Technisch ontwerp

Portfoliowebsite Noah Kamphuisen

Door: Noah Kamphuisen

Datum: 7 oktober 2024

Versie: 1.0

Inhoudsopgave

[1. Inleiding 3](#_Toc180064832)

[2. Technisch ontwerp 3](#_Toc180064833)

[2.1. Gekozen software en tools 3](#_Toc180064834)

[2.2. Gekozen programmeertaal en databasesysteem 3](#_Toc180064835)

[2.3. API 4](#_Toc180064836)

[2.4. Technische uitwerking WebSocket 4](#_Toc180064837)

# Inleiding

Om voor mijn stage goed te kunnen solliciteren, ga ik een portfoliowebsite bouwen waarop ik mijn projecten en vaardigheden kan laten zien.

# Technisch ontwerp

Bij mijn eindproduct (een portfoliowebsite) horen 3 delen: front-end, back-end en een WebSocket.

**Front-end:** dit zal het portfolio zelf worden. Op de voorpagina wil ik een aantal dingen tonen:

* Kleine introductie over mezelf.
* Overzicht van programmeerskills + favoriete Tech Stack.
* Ervaringen van anderen.
* Lijst met projecten (wanneer op een project wordt geklikt wordt er gelinkt naar een nieuwe pagina, met details over dat project).

**Back-end:** hiervoor ga ik een extern CMS gebruiken (Strapi). In dit CMS gaat het mogelijk worden om onder andere alle teksten aan te passen (ook in verschillende talen) en skills toe te voegen.

**WebSocket:** zie technische uitwerking WebSocket

## Gekozen software en tools

Voor dit project gebruik ik een paar verschillende tools:

* **Github** voor versiebeheer
* **Notion** voor de documentatie van onderzoeken en planning

## Gekozen programmeertaal en databasesysteem

De basis wordt gemaakt in Next.js (App Router), omdat ik veel front-end functionaliteit wil toevoegen. Aangezien Next.js een soort extensie van React.js is, is het makkelijker om geavanceerde front-end features toe te voegen zonder nog een externe library te gebruiken. Hoewel ik ook een headless CMS gebruik, vind ik het toch fijn om een full-stack framework als backend te hebben, zodat ik veel gebruik kan maken van SSR en data fetching aan de serverkant.

Als CMS heb ik gekozen voor Strapi: een headless CMS dat gebaseerd is op Node.js. Dit CMS heb ik gekozen nadat ik uitgebreid onderzoek heb gedaan naar systemen (zie onderzoek op mijn Notion) en uiteindelijk toch op Strapi uit ben gekomen. Als database provider voor Strapi heb ik gekozen voor PostgreSQL, aangezien ik uit ervaring (gecorrupte databases) heb gemerkt dat Postgres zeker tijdens development stabieler is dan MySQL.

Voor de styling maak ik gebruik van SCSS (voornamelijk met PostCSS modules); hierdoor worden classes gekoppeld aan een willekeurig ID, waardoor conflicterende styles zo goed als vermeden worden. Ook maak ik gebruik van Tailwind, zodat ik niet voor elk klein stylingdetail en nieuwe class aan moet maken.

Verder gebruik ik nog een paar NPM dependencies. De dependencies die het meest van belang zijn:

* gsap (gebruiksvriendelijke animatie library)
* next-intl (library voor Next.js om gemakkelijk internationalisatie toe te voegen, door middel van middleware)
* axios (werkt zo goed als hetzelfde als de javascript fetch API, maar heeft fijne features, zoals het automatisch omzetten naar JSON)

Uiteraard is de volledige package.json te vinden in de GIT-repo.

Ook ga ik een websocket integreren in mijn portfoliowebsite, waardoor gebruikers de cursors van andere gebruikers kunnen zien. De websocket gaat gerund worden in een los Node.js project, dat gebruik maakt van de ‘ws’ library. Ik connect het hoofdproject dan door via JavaScript een nieuwe websocketconnectie op te zetten.

Om het project uiteindelijk te hosten, ga ik een Raspberry Pi 5 gebruiken met 8GB geheugen. Deze keuze heb ik gemaakt omdat het helaas niet mogelijk is om via DirectAdmin (hosting van GLUwebsite) een Node.js applicatie te runnen. Het voordeel van een Pi is dat ik de volledige controle heb over de server. Om de server te beveiligen maak ik gebruik van CloudFlare tunneling.

## API

Om de data te fetchen vanuit het CMS, gebruik ik API calls, met GraphQL. Strapi heeft ingebouwde GraphQL support en is daarom makkelijk te fetchen met een simpele query van bijvoorbeeld Axios. Hoewel de verschillende pagina’s (homepagina, project-detailpagina) al vast zijn gesteld in mijn Next.js project, ga ik vrijwel alle teksten en afbeeldingen uit de API halen, ook metadata zoals paginatitels. Om de data goed op te halen/te verwerken voordat het naar de views gestuurd wordt ga ik 3 files gebruiken:

* fetcher.ts (maakt API calls en bundelt data van verschillende calls)
* types.ts (definieert de TypeScript interfaces voor de verschillende datatypes)
* models.ts (voert operaties uit op de opgehaalde data, zoals het omzetten van de relatieve URLs van afbeeldingen naar absolute URLs, aangezien de afbeeldingen in Strapi worden opgeslagen)

Dit zorgt ervoor dat ik functies voor het verwerken van data makkelijk kan hergebruiken.

## Technische uitwerking WebSocket

**Uitwerking serverkant**:

Voor mijn WebSocket heb ik een apart Node.js project. Met de ws-library zet ik de server op, op een andere port dan mijn hoofdproject. Wanneer een user een WebSocket-connectie aanmaakt wordt er een nieuwe entry in de ‘clients’ Javascript-map aangemaakt. Als key wordt de WebSocket instance gebruikt en als value wordt een object met voornamelijk metadata gebruikt. In de metadata wordt het volgende opgeslagen:

* Username (randomly generated)
* Kleur (random kleur uit een array met hexcodes)
* LastMove

Om ervoor te zorgen dat er niet teveel inactieve gebruikers zijn op de WebSocket, ga ik timeouts gebruiken. Elke keer dat de gebruiker beweegt of scrollt, wordt de LastMove value in de metadata aangepast naar het huidige tijdstip. Als de server een message binnen krijgt met ‘type: “cursor”’, stuurt hij een message terug naar alle clients (behalve de sender) met de username en de geupdate positie van de gebruiker die de message heeft gestuurd. Om de 15 seconden checkt de server of er gebruikers zijn die al 30 seconden niet actief zijn geweest, als dit het geval is wordt de gebruiker in kwestie gekickt en verwijderd van de map met clients.

**Uitwerking frontend**:

Om de positie van cursors bij te houden, ga ik een event listener met ‘mousemove’ gebruiken. Wanneer de gebruiker diens muis beweegt, wordt er een message naar de serverkant gestuurd met de x- en y-coördinaten. Ook wordt er een timeout gezet van een paar milliseconden. Wanneer deze timeout actief is, worden er geen nieuwe messages naar de server gestuurd, aangezien dit de server zou kunnen overbelasten. Als de timeout over is, begint deze cyclus weer opnieuw.