Giel Jurriëns

500749216

Festival Toilets Backend

Inhoud

[Inleiding 2](#_Toc31307246)

[Back-end 2](#_Toc31307247)

[Front-end 3](#_Toc31307248)

[Conclusie 4](#_Toc31307249)

# Inleiding

Dit project bestaat uit meerdere onderdelen een back-end een front-end en de hardware zelf. Ik zal in dit document alleen een uitleg geven over de front-end en back-end, de hardware is in een eerdere opdracht/document al uitleg over gegeven.

# Back-end

De back-end is gemaakt in .NET CORE (C#). De back-end is een API waar de hardware https POST en PUT requests naar stuurt om de status van de toiletten door te geven. Deze data wordt vervolgens opgeslagen in een SQLite database. Om deze database op te zetten heb ik gebruik gemaakt van een code first methode.

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<FestivalToiletsContext>(opt =>

opt.UseInMemoryDatabase("FestivalList"));

services.AddControllers();

}

Zoals in de code hierboven te zien is het een tijdelijke lokale database die na het afsluiten van de computer of legen van memory niet meer bestaat.

De back-end draait op dit moment op een lokale IIS server en kan met de volgende url gevonden worden: <https://localhost:7502/api/festivals>

Na een get request geeft de back-end een JSON array terug. De post en put requests maken gebruik van JSON-objecten om data door te geven.

# Front-end

De front-end is ook gemaakt in .NET CORE (C#) hier word gebruik gemaakt van MVC en viewcomponents. De styling wordt geregeld met het bootstrap framework en JQuery wordt gebruikt voor de AJAX requests.

De front-end stuurt een get request naar de back-end om zo actuele informatie over de toiletten op de webpagina te tonen.

protected string api = "https://localhost:7502/api/toilets";

public string Get() {

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(api);

request.AutomaticDecompression = DecompressionMethods.GZip | DecompressionMethods.Deflate;

using (HttpWebResponse response = (HttpWebResponse)request.GetResponse())

using (Stream stream = response.GetResponseStream())

using (StreamReader reader = new StreamReader(stream))

{

return reader.ReadToEnd();

}

}

Bovenstaande code wordt elke seconde doormiddel van een AJAX request aangeroepen om zo altijd de nieuwste data te hebben.

var container = $("#toilet\_container");

var refreshComponent = function () {

$.get("/Home/ToiletViewComponent", function (data) { container.html(data); });

};

$(function () { window.setInterval(refreshComponent, 1000); });

Daarnaast weergeeft de front-end ook de temperatuurt van Amsterdam weer, hiervoor is de [openweather](https://openweathermap.org/) api gebruikt.

protected string url = "http://api.openweathermap.org/data/2.5/weather?units=metric";

protected string apiKey = "b444261e71d481871f3048ad80c05dc6";

protected string location = "Amsterdam,nl";

public string Get()

{

string api = url + "&q=" + location + "&appid=" + apiKey;

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(api);

request.AutomaticDecompression = DecompressionMethods.GZip | DecompressionMethods.Deflate;

using (HttpWebResponse response = (HttpWebResponse)request.GetResponse())

using (Stream stream = response.GetResponseStream())

using (StreamReader reader = new StreamReader(stream))

{

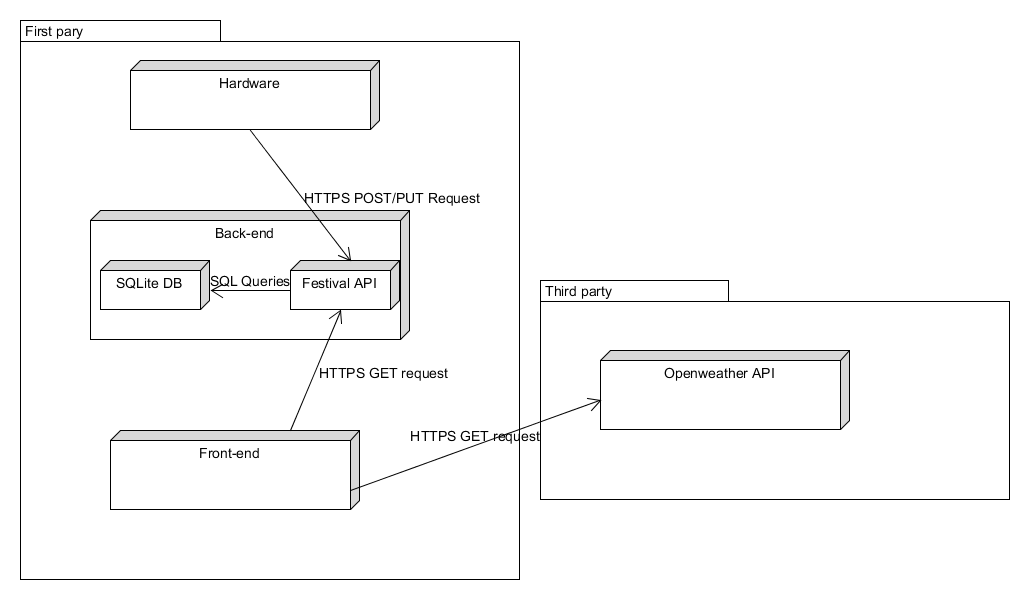
return reader.ReadToEnd();

}

}

# Conclusie

Mijn applicatie bestaat dus uit 3 lagen, de hardware laag, de back-end en de front-end. De front-end maakt connectie naar beide API (back-end en openweather).

De structuur ziet er als volgt uit: