**THE RANKING GAME**

**Applicatie Opzet**

Er is een flow om een spel op te zetten (door de quiz master), om een spel in te vullen (door een deelnemer), om een spel te spelen (door een deelnemer) en om het spel te begeleiden (door de quiz master).

“Vul de namen en e-mail adressen in van de deelnemers van het spel. De naam die ze nu krijgen, wordt ook gebruikt om de ranking vragen in te vullen, en om de quiz te spelen. Het e-mail adres wordt gebruikt om de deelnemers uit te nodigen, en toegang te verlenen tot de quiz.”

Het zou optioneel kunnen zijn om in te loggen (ik heb al een naam en wachtwoord combinatie), maar als het spel een unieke code heeft, is deze code misschien ook voldoende om het spel af te schermen. Misschien eerst een opzet maken met login, ook omdat dat de meest leerzame opzet is…?

Home *startpagina, hier bepaal je of je een spel wilt opzetten, of wilt invullen, of wilt spelen*

Spel opzetten *deze elementen moeten in een ‘stepper’ worden gepresenteerd*

Deelnemers toevoegen *Wie is quiz master, wie is deelnemer (rol is belangrijk)*

Vragen toevoegen

Configuratie pagina…? *Deadline? Op jezelf stemmen? Inloggen?*

Quiz open zetten “*E-mails zijn verstuurd, veel succes!”*

Spel invullen *een groot formulier met alle vragen (wordt opgeslagen op veld verlaten)*

Spel spelen

home

setup

add-participants

add-questions

configuration

completed

rank

play

Hoe ziet de backend er uit, en welke endpoints bestaan daar?

Ik heb een game-controller, onder “/game”, waar ik een “/create” en een “/get/:id” en een “/get-all” kan aanroepen?

TODO

In Setup Component:

* De overgang tussen stappen mooier maken met een animatie (alleen bij openen een collapse van het component maken?)
* De formulieren afmaken
* Navigatie mogelijk maken afhankelijk van de validiteit van de formulieren
* Een edit-modus en view-modus van de formulieren maken (en een edit-knop toevoegen in inactive mode?)

**CORS**

Ik loop tegen problemen met CORS errors; de FE draait op localhost:4200, en de BE draait op localhost:3000, en dat levert CORS issues op. De error is te zien in de console van de browser tools, en niet in de network calls.

Ik dacht het probleem om te lossen met een NestJS instelling in main.ts:

**const** app = **await** NestFactory.create(AppModule, { **cors**: **true** });

Maar dit is niet echt een oplossing die ik graag in de applicatie wil toepassen, want de CORS regel bestaat voor een reden, en om deze compleet te negeren voelt niet goed.

Na het hoofdstuk van NestJS over Middleware doorgenomen te hebben, dacht ik een betere oplossing gevonden te hebben in de vorm van een CORS Middleware class, zoals in deze blog wordt beschreven:

<https://www.thecodecampus.de/blog/creating-cors-middleware-nest-js/>

Dit zou een mooie oplossing geweest zijn, als Angular niet zelf met een alternatieve oplossing kwam, en dat is om gebruik te maken van het proxy.conf.json bestand. Hierin geef ik aan dat alle “/api” calls moeten worden vervangen door “http://localhost:3000”, en daarmee treedt de CORS error niet meer op. Dus uiteindelijk is het probleem opgelost in de FE, en niet in de BE.

**2020-10-16**

Ik ben bezig met het opslaan van data naar de BE, en loop tegen verschillende vragen aan. Wil ik bij iedere stap een post call naar de BE doen? En is vanuit de FE iedere child-component verantwoordelijk voor deze call? Mijn eerste idee is dat dit wel het geval is, maar dan heb ik bijvoorbeeld voor de game-setup wel erg veel calls te maken: 3 stappen, waarbij eigenlijk voor iedere stap zowel een post (opslaan van data) als een get nodig is (bijvoorbeeld wanneer de quiz-master terugkomt om de quiz bij te werken). Met drie stappen en 2 calls per stap, zijn dat al 6 verschillende endpoints die ik moet aanroepen. Dat is zowel op de FE als de BE een hoop controllers en code…

Biedt GraphQL hier een gepaste oplossing, en zo ja, hoe zou deze opzet er dan uitzien? Ik geloof dat ik nog altijd een verschil moet maken tussen een post en een get, ofwel een mutation en een query. Alleen mutations, met geen data wanneer niets is ingevuld, zou volgens mij niet werken (ik ben namelijk bang dat een leeg object voor een eerste mutation, eventueel bestaande data zou kunnen overschrijven waardoor deze data verloren gaat; als dit niet het geval is, en een lege mutation verandert niets, zou dit een krachtige oplossing kunnen zijn). Maar zelfs als de mutation en query apart gedefinieerd moeten worden, is GraphQL voordeliger dan de traditionele aanpak. Omdat ik namelijk met de query kan bepalen welke gegevens ik wil ophalen (bijvoorbeeld alleen questions, of alleen participants), heb ik in totaal nog maar 2 endpoints nodig. Alle sub-onderdelen worden namelijk in hetzelfde object (ofwel endpoint) opgeslagen.