleven ten opzichte van de vorige meting, met uitzondering van de maximale responstijd welke een toename heeft van 500 ms. Het maximale aantal RPS is gedaald met 6.3. Wat wel opvallend is, is dat de responstijd grafiek, zie figuur 25, minder vluchtig is dan in de vorige test.

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/listing	3035	0	1100	2600	3600	1281	63	4672	2776942	22.1	0
GET	/neighbourhood	2954	0	810	1800	2700	929	19	4172	1221	21.2	0
Aggregated		5989	0	920	2300	3500	1107	19	4672	1407852	43.3	0

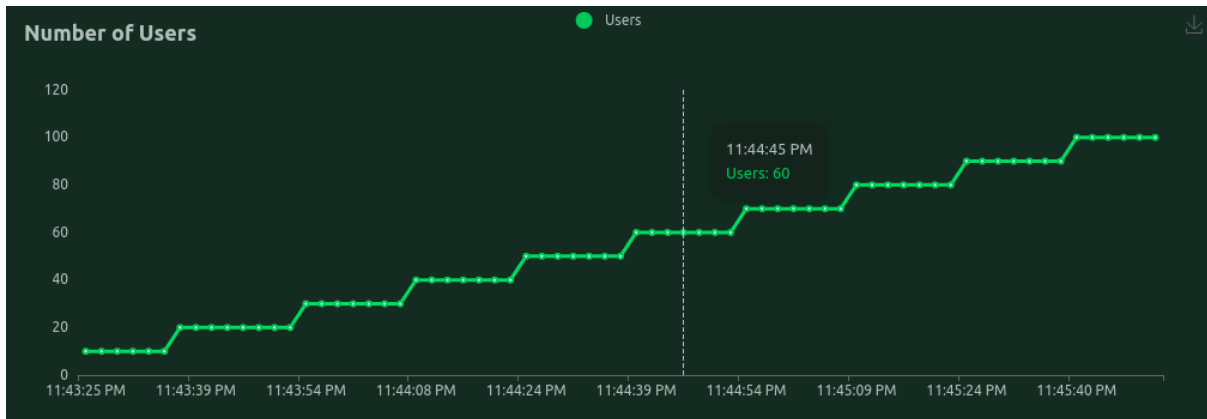
Figuur 24: Tabel met overzicht van optimalisatie 4.3.



Figuur 25: Meting van het aantal requests per seconde over tijd van optimalisatie 4.3.



Figuur 26: Meting van de responstijd over tijd van optimalisatie 4.3.



Figuur 27: Meting van het aantal users over tijd van optimalisatie 4.3.

4.4 Caching

Een traditionele SQL of NOSQL database zijn relatief traag ten opzichte van in-memory databases zoals Redis. Wanneer endpoints (bijna) niet veranderen en er genoeg server resources zijn kan het verstandig zijn bepaalde data op te slaan in een in-memory database. De reden hiervoor is omdat RAM memory simpelweg veel sneller is dan harddisk/ssd memory.

4.4.1 Implementatie

Het implementeren van een Redis cache is vrij eenvoudig. In de figuren 28 en 29 staat een voorbeeld van deze implementatie.

```
// Program.cs
builder.Services.AddStackExchangeRedisCache(options =>
{
    options.Configuration = builder.Configuration.GetConnectionString("Redis");
});
```

Figuur 28: Implementatie van optimalisatie 4.4 in Program.cs.

```
// ListingController.cs
[HttpGet]
public async Task<IActionResult> Get(int? minPrice, int? maxPrice, string?
    neighbourhood, int? minNrOfReviews, int? maxNrOfReviews)
{
    IEnumerable<Listing> listings;
    string cachedListings = await _distributedCache.GetStringAsync("listings");

    if (!string.IsNullOrEmpty(cachedListings))
    {
        listings = JsonSerializer
            .Deserialize<IEnumerable<Listing>>(cachedListings);
    }
}
```

```

else {
    listings = await _listingService.Get(minPrice, maxPrice, neighbourhood,
                                         minNrOfReviews, maxNrOfReviews);
    await _distributedCache.SetStringAsync("listings", JsonSerializer
                                         .Serialize(listings));
}

return Ok(listings);
}

```

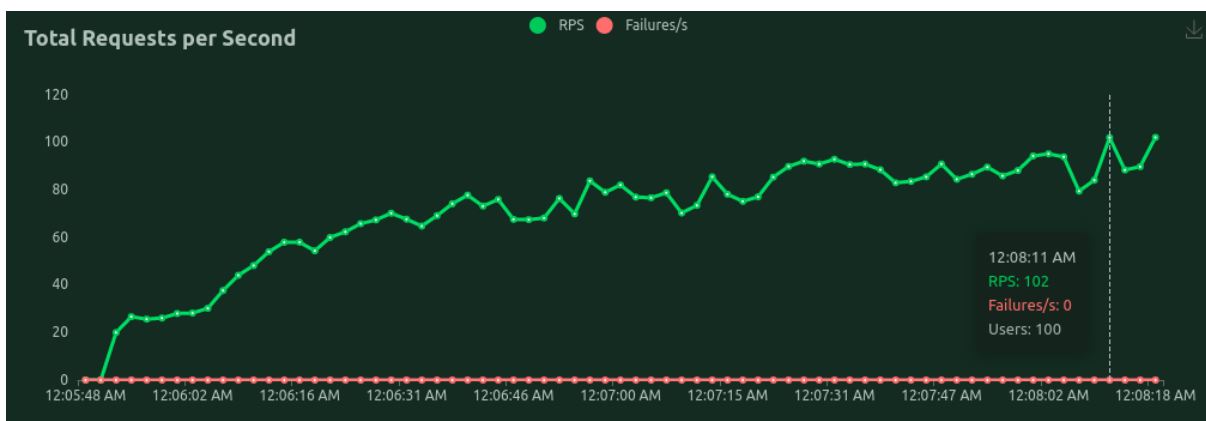
Figuur 29: Implementatie van optimalisatie 4.4 in ListingController.cs.

4.4.2 Resultaten

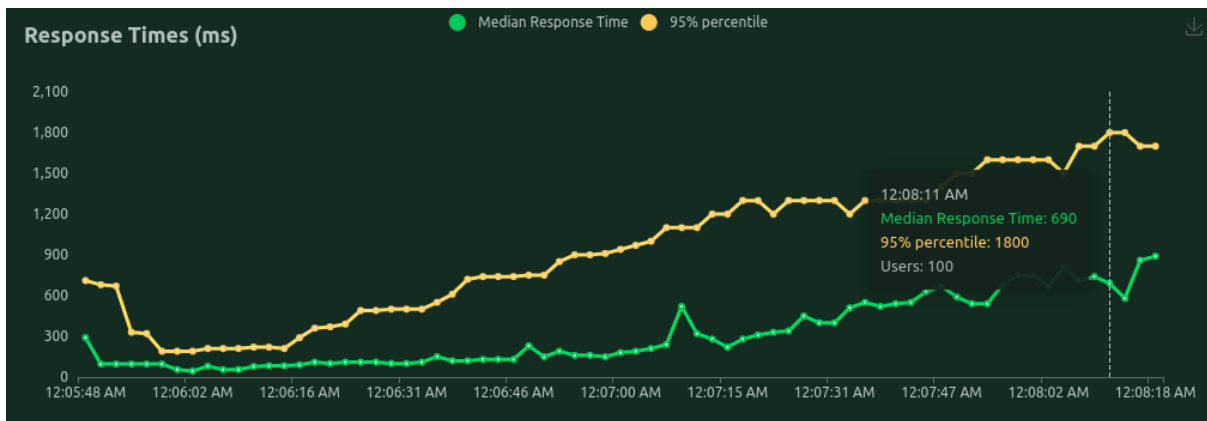
De resultaten van deze meting waren erg goed. De gemiddelde responstijd is meer dan de helft minder ten opzichte van de vorige meting. Hetzelfde geldt voor de maximale responstijd. De minimale response tijd is van 19 ms naar 1 ms gezakt. Het maximaal aantal RPS is verdubbeld.

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/listing	5700	0	750	1500	1800	779	43	1995	2776942	49.7	0
GET	/neighbourhood	5623	0	60	540	1200	176	1	1530	1221	52.4	0
	Aggregated	11323	0	230	1300	1700	479	1	1995	1398519	102.1	0

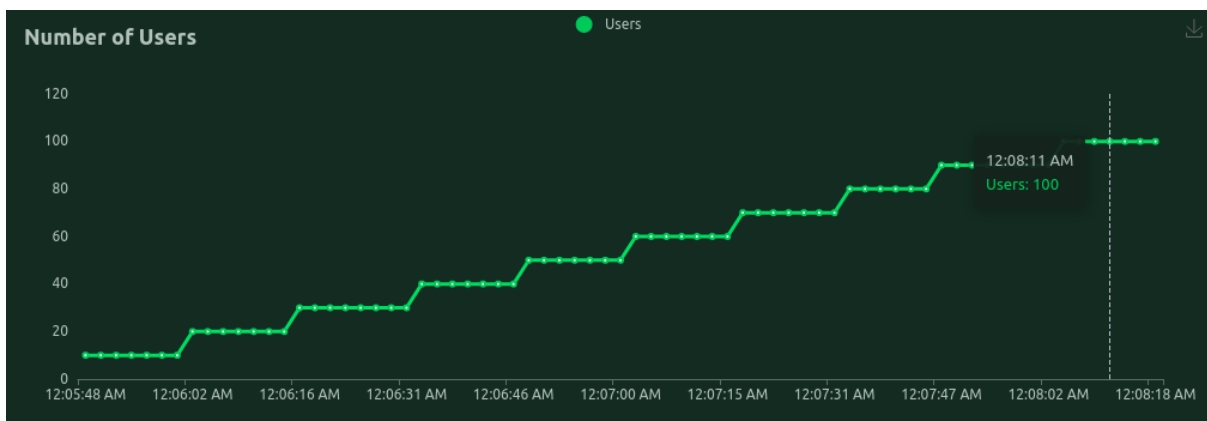
Figuur 30: Tabel met overzicht van optimalisatie 4.4.



Figuur 31: Meting van het aantal requests per seconde over tijd van optimalisatie 4.4.



Figuur 32: Meting van de responstijd over tijd van optimalisatie 4.4.

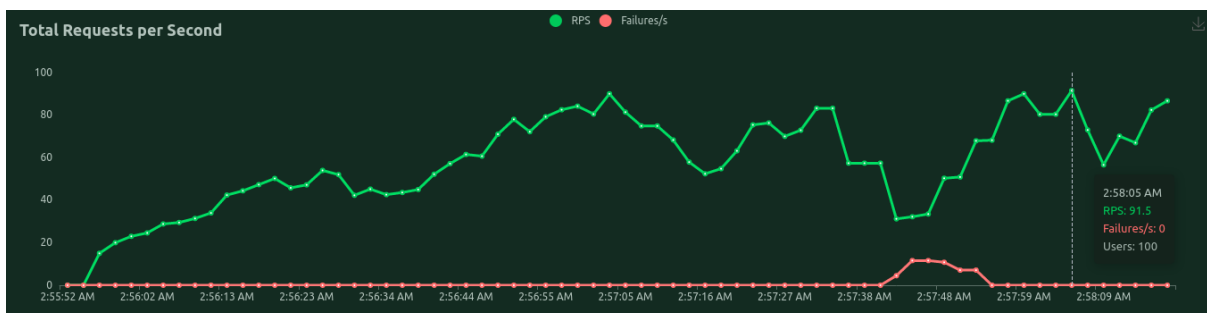


Figuur 33: Meting van het aantal users over tijd van optimalisatie 4.4.

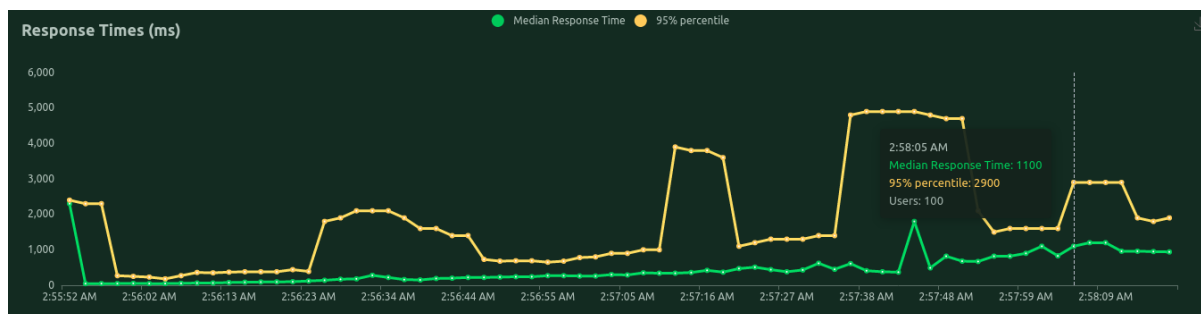
Uit de metingen van [optimalisatie 4.3](#) bleek dat asynchrone endpoints een negatieve invloed heeft. Daarom heb ik Redis ook synchroon getest. Uit de resultaten blijkt dat Redis niet goed werkt met synchrone endpoints. Dat is te zien aan de responstijden, de maximale responstijd is gestegen met 3 seconden. Tevens worden er 500 fouten veroorzaakt doordat de Redis cache de timeout van 4.5 seconden overschrijdt. Dit getal is gebaseerd op de maximale responstijd van [optimalisatie 4.3](#): 4.672 seconden.

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/listing	4625	112	830	1800	4800	1037	39	5145	2709792	41.4	0
GET	/neighbourhood	4620	4	92	510	1700	219	1	4670	1223	41.4	0
Aggregated		9245	116	320	1500	4700	628	1	5145	1356240	82.8	0

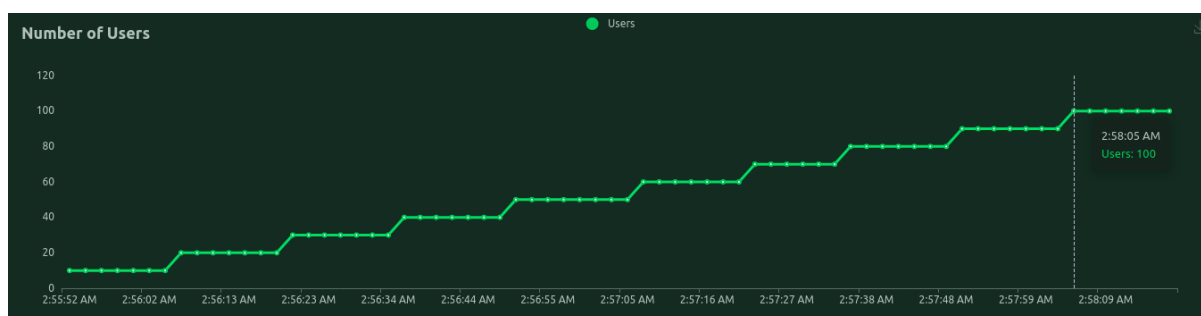
Figuur 34: Tabel met overzicht van optimalisatie 4.4 synchroon.



Figuur 35: Meting van het aantal requests per seconde over tijd van optimalisatie 4.4 synchroon.



Figuur 36: Meting van de responstijd over tijd van optimalisatie 4.4 synchroon.



Figuur 37: Meting van het aantal users over tijd van optimalisatie 4.4 synchroon.

4.5 Conclusie

De belangrijke waarden uit alle metingen zijn samengevat in onderstaand tabel. Het aantal RPS is gestegen met 1519.05%. De responstijden zijn ongeveer met 97% gezakt. Hieruit kun je concluderen dat de performance significant is verbeterd.

Optimalisatie	Database / Cache	Minimale responstijd	Maximale responstijd	Gemiddelde responstijd	Maximaal RPS
Baseline	MSSQL	83 ms	64038 ms	16162 ms	6.3
<u>4.1 AsNoTracking</u>	MSSQL	83 ms	52838 ms	15571 ms	9.9
<u>4.2 DTOs</u>	MSSQL	19 ms	4228 ms	1084 ms	55.1
<u>4.3 Async</u>	MSSQL	19 ms	4672 ms	1107 ms	48.4
<u>4.4 Redis</u>	Redis en MSSQL	1 ms	1995 ms	479 ms	102
VERSCHIL		- 82 ms	- 62043 ms	- 15683 ms	+ 95.7
VERSCHIL %		- 98,80	- 96.88%	- 97.04%	+ 1519.05%

Tabel 4: Samenvatting performance verschillen per optimalisatie..

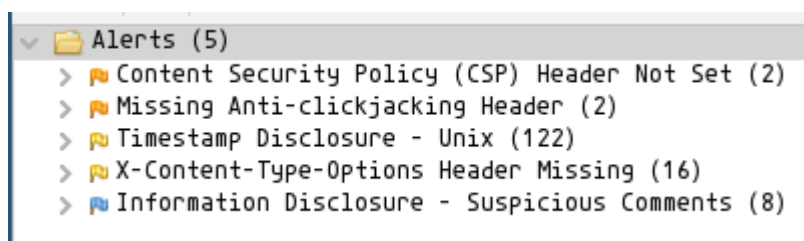
5 Security

In dit hoofdstuk wordt uitleg en context gegeven op security maatregelen die genomen zijn. Er is een security analyse uitgevoerd met behulp van OWASP ZAP. Tevens zijn er handmatige security maatregelen getroffen.

5.1 OWASP ZAP

Om de applicatie te hosten is er gebruikt van een gratis *Student plan*, wat erg gelimiteerd is in resources. Hierdoor werd het erg moeilijk om de OWASP ZAP tool te gebruiken op de gedeployede applicatie. De quota limieten werden al bereikt na 1 scan.

Hierdoor is besloten om de applicatie lokaal te testen en te elaboreren. De onderstaande Alerts (zie figuur 38) zijn de issues die gevonden zijn binnen de webapplicatie.



Figuur 38: Resultaten analyse OWASP ZAP.

5.1.1 Content Security Policy (CSP) Header Not Set

Content Security Policy (CSP) is een extra beveiligingslaag die helpt bij het detecteren en beperken aanvallen zoals maar niet alleen XSS aanvallen. Door middel de CSP kunnen goedgekeurde bronnen worden aangegeven waarvan resources (images, scripts, css stylesheets, etc.) geladen worden.

De CSP kan op verschillende manieren worden toegevoegd. In deze applicatie heb ik besloten om deze toe te voegen via het *Web.config* bestand.

```
<!-- Web.config -->
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    ....
    <system.webServer>
        <httpProtocol>
            <customHeaders>
                <clear />
                <add name="Content-Security-Policy" value="
                    base-uri 'self';
                    block-all-mixed-content;
                    default-src 'self';
```

```

img-src      data:
            https;;
object-src   'none';
script-src   'self'
            'sha256-v8v3RKRpmN4odZ1CWM5gw80QKPCCWMcpNeOmimNL2AA='
            'unsafe-eval';
style-src    'self'
            'sha256-o0jHpRm+vtLUVHXjvHxevgOP6bXpGr2TyJfd+MGdDbQ='
            'sha256-zaa/hg53qR9MPgy+ECJwd0xTOKObjCubYhUrlkrEKxs='
            'sha256-47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU='
            'sha256-9Nkt6zGDtyRXhEF45Xj607dE12YASsy2b157ViK9w6E='
            'sha256-87zpVzuJvtFHgIebXECJF9wkxNQfwaEpzfzuweZn+fo='
            'sha256-z7zcnw/4WalZqx+PrNaRnoeLz/G9WXuFqV1WCJ129sg='
            'unsafe-hashes';
connect-src  'self'
            https://inside-airbnb-amsterdam-api.azurewebsites.net
            https://dev-8nj995io.eu.auth0.com/
            https://api.mapbox.com/
            https://events.mapbox.com/;
frame-src    'self'
            https://dev-8nj995io.eu.auth0.com/;
worker-src   'self'
            blob;;
frame-ancestors 'none';
upgrade-insecure-requests;" />

...
</customHeaders>

...
</httpProtocol>
</system.webServer>

...
</configuration>

```

Figuur 39: Implementatie security issue 5.1.1: Content Security Policy in *Web.config*.

De vorig genoemde oplossing is helaas alleen toepasbaar op een productieserver, zoals gebruikt in Azure. Gelukkig kan een CSP ook doorgegeven worden via <meta> tags in de <head> tags in een HTML bestand.³¹ De Blazor WASM client applicatie van dit project bevat een enkel html bestand: index.html.

```

<!-- index.html -->
<!DOCTYPE html>

```

³¹

<https://docs.microsoft.com/en-us/aspnet/core/blazor/security/content-security-policy?view=aspnetcore-6.0>

```

<html lang="en">
<head>
...
<meta
  http-equiv="Content-Security-Policy"
  content="base-uri 'self';
    block-all-mixed-content;
    default-src 'self';
    img-src data:
      https;;
    object-src 'none';
    script-src 'self'
      'sha256-v8v3RKRPmN4odZ1CWM5gw80QKPCCWMcpNeOmimNL2AA='
      'unsafe-eval';
    style-src 'self'
      'sha256-o0jHpRm+vtLUVHXjvHxevgOP6bXpGr2TyJfd+MGdDbQ='
      'sha256-zaa/hg53qR9MPgy+ECJwd0xTOkObjCubYhUrlkrEKxs='
      'sha256-47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU='
      'sha256-9Nkt6zGDtyRXhEF45Xj607dE12YASsy2b157ViK9w6E='
      'sha256-87zpVzuJvtFHgIebXECJF9wkxNQfwaEpzfzuweZn+fo='
      'sha256-z7zcnw/4WalZqx+PrNaRnoeLz/G9WXuFqV1WCJ129sg='
      'unsafe-hashes';
    connect-src 'self'
      https://localhost:7294
      https://inside-airbnb-amsterdam-api.azurewebsites.net
      https://dev-8nj995io.eu.auth0.com/
      https://api.mapbox.com/
      https://events.mapbox.com/;
    frame-src 'self'
      https://dev-8nj995io.eu.auth0.com/;
    worker-src 'self'
      blob;;
    frame-ancestors 'none';
    upgrade-insecure-requests;"
  />
...
</head>
...

```

Figuur 40: Implementatie security issue 5.1.1: Content Security Policy in een html bestand.

5.1.2 Missing Anti-clickjacking Header

Clickjacking is een aanval waarbij een gebruiker wordt misleid door op een element te klikken dat onzichtbaar is of vermomd is als een ander element. Dit kan ertoe leiden dat

gebruikers bijvoorbeeld onbewust malware downloaden, schadelijke/ongewenste websites bezoeken.

De oplossing voor dit probleem is hetzelfde als besproken in 5.1.2. Er moet een *X-Frame-Options* header toegepast worden in de response headers via *Web.config* of een *frame-ancestors* in de “Content-Security-Policy” in de *meta* tag van een html pagina.

```
<!-- Web.config -->
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    ....
    <system.webServer>
        <httpProtocol>
            <customHeaders>
                <clear />
                <add name="X-Frame-Options" value="DENY"/>
                ...
            </customHeaders>
            ...
        </httpProtocol>
    </system.webServer>
    ...
</configuration>
```

Figuur 41: Implementatie security issue 5.1.2: X-Frame-Options in *Web.config*.

```
<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
    <head>
        ...
        <meta
            http-equiv="Content-Security-Policy"
            content="...
                frame-ancestors 'none';
                ...
        />
        ...
    </head>
    ...

```

Figuur 42: Implementatie security issue 5.1.2: frame-ancestors in content-security-policy.

5.1.3 Timestamp Disclosure - Unix

Naast dat deze Alert een false positive is, is het ook niet in een door mij beheerd bestand gevonden. Er is een waarde in *bootstrap.min.css* gevonden die toevallig vertaald naar een Unix timestamp. Echter representeert deze waarde een kleur en geen timestamp.

```
background-image:url("data:image/svg+xml,%3csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 8 8'%3e%3cpath
fill='%23198754' d='M2.3 6.73L6 4.53c-.4-1.04.46-1.4 1.1-.811.1 1.4
3.4-3.8c.6-.63 1.6-.27 1.2.71-4 4.6c-.43.5-.8.4-1.1.1z'/%3e%3c/svg%3e")
```

Figuur 43: Unix timestamp disclosure false positive. De waarde is gemarkeerd in donker geel.

5.1.4 X-Content-Type-Options Header Missing

Door het weglaten van deze header kunnen oudere versies van Internet Explorer en Chrome MIME-sniffing uitvoeren op de response, waardoor de response inhoud als een ander/verkeerd type kan worden geïnterpreteerd.

De oplossing is om een X-Content-Type-Options header met als waarde 'nosniff' aan een response mee te sturen. Dit kan via *Web.config*.

```
<!-- Web.config -->
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  ....
  <system.webServer>
    <httpProtocol>
      <customHeaders>
        <clear />
        <add name="X-Content-Options" value="nosniff"/>
        ...
      </customHeaders>
      ...
    </httpProtocol>
  </system.webServer>
  ...
</configuration>
```

Figuur 44: Implementatie security issue 5.1.4: X-Content-Options in *Web.config*.

5.1.5 Information Disclosure - Suspicious Comments


In de stylesheets en scripts staan comments die een aanvaller kunnen helpen met het exploiteren van de applicatie. Echter staan deze comments in bestanden die ik niet zelf in mijn beheer heb, zoals scripts van Mapbox, Bootstrap en ChartJS. De oplossing is om deze comments te evalueren op informatie en eventueel te verwijderen.

5.2 Eigen maatregelen

5.2.1 Endpoint protection

Auth0 maakt het mogelijk om endpoints van een API te beveiligen op basis van autorisatie en rollen. Hier heb ik gebruik gemaakt door de [GET] /statistics endpoint alleen toegang te geven aan gebruikers met de rol "admin". Hiervoor moesten eerst permissies worden toegevoegd in de auth0 portal. Deze permissies zijn gekoppeld aan een user role. Vervolgens kan er door middel van een simpel attribute in de StatisticsController toegang verlenen aan deze rollen.

[← Back to APIs](#)



Inside AirBNB Amsterdam API

Custom API Identifier `https://localhost:7294`

[Quick Start](#) [Settings](#) [Permissions](#) [Machine to Machine Applications](#)

Add a Permission (Scope)

Define the permissions (scopes) that this API uses.

Permission (Scope) *	Description *
<input type="text" value="read:appointments"/>	<input type="text" value="Read your appointments"/>

[+ Add](#)

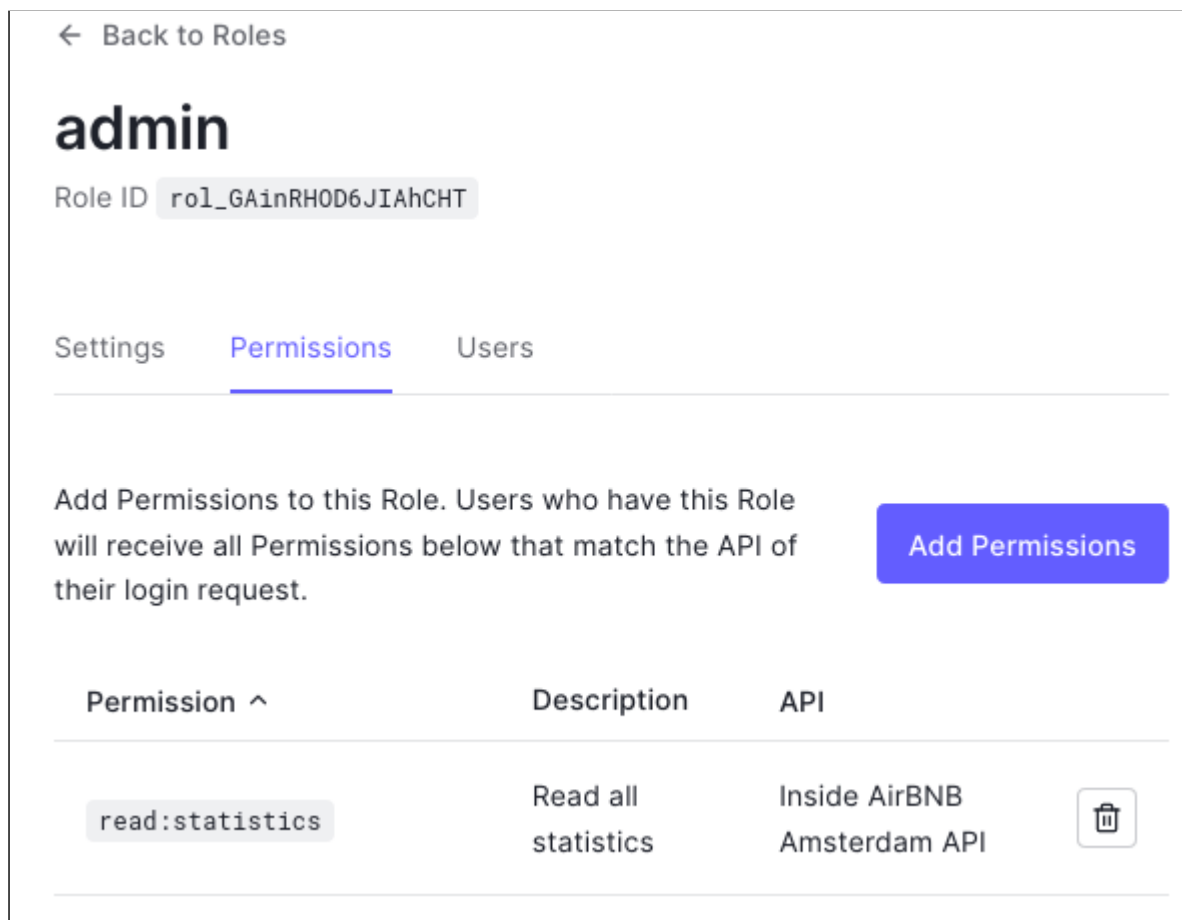
List of Permissions (Scopes)

These are all the permissions (scopes) that this API uses.

Permission	Description
<code>read:statistics</code>	Read all statistics

30

Figuur 45: Implementatie security issue 5.2.1: API permissies in de auth0 portal.



Figuur 46: Implementatie security issue 5.2.1: permissie toekenning aan een rol in de auth0 portal.

```
// api/Program.cs
builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("ReadStatistics", policy =>
        policy.RequireClaim("permissions", "read:statistics")
    );
});
```

Figuur 47: Implementatie security issue 5.1.4: X-Content-Options in *Web.config*.

```
// StatisticsController.cs
[ApiController]
[Route("[controller]")]
+ [Authorize(Policy = "ReadStatistics")]
public class StatisticsController : ControllerBase, IStatisticsController
{
```

Figuur 48: Implementatie security issue 5.2.1 in *StatisticsController.cs*.

5.3 Conclusie

Zoals te zien is in figuur 38 zijn er 5 security alerts gevonden door ZAP. Hiervan heb ik er 3 op kunnen lossen. De overige 2 alerts heb ik niet op kunnen lossen, omdat een van de alerts false positive is en de ander zit in de stylesheet van Mapbox.

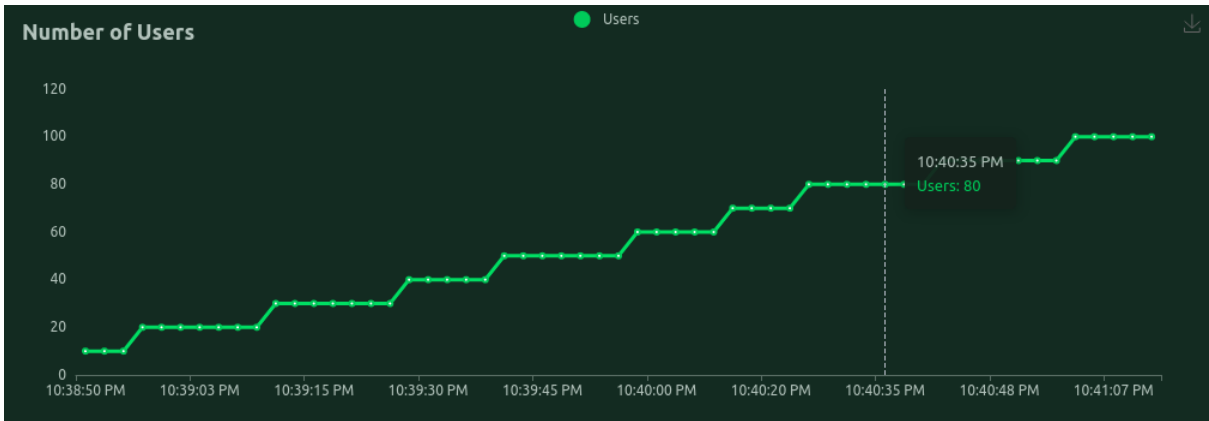
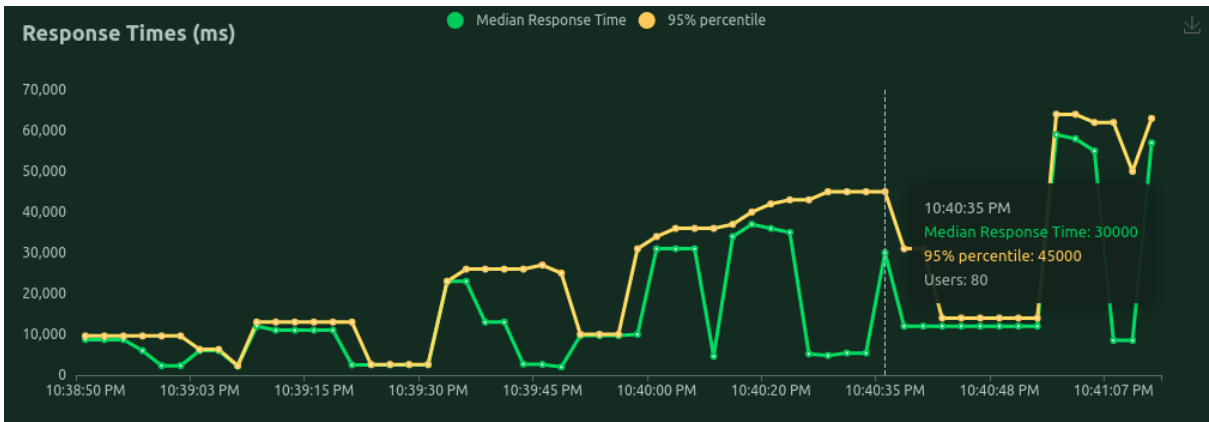
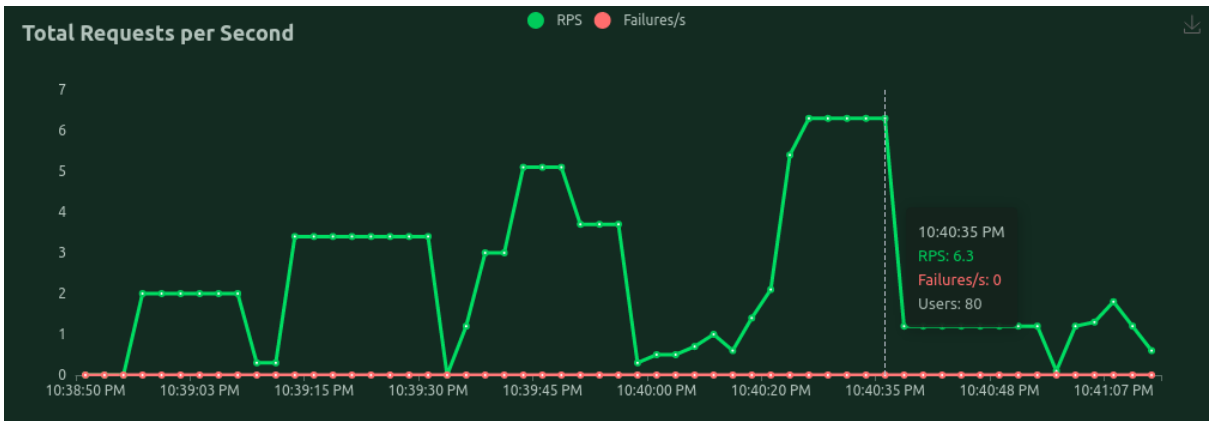
Veel voorkomende security issues zijn tevens al opgelost door gebruik te maken van ASP.NET Core, Entity Framework en Auth0. Daarbij heb ik zelf nog gedacht aan het beveiligen van de [GET] /statistics endpoint.

Bijlagen

Performance tests

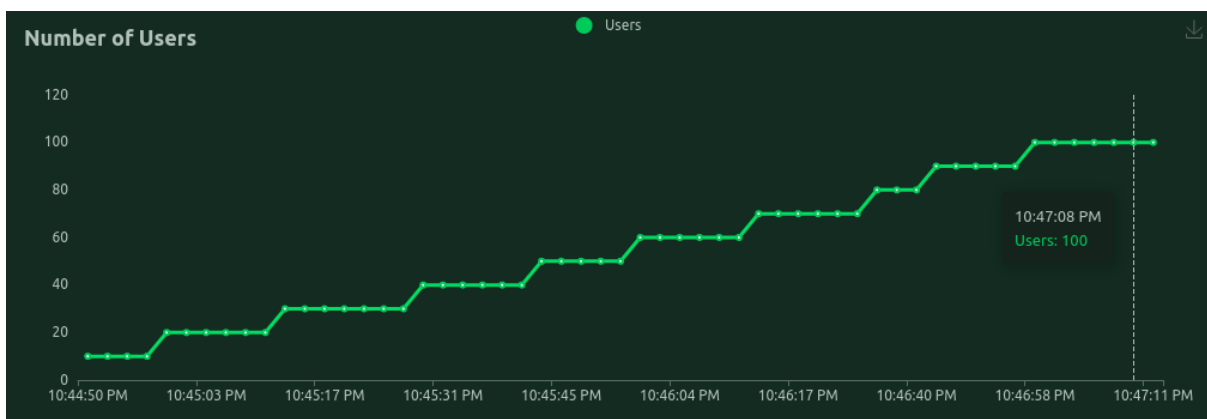
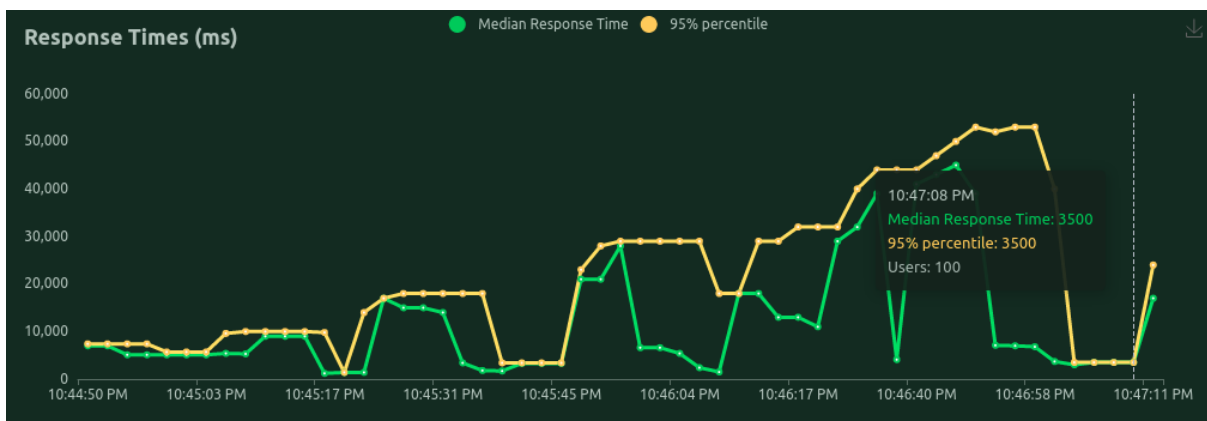
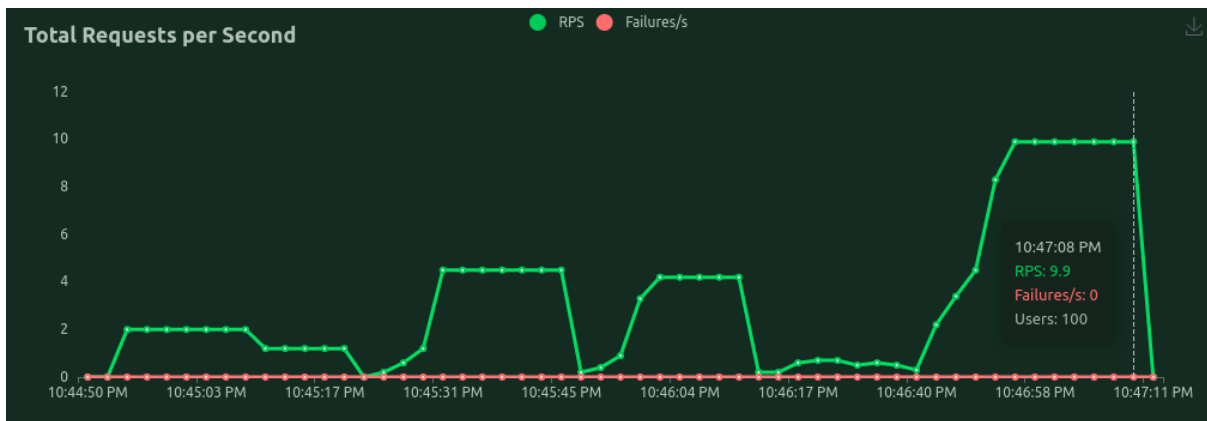
Baseline

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/listing	143	0	26000	50000	63000	28459	5984	64038	107684529	0.1	0
GET	/neighbourhood	141	0	2600	9700	14000	3691	83	13679	2391	0.5	0
Aggregated		284	0	9400	41000	62000	16162	83	64038	54222622	0.6	0



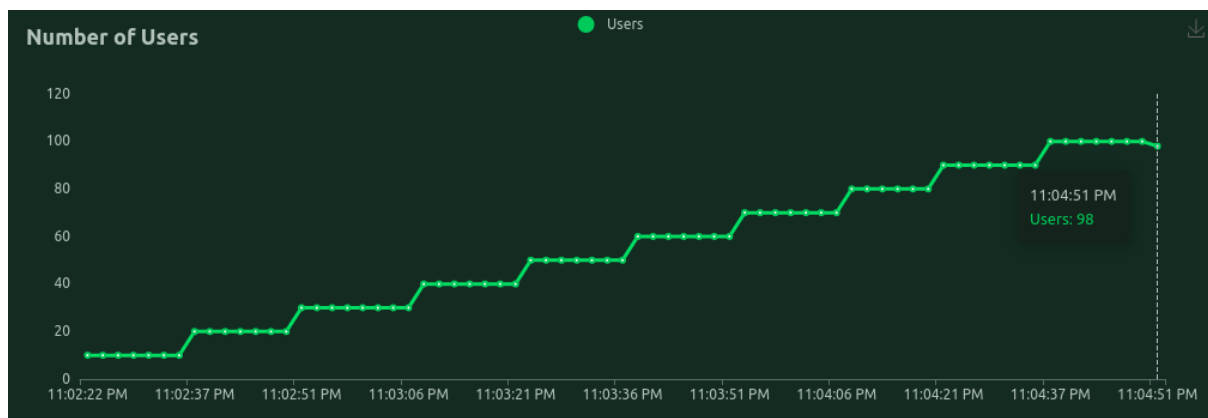
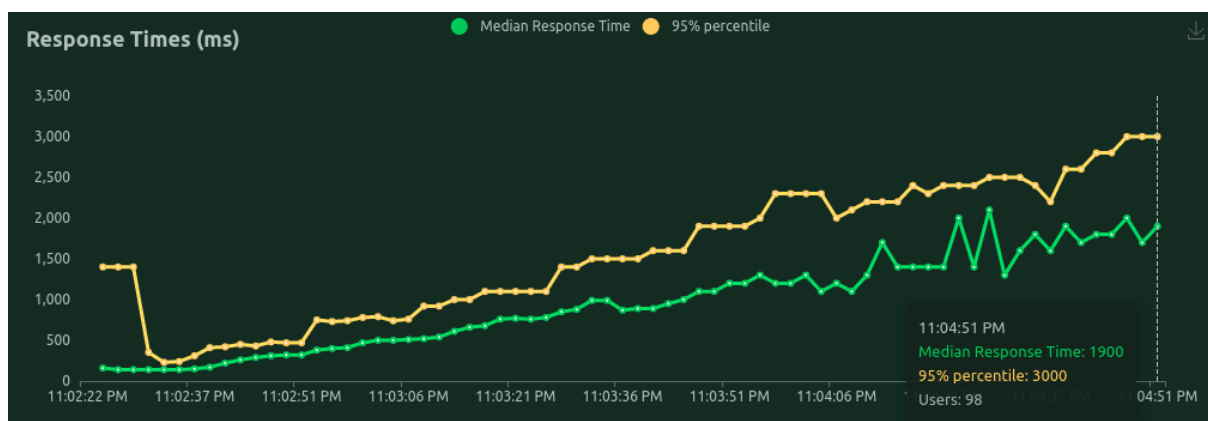
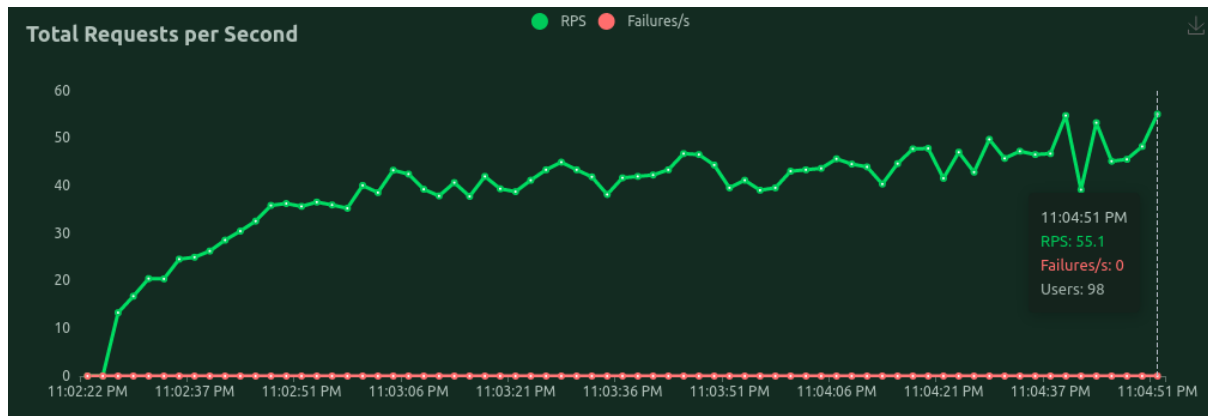
Optimalisatie 4.1 AsNoTracking()

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/listing	189	1	28000	47000	53000	27625	5076	52838	107114770	0	0
GET	/neighbourhood	188	0	2400	7000	17000	3454	83	17089	2391	0	0
Aggregated		377	1	7200	43000	53000	15571	83	52838	53700639	0	0



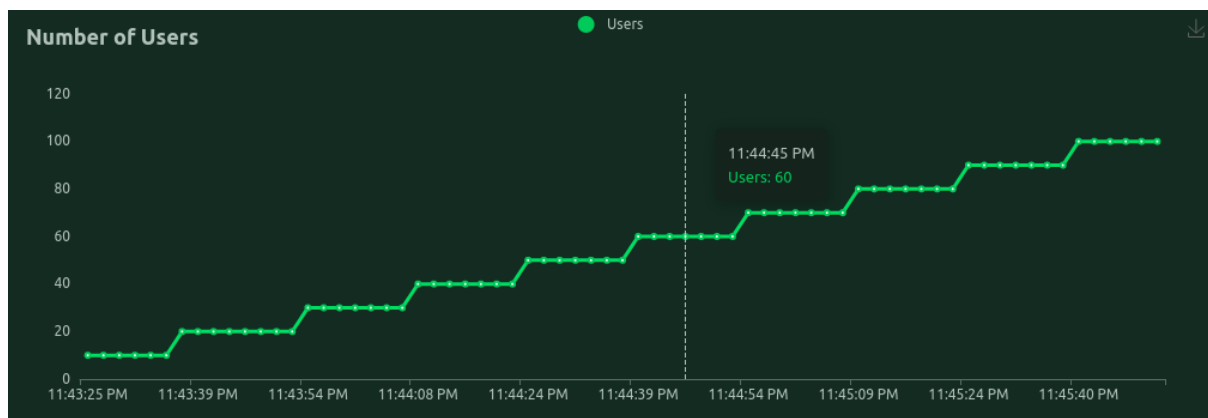
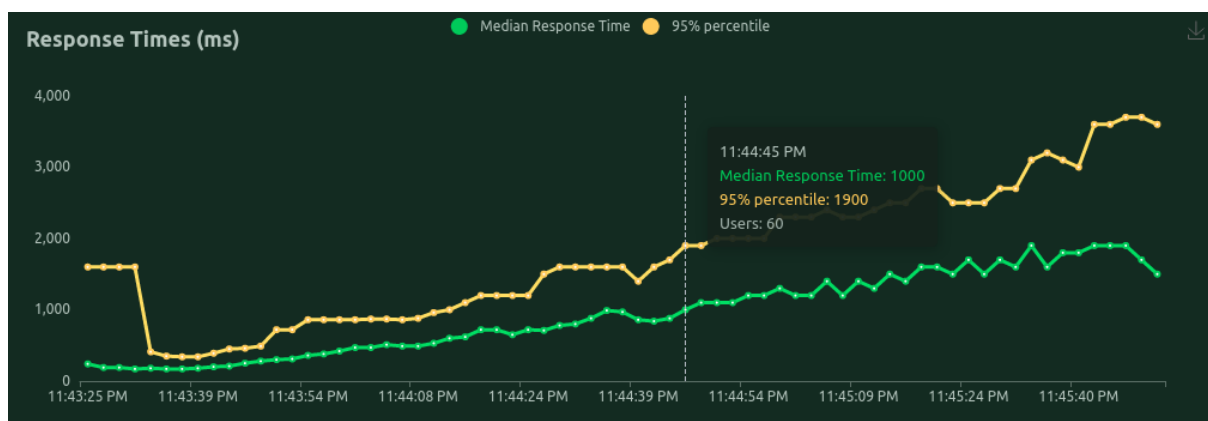
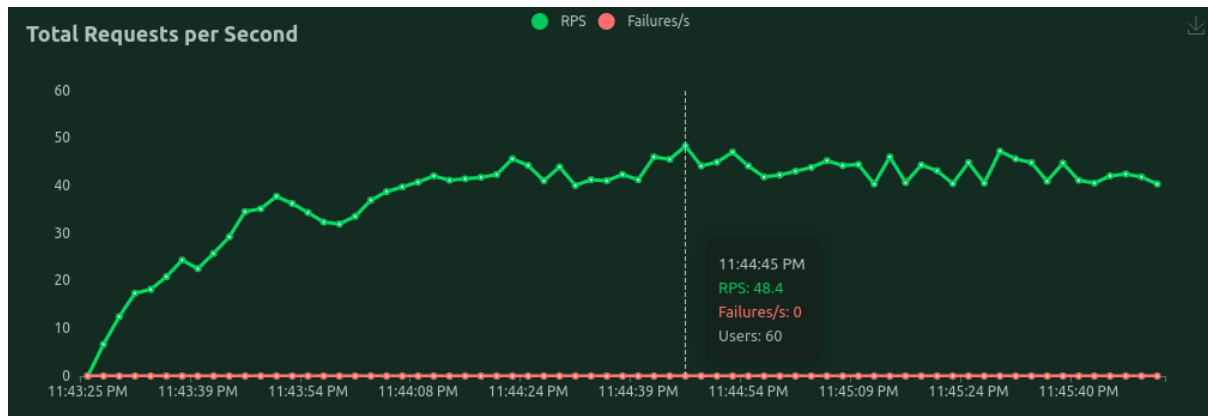
Optimalisatie 4.2 DTO's

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/listing	3115	0	1400	2300	2800	1341	44	4228	2776942	28.6	0
GET	/neighbourhood	3040	0	780	1400	2600	822	19	3009	1221	26.5	0
Aggregated		6155	0	970	2100	2800	1084	19	4228	1405993	55.1	0



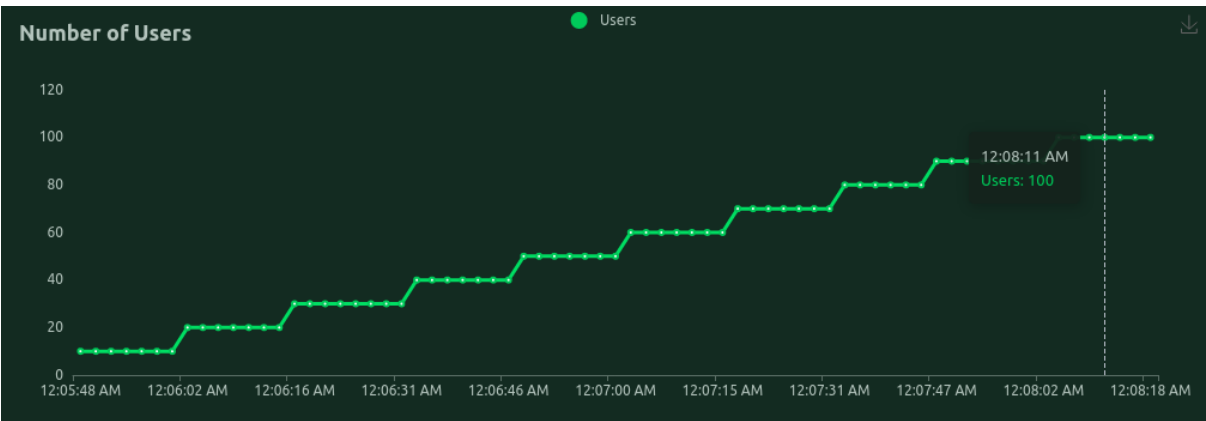
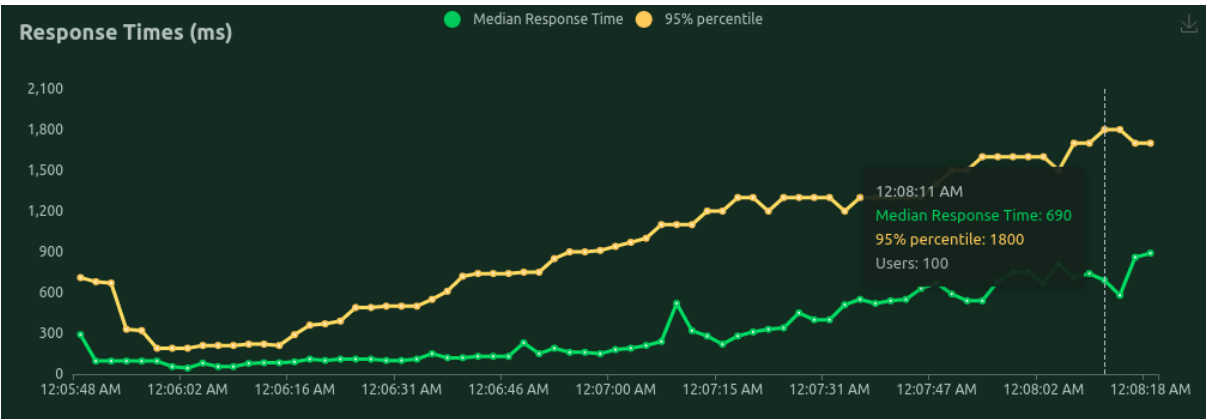
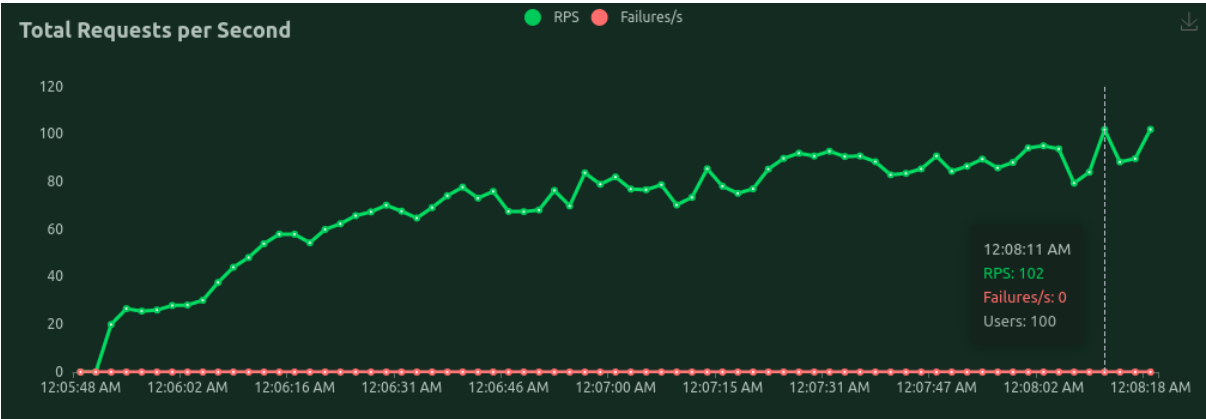
Optimalisatie 4.3 Async

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/listing	3035	0	1100	2600	3600	1281	63	4672	2776942	22.1	0
GET	/neighbourhood	2954	0	810	1800	2700	929	19	4172	1221	21.2	0
Aggregated		5989	0	920	2300	3500	1107	19	4672	1407852	43.3	0



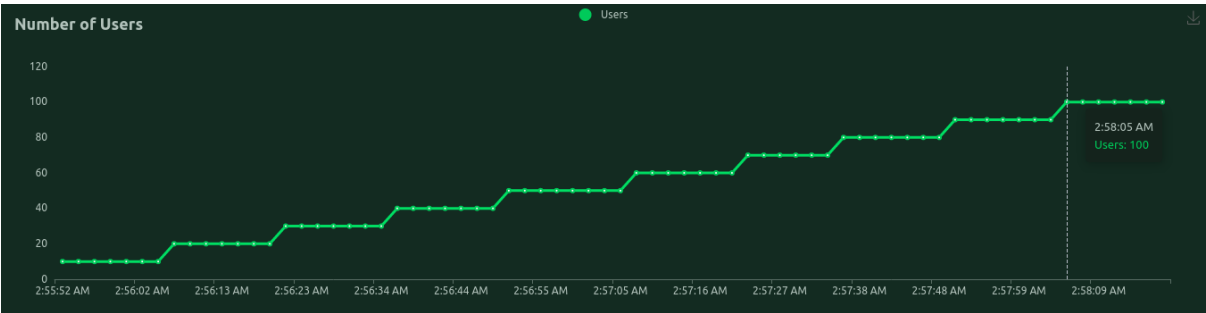
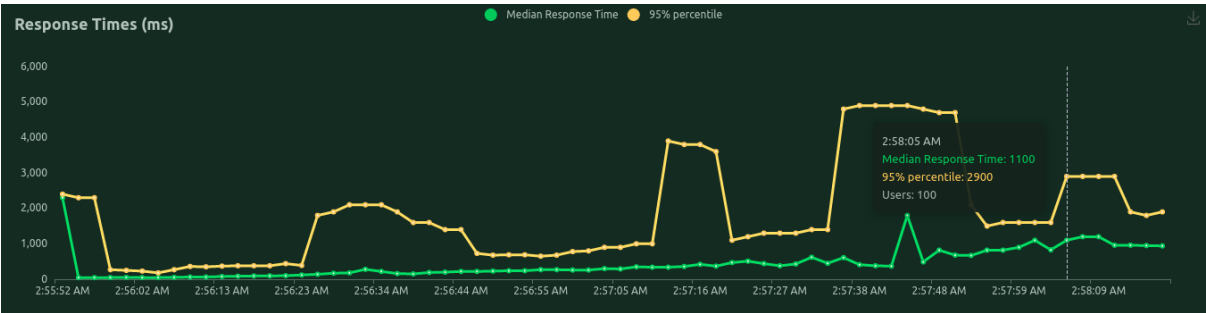
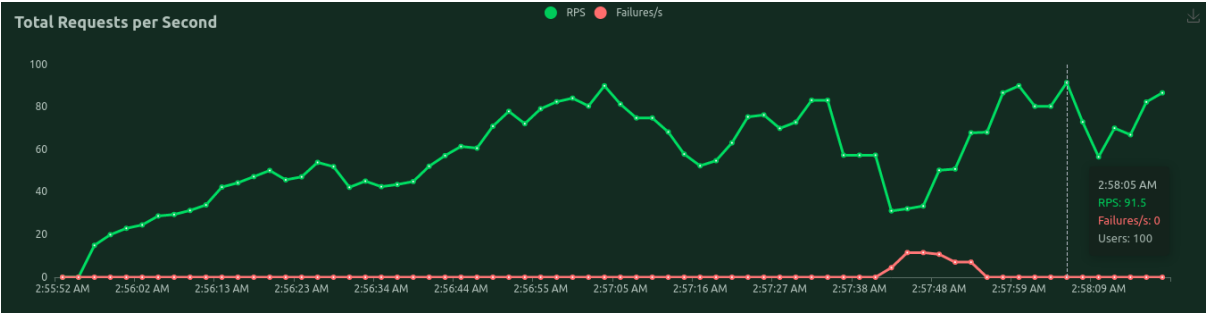
Optimalisatie 4.4 Redis cache

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/listing	5700	0	750	1500	1800	779	43	1995	2776942	49.7	0
GET	/neighbourhood	5623	0	60	540	1200	176	1	1530	1221	52.4	0
Aggregated		11323	0	230	1300	1700	479	1	1995	1398519	102.1	0



Optimalisatie 4.4 Redis cache synchron

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/listing	4625	112	830	1800	4800	1037	39	5145	2709792	41.4	0
GET	/neighbourhood	4620	4	92	510	1700	219	1	4670	1223	41.4	0
Aggregated		9245	116	320	1500	4700	628	1	5145	1356240	82.8	0



Literatuurlijst

1. Auth0. (z.d.-a). Auth0. Auth0 Docs. Geraadpleegd op 1 juni 2022, van <https://auth0.com/docs/>
2. Auth0. (z.d.-b). How Uses Identity Industry Standards. Geraadpleegd op 1 juni 2022, van <https://auth0.com/learn/how-auth0-uses-identity-industry-standards/>
3. Auth0. (2021, 2 november). Role-Based Authorization for ASP.NET Web APIs. Auth0 - Blog. Geraadpleegd op 10 juni 2022, van <https://auth0.com/blog/role-based-authorization-for-aspnet-webapi/>
4. Auth0. (2022a, maart 31). Securing Blazor WebAssembly Apps. Auth0 - Blog. Geraadpleegd op 10 juni 2022, van <https://auth0.com/blog/securing-blazor-webassembly-apps/>
5. Auth0. (2022b, maart 31). Securing Blazor WebAssembly Apps. Auth0 - Blog. Geraadpleegd op 10 juni 2022, van <https://auth0.com/blog/securing-blazor-webassembly-apps/>
6. Auth0. (2022c, maart 31). Securing Blazor WebAssembly Apps. Auth0 - Blog. Geraadpleegd op 10 juni 2022, van <https://auth0.com/blog/securing-blazor-webassembly-apps/>
7. Auth0. (2022d, maart 31). Securing Blazor WebAssembly Apps. Auth0 - Blog. Geraadpleegd op 10 juni 2022, van <https://auth0.com/blog/securing-blazor-webassembly-apps/>
8. Brandi, D. (2021, 11 december). The “Real” Repository Pattern in Android - ProAndroidDev. Medium. Geraadpleegd op 1 juni 2022, van <https://proandroiddev.com/the-real-repository-pattern-in-android-efba8662b754>
9. Chart.js | Open source HTML5 Charts for your website. (z.d.). ChartJS. Geraadpleegd op 1 juni 2022, van <https://www.chartjs.org/>
10. Docker. (2022a, juni 9). Docker Engine overview. Docker Documentation. Geraadpleegd op 10 juni 2022, van <https://docs.docker.com/engine/>
11. Docker. (2022b, juni 9). Overview of Docker Compose. Docker Documentation. Geraadpleegd op 10 juni 2022, van <https://docs.docker.com/compose/>
12. El Kaddouri, I. (z.d.). inside-airbnb-amsterdam/docker-compose.yml at main · ilivss/inside-airbnb-amsterdam. GitHub. Geraadpleegd op 10 juni 2022, van <https://github.com/ilivss/inside-airbnb-amsterdam/blob/main/docker-compose.yml>
13. Locust. (z.d.-a). Locust.io. Geraadpleegd op 10 juni 2022, van <https://locust.io/>
14. Locust. (z.d.-b). Writing a locustfile — Locust 2.9.0 documentation. Locust Docs. Geraadpleegd op 10 juni 2022, van <https://docs.locust.io/en/stable/writing-a-locustfile.html>
15. Mapbox GL JS. (z.d.). Mapbox. Geraadpleegd op 10 juni 2022, van <https://docs.mapbox.com/mapbox-gl-js/guides/>
16. Microsoft. (z.d.-a). ASP.NET Web APIs | Rest APIs with .NET and C#. Geraadpleegd op 10 juni 2022, van <https://dotnet.microsoft.com/en-us/apps/aspnet/apis>
17. Microsoft. (z.d.-b). DbExtensions.AsNoTracking Method (System.Data.Entity). Microsoft Docs. Geraadpleegd op 10 juni 2022, van <https://docs.microsoft.com/en-us/dotnet/api/system.data.entity.dbextensions.asnottracking?view=entity-framework-5.0.0>
18. Microsoft. (z.d.-c). Microsoft.AspNetCore.Components.WebAssembly.Authentication Namespace. Microsoft Docs. Geraadpleegd op 10 juni 2022, van

- <https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.components.webassembly.authentication?view=aspnetcore-6.0>
19. Microsoft. (z.d.-d). Microsoft.EntityFrameworkCore.Design Namespace. Microsoft Docs. Geraadpleegd op 10 juni 2022, van <https://docs.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.design?view=efcore-6.0>
 20. Microsoft. (z.d.-e). Microsoft.Extensions.Http Namespace. Microsoft Docs. Geraadpleegd op 10 juni 2022, van <https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.http?view=dotnet-plat-ext-6.0>
 21. Microsoft. (z.d.-f). RedisCache Class (Microsoft.Extensions.Caching.StackExchangeRedis). Microsoft Docs. Geraadpleegd op 10 juni 2022, van <https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.caching.stackexchange.redis.redisocache?view=dotnet-plat-ext-3.1>
 22. Microsoft. (2021, 3 december). Microsoft SQL Server Database Provider - EF Core. Microsoft Docs. Geraadpleegd op 10 juni 2022, van <https://docs.microsoft.com/en-us/ef/core/providers/sql-server/?tabs=dotnet-core-cli>
 23. Microsoft. (2022a, maart 9). Reverse Engineering - EF Core. Microsoft Docs. Geraadpleegd op 10 juni 2022, van <https://docs.microsoft.com/en-us/ef/core/managing-schemas/scaffolding?tabs=dotnet-core-cli>
 24. Microsoft. (2022b, maart 15). Microsoft SQL documentation - SQL Server. Microsoft Docs. Geraadpleegd op 1 juni 2022, van <https://docs.microsoft.com/en-us/sql/?view=sql-server-ver16>
 25. Microsoft. (2022c, mei 11). EF Core tools reference (.NET CLI) - EF Core. Microsoft Docs. Geraadpleegd op 10 juni 2022, van <https://docs.microsoft.com/en-us/ef/core/cli/dotnet>
 26. Microsoft. (2022d, juni 3). ASP.NET Core Blazor. Microsoft Docs. Geraadpleegd op 1 juni 2022, van <https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-6.0>
 27. Microsoft. (2022e, juni 3). Enforce a Content Security Policy for ASP.NET Core Blazor. Microsoft Docs. Geraadpleegd op 10 juni 2022, van <https://docs.microsoft.com/en-us/aspnet/core/blazor/security/content-security-policy?view=aspnetcore-6.0>
 28. Microsoft. (2022f, juni 8). Call JavaScript functions from .NET methods in ASP.NET Core Blazor. Microsoft Docs. Geraadpleegd op 10 juni 2022, van <https://docs.microsoft.com/en-us/aspnet/core/blazor/javascript-interoperability/call-javascript-from-dotnet?view=aspnetcore-6.0>
 29. Microsoft. (2022g, juni 8). Call .NET methods from JavaScript functions in ASP.NET Core Blazor. Microsoft Docs. Geraadpleegd op 10 juni 2022, van <https://docs.microsoft.com/en-us/aspnet/core/blazor/javascript-interoperability/call-dotnet-from-javascript?view=aspnetcore-6.0>
 30. Muntean, M. (z.d.). GitHub - mariusmuntean/ChartJs.Blazor: Brings Chart.js charts to Blazor. GitHub. Geraadpleegd op 1 juni 2022, van <https://github.com/mariusmuntean/ChartJs.Blazor>
 31. OWASP. (z.d.-a). OWASP ZAP – Content Security Policy (CSP) Header Not Set. Zaproxy. Geraadpleegd op 10 juni 2022, van <https://www.zaproxy.org/docs/alerts/10038/>

32. OWASP. (z.d.-b). OWASP ZAP – Information Disclosure - Suspicious Comments. Zaproxy. Geraadpleegd op 10 juni 2022, van <https://www.zaproxy.org/docs/alerts/10027/>
33. OWASP. (z.d.-c). OWASP ZAP – Missing Anti-clickjacking Header. Zaproxy. Geraadpleegd op 10 juni 2022, van <https://www.zaproxy.org/docs/alerts/10020-1/>
34. OWASP. (z.d.-d). OWASP ZAP – Timestamp Disclosure. Zaproxy. Geraadpleegd op 10 juni 2022, van <https://www.zaproxy.org/docs/alerts/10096/>
35. OWASP. (z.d.-e). OWASP ZAP – X-Content-Type-Options Header Missing. Zaproxy. Geraadpleegd op 10 juni 2022, van <https://www.zaproxy.org/docs/alerts/10021/>
36. Redis. (2022, 2 juni). In-Memory Caching Solutions | Redis Enterprise. Geraadpleegd op 1 juni 2022, van <https://redis.com/solutions/use-cases/caching/>