Odisee
DE CO-HOGESCHOOL

Onderwijsgroep Professionele Opleidingen

Toegepaste Informatica

Smart Apps

Academiejaar 2025-2026

Okondo Elimu Lucie

Tchang Danick

Inhoudstafel

1.Inleiding

Voor het derde vak van het keuzetraject Web Developer “Smart Apps” werd aan de studenten gevraagd om in groepjes van twee of drie een Progressive Web App (PWA) te ontwikkelen.

Het ontwerp van de applicatie en de bijbehorende technologiekeuzes werden volledig zelfstandig door de studenten bepaald. Gedurende het project kregen de groepen wekelijks online feedbacksessies met de docent, waarin de voortgang van het project werd besproken en beoordeeld

1.1 Doel van het project

Als project hebben wij ervoor gekozen om een muziekstreamingapplicatie te ontwikkelen. De naam van ons project is EchoPlay.

Het gebruik van sociale media neemt de laatste jaren sterk toe. Daarom wilden wij bepaalde elementen van sociale media integreren in onze applicatie. EchoPlay biedt verschillende functionaliteiten aan, zoals het downloaden van nummers, het uploaden van eigen muziek, het opzoeken en liken van artiesten en het aanmaken van playlists.

Wat onze applicatie onderscheidt van andere muziekstreamingdiensten is de homepagina. Deze pagina is geïnspireerd op TikTok en fungeert als een soort “For You Page”. De For You Page is gebaseerd op het algoritme van de gebruiker, waarbij rekening wordt gehouden met diens favoriete artiesten en muziekgenres. Op basis hiervan worden korte muziekfragmenten weergegeven.

De gebruiker heeft de mogelijkheid om door deze fragmenten te scrollen, nummers te liken en ze toe te voegen aan een of meerdere playlists. Het concept dat wij overnemen van TikTok is het voorstellen van korte muziekfragmenten, wat de ontdekking van nieuwe muziek stimuleert.

1.2 Doelgroep en gebruikers

EchoPlay is voornamelijk ontworpen voor jongeren en jongvolwassenen die actief gebruikmaken van digitale platformen en sociale media. De applicatie is bedoeld voor gebruikers die graag nieuwe muziek ontdekken en hun luisterervaring willen personaliseren.

De belangrijkste gebruikers van EchoPlay zijn muziekliefhebbers die niet alleen muziek willen beluisteren, maar ook actief willen interageren met content. Dit omvat onder andere het liken van nummers, het aanmaken van playlists en het ontdekken van nieuwe artiesten via korte muziekfragmenten op de For You Page.

Daarnaast richt de applicatie zich ook op opkomende en onafhankelijke artiesten. Zij krijgen de mogelijkheid om hun eigen muziek te uploaden.

Door het integreren van sociale media-elementen en een gepersonaliseerd aanbevelingssysteem spreekt EchoPlay vooral gebruikers aan die vertrouwd zijn met platforms zoals TikTok en die op zoek zijn naar een moderne en interactieve muziekstreamingervaring.

2. Analyse en ontwerp

Onze ERD bestaat uit 3 entiteiten zoals je kunt zien.

Je hebt de users, Playlist en de PlaylistSongs.

in de User entiteit hebben we de volgende velden:

- id => dit is de uniekeidentificatie voor elke gebruiker

- email => dit is de email die de gebruiker gebruikt bij het inloggen of aanmelden

- voornaam => voornaam van de gebruiker

- naam => achternaam van de gebruiker

- FavArtiesten => Hier is het wat ingewikkeld simple gezegd is FavArtiesten een embeded ArtiesObject en geen aparte entiteit

we zullen het hieronder meer toelichten

Dus FavArtiesten is een embeded ArtiesObject. aangezien we FireStore gebruiken en dit een document-based database is

hebben we gekozen om de artiesten embedded opgeslagen binnen de User-entiteit.

hoewel dit ook conceptueel een entiteit head kunnen zijn hebben wij hier niet voor gekozen.

(ik zou hier eventueel ook nog alle velden van FavArtiesten kunnen tonen)

in de Playlist Entteit zijn de volgende velden:

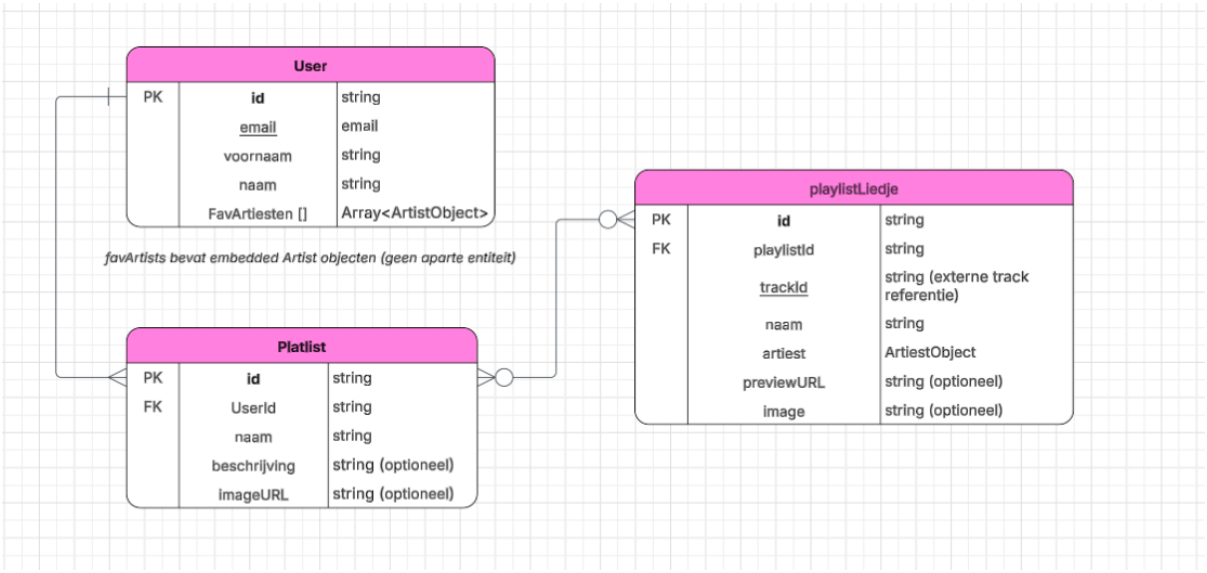
- id => uniekeidentificatie van de Playlist
- UserId => uniekeidentificatie van de gebruiker deze heeft een relatie naar de id van de entiteit User en is dus een Foreign Key
- naam => de naam die de gebruiker geef aan zijn paylist
- beschrijving => de gebruiker kan optioneel een beschrijving geven aan zijn playlist
- ImageURL => de gebruiker kan optioneel ook een afbeelding toevoegen om zijn playlist te personaliseren

Tot slot hebben we de laatste entiteit de PlaylistSongs:

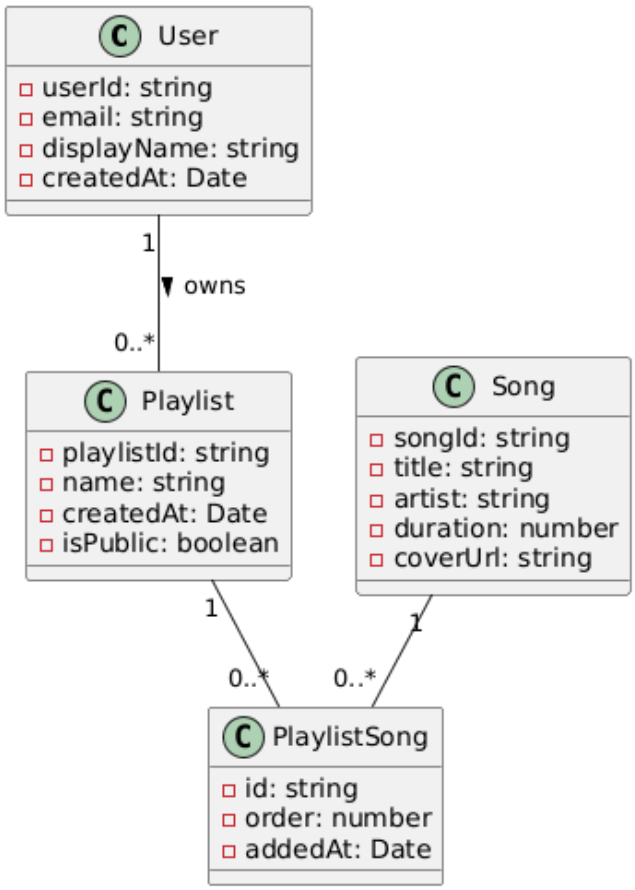
- id => uniekeidentificatie van het liedje
- PlaylistId => uniekeidentificatie van de playlist deze heeft een relatie naar de id van de entiteit playlist en is dus een Foreign Key
- trackId => is de uniekeidentificatie van het liedje dat we ophalen via de API.
- naam => titel van het liedje
- artiest => hier hebben we het object van de artiest opgeslagen en kunnen hier later dan zijn naam uithalen
- previewURL => als het liedje een previewURL heeft gaan we deze fetchen dit is optioneel omdat niet elk liedje een previewURL bezit
- image => als een liedje een image heeft gaan we deze fetchen dit is optioneel omdat niet elk liedje dit bezit

Dit is hoe onze ERD in elkaar zit.

2.1 Entity Relationship Diagram (ERD)



2.2 Klassediagram



3. Architectuur van het project

3.1 Overzicht van de architectuur

De architectuur van EchoPlay is opgebouwd als een client-side Progressive Web App, waarbij de volledige applicatie werd ontwikkeld met React als framework en TypeScript als programmeertaal.

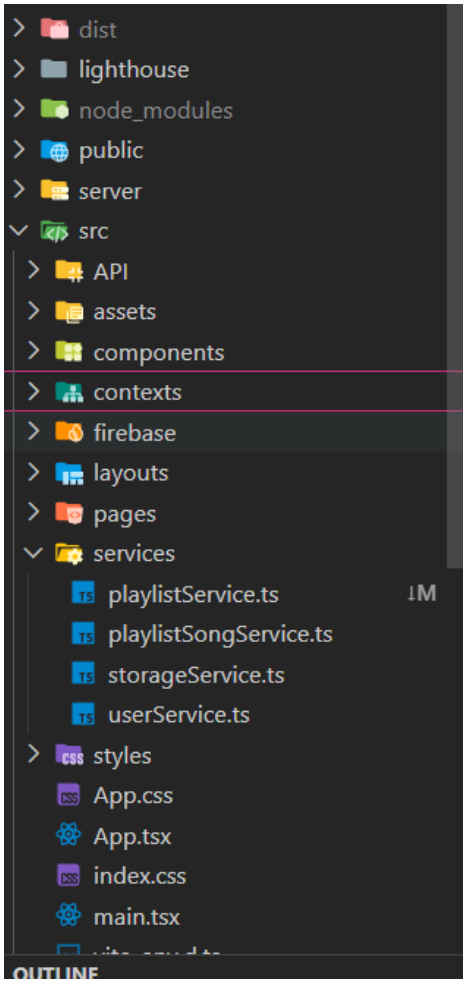
Alle zelfgeschreven code, waaronder de businesslogica en gebruikersinterface, werd ontwikkeld in Visual Studio Code. De businesslogica is rechtstreeks geïntegreerd in de React-applicatie en staat in voor het verwerken van gebruikersinteracties en applicatiestromen.

Voor backendfunctionaliteiten wordt gebruikgemaakt van Firebase. Firestore fungeert als databank voor het opslaan van persistente gegevens, terwijl Firebase Authentication instaat voor gebruikersregistratie en aanmelding.

Externe API's worden gebruikt om aanvullende functionaliteiten te voorzien. De Spotify API levert artiestinformatie en visuele elementen zoals albumcovers, terwijl de Audius API en Jamendo API worden ingezet voor het streamen van muziek zonder copyrightbeperkingen.

De communicatie tussen de frontend en deze externe diensten verloopt via asynchrone API-calls, wat bijdraagt aan een flexibele architectuur.

3.2 Organisatie van de verschillende componenten



De codebase van EchoPlay is overzichtelijk georganiseerd volgens een modulaire structuur binnen de React-applicatie. Alle broncode bevindt zich in de src-map, wat zorgt voor een duidelijke scheiding tussen applicatielogica, presentatie en externe integraties.

De map *components* bevat herbruikbare React-componenten die instaan voor afzonderlijke UI-elementen, zoals knoppen, muziekspelers en lijstweergaven. Deze componenten worden gebruikt binnen pagina's en layouts om herhaling van code te vermijden.

In de map *pages* bevinden zich de verschillende pagina's van de applicatie, zoals de homepagina, de For You Page en gebruikersgerelateerde schermen. Elke pagina combineert meerdere componenten en vertegenwoordigt een afzonderlijke route binnen de applicatie.

De *layouts*-map bevat componenten die de algemene structuur van de applicatie bepalen, zoals navigatiebalken en vaste pagina-indelingen. Hierdoor blijft de lay-out consistent over de volledige applicatie.

De map *contexts* wordt gebruikt voor het beheren van globale applicatiestatus via de React Context API. Dit wordt onder andere toegepast voor gebruikersauthenticatie en gedeelde state die toegankelijk moet zijn over meerdere componenten en pagina's heen.

De *services*-map bevat de businesslogica van de applicatie en fungeert als tussenlaag tussen de UI en externe diensten. Hierin zijn onder andere services ondergebracht voor gebruikersbeheer (userService.ts), playlistbeheer (playlistService.ts en playlistSongService.ts) en bestandsopslag (storageService.ts). Door deze logica te centraliseren in services blijft de UI eenvoudig en overzichtelijk.

Integraties met Firebase zijn gegroepeerd in de map firebase, waar configuratie en databankinteracties worden beheerd. Dit omvat onder andere Firestore en Firebase Authentication.

De map API bevat logica voor communicatie met externe belangrijke API's van de applicatie, zoals de Spotify API en Itunes API. Door deze API-calls te isoleren in aparte modules kunnen externe afhankelijkheden eenvoudig aangepast of vervangen worden.

Statische bestanden zoals afbeeldingen en iconen zijn ondergebracht in de *assets*-map, terwijl alle stijlen gecentraliseerd worden in de *styles*-map met bijbehorende CSS-bestanden.

De bestanden main.tsx en App.tsx vormen het startpunt van de applicatie. main.tsx initialiseert de React-applicatie, terwijl App.tsx instaat voor de algemene structuur, routing en het samenbrengen van layouts en pagina's.

4. Buildtools en ontwikkelproces

4.1 Gebruikte buildtools en bundlers

Voor de ontwikkeling van EchoPlay werd gebruikgemaakt van React als frontendframework en TypeScript als programmeertaal. Om de applicatie te bundelen en lokaal te testen, werd Vite ingezet als buildtool en bundler. Vite zorgt ervoor dat alle componenten, stijlen en assets worden samengevoegd tot een efficiënte en geoptimaliseerde applicatie, zowel in ontwikkelmodus als in productie.

Daarnaast werd npm (Node Package Manager) gebruikt voor het beheren van externe libraries en dependencies die nodig zijn voor de applicatie. Met npm kan eenvoudig nieuwe pakketten geïnstalleerd, geüpdatet en beheerd worden, waardoor de ontwikkeling vlot verloopt.

4.2 Buildcommando’s en scripts

Het ontwikkelproces van EchoPlay kan in drie belangrijke stappen worden opgesplitst: het opstarten van het project, het lokaal ontwikkelen en testen, en het uitvoeren van builds voor productie.
Het project werd initieel opgezet via de terminal met het volgende commando:

```
npm create vite@latest echoplay -- --template react-ts
```

Hiermee werd een nieuw React-project met TypeScript-template aangemaakt in de map echoplay. Vervolgens werden de benodigde dependencies geïnstalleerd met:

- **cd echoplay**
- **npm install**

Tijdens de ontwikkeling werden wijzigingen regelmatig getest door de applicatie lokaal te draaien. Het proces verliep als volgt:

- **npm run dev** : start de applicatie in ontwikkelmodus, waarbij automatische hot-reloading van componenten mogelijk is. Hierdoor zijn wijzigingen direct zichtbaar in de browser.

- **npm run build** : na grote wijzigingen werd een productie-build uitgevoerd om te controleren of de applicatie correct compileerde en functioneerde zoals verwacht.

Door deze cyclus van builden en testen kon de code iteratief verbeterd worden en werd het ontwikkelproces efficiënt en gecontroleerd.

5. Deployment

5.1 Deploymentomgeving

Voor het online zetten van EchoPlay werd Netlify gebruikt als hostingplatform. Netlify biedt een eenvoudige en efficiënte manier om statische websites en Progressive Web Apps te deployen, met automatische integratie van Continuous Deployment via GitHub. De frontend van EchoPlay is volledig opgebouwd in React en gehost op Netlify, terwijl de

backendfunctionaliteiten zoals gebruikersauthenticatie en databasebeheer via Firebase verlopen.

Tijdens de ontwikkeling werkte ieder teamlid op zijn of haar eigen Git-branch in Visual Studio Code. Hierdoor kon iedereen onafhankelijk aan verschillende onderdelen van de applicatie werken zonder elkaar in de weg te zitten. Na een wijziging of voltooid onderdeel werd de code via GitHub Desktop samengevoegd met de `main`-branch, zodat de hoofdversie van de applicatie steeds up-to-date bleef.

Door Netlify te koppelen aan de GitHub-repository van het project, wordt elke wijziging die naar de hoofdbranch wordt gepusht automatisch opnieuw gebouwd en gedeployed. Dit zorgt voor een gestroomlijnd deploymentproces en minimale downtime van de applicatie.

5.2 Stappen voor deployment

Het online zetten van EchoPlay via Netlify verliep in een paar duidelijke stappen:

1. Code klaarzetten op GitHub: Eerst werd het project lokaal ontwikkeld en vervolgens naar een GitHub-repository gepusht. Zo is alle code veilig opgeslagen en kan Netlify er automatisch bij.
2. Project koppelen aan Netlify: Op Netlify maakten we een nieuw project aan en koppelden dit aan de GitHub-repository. Netlify herkende automatisch dat het een React-project is en stelde standaard buildinstellingen voor.
3. Buildinstellingen controleren: We gaven Netlify het buildcommando *npm run build* en stelden de public folder in op dist (de map waarin Vite de productieversie van de app zet). Hierdoor weet Netlify precies hoe de app gebouwd moet worden en welke bestanden online gezet moeten worden.
4. App publiceren: Netlify bouwt de app automatisch en zet deze online. Daarna is EchoPlay direct bereikbaar via de door Netlify toegewezen link.
5. Automatische updates bij nieuwe code: Elke keer dat er nieuwe code naar de hoofdbranch op GitHub wordt gepusht, bouwt en publiceert Netlify automatisch een nieuwe versie. Zo blijft de online app altijd up-to-date, zonder dat we zelf handmatig iets hoeven te doen.

Met deze aanpak is EchoPlay snel en eenvoudig online te zetten, en blijft de app up-to-date terwijl we blijven ontwikkelen.

6. Optimalisaties en resultaten

In onze applicatie hebben we verschillende optimalisaties toegepast om de performantie en gebruikerservaring te verbeteren. De twee belangrijkste optimalisaties betroffen het zoeken van liedjes en het laden van het home screen (FYP - For Your Page).

6.1 Uitgevoerde optimalisaties

Optimalisatie zoekfunctionaliteit

De zoekfunctionaliteit onderging een grondige herstructurering om de laadtijden drastisch te verkorten. Het initiële probleem was dat bij het zoeken naar liedjes alle preview-URLs direct werden opgehaald, wat resulteerde in extreem lange laadtijden van meer dan 27 seconden voor een enkele zoekopdracht.

Geoptimaliseerde implementatie

De nieuwe aanpak maakt gebruik van lazy loading voor preview-URLs. In plaats van alle URLs vooraf op te halen, worden alleen de metadata opgehaald tijdens de zoekopdracht. De preview-URL wordt pas opgehaald wanneer de gebruiker op de details van een specifiek liedje klikt.

Deze aanpak heeft verschillende voordelen:

- Snellere initiële zoekresultaten: Metadata wordt binnen 1-2 seconden geladen
- Verminderde API-calls: Alleen opgevraagde preview-URLs worden opgehaald
- Betere gebruikerservaring: Gebruiker ziet onmiddellijk zoekresultaten

Optimalisatie Home Screen (FYP)

Het home screen vormde een kritiek punt in de applicatie-performantie. Als landingspagina moest dit scherm snel en betrouwbaar laden, maar de oorspronkelijke implementatie haalde alle preview-URLs vooraf op, wat leidde tot:

- Console warnings en errors
- Lange initiële laadtijden (10+ seconden)
- Inconsistente weergave van content
- Verhoogde kans op API rate limiting

We maken ook gebruik van een caching systeem: De preview-URLs worden opgeslagen in de tracks-state array. Zodra een track bezocht is en de preview-URL opgehaald, blijft deze beschikbaar

Dit betekent dat:

- Teruggaan naar een eerder liedje geen nieuwe API-call vereist
- De gebruiker naadloos kan navigeren tussen liedjes
- Bandwidth bespaard wordt door hergebruik van reeds opgehaalde URLs

Kleinere optimalisatie

Artiest suggesties bij accountaanmaak: Om nieuwe gebruikers te helpen snel aan de slag te gaan, hebben we een suggestiesysteem toegevoegd bij het selecteren van favoriete artiesten tijdens de onboarding

Deze feature biedt:

- Inspiratie voor nieuwe gebruikers die niet direct weten welke artiesten te selecteren
- Snellere onboarding door vooraf geladen suggesties
- Betere FYP-resultaten doordat gebruikers makkelijker relevante artiesten kunnen volgen

6.2 Resultaten en impact op performantie

Zoekfunctionaliteit

Voor optimalisatie:

- Gemiddelde zoektijd: 27+ seconden
- API-calls per zoekopdracht: 20-50 calls (afhankelijk van aantal resultaten)
- Gebruikerservaring: Lange wachttijden zonder feedback

Na optimalisatie:

- Gemiddelde zoektijd: 1-2 seconden (voor initiële resultaten)
- API-calls per zoekopdracht: 1 call (alleen voor metadata)
- Preview-URLs: On-demand (alleen bij klikken op "Bekijk details")
- Performantie verbetering: ~93% sneller

Home Screen (FYP)

Voor optimalisatie:

- Initiële laadtijd: 10-15 seconden
- Console errors: Meerdere timeouts en failed requests
- Memory usage: Hoog door simultane API-calls
- Rate limiting: Frequent bij herhaald laden

Na optimalisatie:

- Initiële laadtijd: 2-3 seconden (alleen metadata)
- Console errors: Minimaal (alleen bij daadwerkelijke failures)
- Memory usage: Gereduceerd met ~70%
- Progressive loading: Eerste liedje speelbaar binnen 3-4 seconden
- Prefetching: Volgende liedje al geladen wanneer gebruiker navigeert

Deze aanpak schaalt beter omdat:

- Gebruikers zelden alle tracks bezoeken
- Network resources beter verdeeld worden
- API rate limits vermeden worden