Verantwoordingsdocument

Naam: Nangialai Waziri

Datum: 14-11-2021

Opdracht: Full-stack

Inhoudsopgave

Inleiding

Tijdens de ontwikkeling van applicatie ben ik tegen verschillende zaken aangelopen waarin ik beslissingen heb genomen die zowel voor- en nadelen hebben. Om dit overzichtelijk weer te geven is dit document onderverdeeld in de volgende drie onderdelen:

1. Backend
2. Front-end
3. Limitaties

Backend

1. Functionaliteit

Tijdens de ontwikkeling van de backend heb ik direct keuzes moeten maken die implicatie heeft voor de architectuur van de gehele applicatie.

Een kern functionaliteit betreft het uploaden en converteren van audio files naar mp3 formaat. Het converteren van de audio naar mp3 kan in Java op verschillende manieren. Allereest is er de keuze tussen verschillende libraries, daarna is het van belang om te bepalen of de inkomende audio files in het geheugen geconverteerd worden of dat een .mp3 file wordt uitgeschreven. Na onderzoek gedaan te hebben naar de libraries heb ik de keuze gemaakt om te converteren met *Java Audio/Video Encoder (JAVE)*. Deze keuze heeft vooral te maken gehad met de complexiteit van de andere libraries zoals de Java Audio API waarin een grotere hoeveelheid kennis wordt van audio data wordt verwacht van de gebruiker. De JAVE voldoet aan de requirements van de functionaliteit die gebouwd moet worden.

Een beperking van JAVE is dat een audio file wordt uitgeschreven na het converteren, hierdoor ben ik afgeweken van mijn initiële design waarin ik de inkomende audio file in memory wilde converteren en de resulterende array aan bytes als blob op te slaan in de database. Hierdoor heb ik ervoor gekozen om een folder aan te maken waarin de inkomende audio file wordt opgeslagen, vervolgens wordt de audio file geconverteerd en in een andere folder bewaard en dus niet in de database. Via verschillende andere (libraries) kon ik uiteindelijk de inkomende file in memory converteren, echter werd hierdoor mijn code dusdanig complex dat het lastig was om de applicatie te onderhouden en door te ontwikkelen. Vaak begreep ik niet goed hoe bepaalde stukjes code snippets precies werkten en heb ik uiteindelijk gekozen om toch de JAVE te gebruiken. Een logische ontwikkeling voor een volgende versie van de applicatie zou dan ook zijn om de omzetting naar mp3 in memory te doen.

Doordat de geconverteerde file niet direct in de database wordt opgeslagen kan niet direct een record worden aangemaakt in de database. Hierdoor klopt dan ook de database model niet meer en heb ik ervoor gekozen om de geconverteerde mp3 file weer in te lezen en bepaalde data zoals de naam van de file, de gebruiker die de file heeft geüpload, etc. Dit had ik beter kunnen doen door de hierboven besproken in-memory conversie te doen of een andere manier te vinden hoe ik direct records in de database kon opslaan. Nu haal ik informatie op uit zowel de originele file die geüpload is en de geconverteerde en combineer ik beiden om een record in de database te maken. Dit zorgt voor onnodige code om data te combineren en is ook meer gevoelig voor bugs.

Een tweede kern functionaliteit van de gebouwde applicatie betreft het commentaar geven op de geüploade audio file. Door dat ik gekozen heb voor een stukje commentaar per file, ontstaat er een one-to-one relatie tussen file en commentaar. Dit betekent ook dat wanneer een file geüpload wordt direct een willekeurige string ingevoerd als commentaar om de record aan te maken in de commentaren tabel. Deze kan via een update aangepast worden in de database. Hier is niet gekozen voor een many-to-many relatie omdat het onoverzichtelijker is om de “nieuwste” stukje commentaar bij de juiste file te vinden.

Om een entry te maken in de commentaren tabel wordt direct na het uploaden van een file een stukje commentaar aangemaakt met een standaard tekst in plaats van wachten tot de eerste entry wordt aangemaakt nadat de admin deze een stukje commentaar geeft op de audio, dit zorgt ervoor dat de producer/gebruiker direct kan zien dat er nog geen feedback is op de demo.

Een laatste keuze die gemaakt is met betrekking tot de kern functionaliteiten van de applicatie is het feit dat de one-to-many relatie tussen User en MusicFile modellen niet in is geïmplementeerd in de eind oplossing. Een User kan meerdere files uploaden, echter na het aanmaken van deze relatie was de eerste entry (foreign key van de user in de music_files tabel) in de database altijd null. Het koste meer tijd dan verwacht om dit toch voor elkaar te krijgen. Door deze bug heb ik ervoor gekozen om de naam van de gebruiker vanuit de front-end mee te sturen naar de backend zodat in ieder geval bekend is welke user de file heeft geüpload. In een volgende versie zou wel een many-to-many relatie geïmplementeerd moeten worden in de databasemodel.

2. Configuratie

Bij het configureren van springboot zijn ook een aantal keuzes gemaakt die niet in lijn zijn met de demo`s en cursus materiaal die aangeboden is door de Peter Anema. In mijn front-end heb ik de ID velden van de music_files tabel en de comments tabel nodig gehad. Deze werden echter niet mee gestuurd waardoor een extra stukje configuratie is toegevoegd in de RepositoryConfig.java file. Hiermee heb ik de ID van de Comment model aan de response kunnen toevoegen. Omdat de Comment model een one-to-one relatie heeft met de musicFile model, zijn ook de ID van de music_files tabel toegevoegd aan de JSON response.

Tijdens het configuren van de CORS liep ik bij bepaalde requests (POST, PUT, DELETE en GET (downloaden van de file vanuit de front-end)) tegen pre-flight errors aan. Hierdoor heb ik voor de desbetreffende endpoints voor de OPTIONS request die voor de daadwerkelijk request uitgevoerd wordt opengezet. De configuraties zoals behandeld in de cursus hadden geen effect CORS issues. De daadwerkelijk requests zijn wel beveiligd.

3. Unit testing en mocking

Bij het implementeren van de unit tests en integratie tests voor de backend heb ik keuzes moeten maken bij het bepalen van welke componenten getest moeten worden. Doordat de focus lag op het stijlen van de front-end. De controllers, repositories, security en void functies van de services en utils zijn niet getest.

Controllers

Controllers zijn niet getest omdat in de controllers services worden aangeroepen. De services zijn wel getest om deze reden was de prioriteit om te focussen op het testen van de business logic (utils en services). Om deze reden en het feit dat deze functies alleen een response sturen zag ik niet veel toegevoegde waarde om de controllers te mocken.

Repositories

Repositories bevatten methods die al inbegrepen zijn in springboot. Echter, gebruik ik een aangepaste query in de musicFileRepository, deze worden gebruikt in de service en zijn ook in de service getest. Had ik meer tijd gehad voor de backend ontwikkeling dan waren deze wel apart getest.

Security

Security (JWT functionaliteit en algemene configuratie) is niet gemocked omdat deze heel abstract was binnen de cursus en de direct gebruik mochten maken van de code van Peter Anema. Echter, door te verbinden met de backend-end vanuit de front-end is security wel getest op zijn functionaliteit.

Void functies

Deze functies hebben geen return values, echter maken deze wel de grootste deel uit van deze applicatie. Omdat deze geen return values en meestal gebruikt worden binnen de services heb ik geen aparte unit tests geschreven. Echter bepaalde void functies zoals het opslaan van de demo in de database is in de service wel gemokt. Ook heeft het apart testen van deze functies weinig toegevoegde in deze fase van de applicatie. Een voorbeeld is een functie die twee folders maakt als deze niet bestaan. De vraag is hoeveel toegevoegde waarde een unit test heeft om nog twee extra folders te maken puur om het te kunnen testen.

Een laatste keuze met betrekken tot het unit testen en mocken betreft de annotaties die ik gebruikt hebt. Deze wijken af van de in cursus behandelde annotaties omdat deze dusdanig veel en verschillende configuratie errors gaven dat ik geblokt werd in de ontwikkeling van de applicatie. Vaak kon ik niet goed inschatten waar de errors vandaan kwamen. Door gebruik te maken van de @ExtendsWith, @Mock en @InjectMocks heb ik toch het zelfde kunnen bereiken.

Front-end

1. React

Met betrekking tot de front-end logica heb verschillende keuzes gemaakt waarvan sommige ook afweken van mijn initiële plan. Sommige van deze keuzes waren ook beïnvloed door keuzes die in de backend zijn gemaakt.

Unit tests

Unit tests zijn ontwikkeld voor de help functies. Echter na de installatie van jest, start de applicatie niet meer op door issue met de versies. Om deze reden in jest uit de package.json gehaald. Om toch de unit tests te kunnen draaien, moet jest on the fly worden geïnstalleerd. Dit moet zeker opgelost worden in de volgende versie.

React Componenten

In het begin van de front-end ontwikkeling koos ik ervoor om alles in React componenten te maken. Dus voor elke stukje code een aparte component te maken. Echter, waren er bij een aantal componenten issues waardoor de keuze heb gemaakt om er geen aparte componenten van te maken omdat ik teveel tijd kwijt was om het met componenten op te lossen. Het gaat hier specifiek om de SignUpForm en LoginForm waarin ik geen zelfde gemaakte input component gebruik. In eerste instantie had ik een aparte input component gemaakt waarin ik de code { …register()} gebruikte om de data in een object op te slaan. Echter, na het inladen van deze component in de form componenten en deze weer in te laden op de gerelateerde pagina`s kreeg ik verschillende soorten bugs die ik moest oplossen met refs. Ook na het toepassen van de refs kwam was het data object van beide formulieren leeg. Omdat deze formulieren alleen maar gebruikt worden om te registreren en om in te loggen. Heb ik besloten om deze zelfde gemaakte component niet te verwijderen en voor de formulieren de <input/> tag te gebruiken en daarin {…register()} op te nemen. Op deze manier bevatte het data object wel data. Verder bestaat alle andere herhalende code wel uit zelf gemaakte react componenten.

Gebruikerservaring

Voor de gebruikerservaring heb ik besloten om bij bepaalde acties error, loading messages weer te geven. Nadat de gebruiker een correctie actie uitvoert moeten deze messages natuurlijk verdwijnen uit de UI, echter heb ik dit niet voor de forms kunnen doen. Verder heb ik ook niet verplichte velden verplicht kunnen maken. Deze keuzes heb ik gemaakt om te kunnen focussen op andere functionaliteiten van de applicatie. Had ik meer tijd gehad voor het project dan had ik deze acties wel toegepast.

2. HTML en CSS

De verschillende keuzes die gemaakt zijn met betrekking tot styling zijn toepasbaar op de HTML en CSS onderdelen van de applicatie. Om deze reden worden ze samen besproken.

De eerste keuze die ik gemaakt heb is door de scope van de styling te bepalen. Ik koos ervoor om niet een hele complexe design te nemen. Dit omdat door gebrek aan ervaring met designs en het tijdsbestek van de applicatie ontwikkeling. Ik heb gekozen voor een simpele design met verschillende effecten zoals transities en hovers.

Na het ontwikkelen van de prototype heb ik besloten om een achtergrond te nemen die past bij de inhoud van de applicatie. De kleuren van de applicatie zijn zo gekozen dat ze passen bij de achtergrond de front-end pagina`s.

Sommige react componenten zijn direct in het component gestyled. Echter, andere componenten zijn op de pagina's gestyled omdat deze beter pasten bij de pagina zelf. Bijvoorbeeld error messages zijn binnen de component gestyled.

Limitaties

Deze ontwikkelde applicatie heeft verschillende limitaties. Hiermee wordt vooral bedoeld dat de applicatie nog niet gereed zou zijn om in productie te gebruiken.

Een eerste limitatie is de gebruikerservaring. Deze zou aangepast moeten worden in volgende versie om ervoor te zorgen dat gebruikers niet weglopen bij de website. Bijvoorbeeld door de styling professioneel te maken, meer terugkoppeling op de front-end geven aan de gebruikers, gebruik op andere apparaten, etc.

Een tweede limitatie is dat de applicatie niet kan schalen op een onderliggende infrastructuur. Er zijn geen stress testen uitgevoerd om te bepalen hoe goed de applicatie schaalt bij een toenemend gebruik.

Naast het feit dat het onbekend is hoe goed de applicatie kan schalen, is er tijdens de ontwikkeling geen rekening met variabelen die veel invloed hebben op de snelheid van de applicatie zoals content opslaan in cache en rekening houden met de locatie van gebruikers.

Verder moet ook rekening gehouden worden met het feit dat nog niet alle CRUD operaties voor de verschillende entiteiten aanwezig zijn. Hierdoor is het nog onbekend hoe de applicatie zich gedraagt als bepaalde entiteiten uit de verschillende tabellen worden verwijderd. Tijdens de doorontwikkeling van de applicatie heeft dit een zeer hoge prioriteit. Voorkomen moet worden dat wanneer relatief veel data is verzameld en nieuwe CRUD operaties worden ontwikkeld, de database model niet stabiel is.

Een andere limitatie heeft betrekking tot de hierboven geschreven CRUD operaties. Deze moeten voor de admin aangemaakt worden. Verder moet het ook mogelijk om nieuwe admin users te maken. Hier is nu nog geen gebruik van gemaakt.

De veiligheid van de applicatie is nog niet productie waardig. Een voorbeeld is dat verkeer

tussen de front-end en backend gaat via http. Certificaten moeten geïnstalleerd worden om een veilige kanaal op te zetten dat ervoor zorgt dat in de gehele communicatie encryptie wordt toegepast.

Hoewel de applicatie nog meer limitaties heeft, zijn de hierboven genoemde de belangrijkste volgende stappen voor het door ontwikkelen van de software.