Haaga-Helia
ammattikorkeakoulu Oy

Soittotapahtumien raportointisovellus radiokanavalle

Teemu Kostamo

Opinnäytetyö
Tietojenkäsittelyn
koulutusohjelma
2021

Tiivistelmä

Haaga-Helia ammattikorkeakoulu Oy

Tekijä(t)
Teemu Kostamo

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Raportin/Opinnäytetyön nimi
Soittotapahtumien raportointisovellus radiokanavalle

Sivu- ja liitesivumäärä
52

Opinnäytetyönä kehitetään soittotapahtumien raportointisovellus radiokanavalle. Moderneilla web-teknologioilla toteutettu uusi sovellus vähentää radiokanavan henkilökunnan soittotapahtumien raportointiin käytettävää aikaa automatisoimalla datansyöttöä.

Johdanto-osuudessa kerrotaan miksi kyseinen sovellus on tarpeen ja miten se helpottaa kanavan työntekijöiden arkea. Käytettävyys-luvussa kerrotaan käyttäjälähtöisen suunnittelun teoriasta ja sovelluksen vaatimusmäärittelyssä käytetyistä tutkimusmenetelmistä. Teknologiaosuudessa esitellään sovelluksessa käytetyt teknologiat, sekä avataan lyhyesti syitä miksi kyseiset teknologiat valikoituvat käyttöön. Sovelluksen toiminta ja rakenne –osuudessa esitellään miten sovellusta käytetään, sekä millainen rakenne sovelluksella on. Viimeisissä luvuissa kerrotaan sovelluksen testauksesta, TypeScript-refaktoroinnista, sekä analysoidaan kehitysprosessin onnistumista ja sovelluksen mahdollista jatkokehitystä.

Opinnäytetyön tuloksena syntyy tuotantovalmis soittotapahtumien raportointisovellus radiokanavalle.

Asiasanat
radio, React, Node.js, TypeScript, tekijänoikeus

Sisällys

1 Johdanto 1
1.1 Vastaavien sovellusten markkinatilanne 2
1.2 Käsitteet 2
2 Käyttäjälähtöinen suunnittelu & käytettävyys 5
2.1 Vaatimusmäärittelyn tutkimusmenetelmä 6
3 Teknologiat 8
3.1 Yleiset 8
3.1.1 Javascript 8
3.1.2 Tekstieditori, koodin formatointityökalut ja selainten kehittäjätyökalut 8
3.1.3 Git & Github 9
3.2 Frontend 10
3.2.1 React 10
3.2.2 Redux 12
3.3 Backend 13
3.3.1 Node.js & Express 13
3.3.2 REST & Postman 14
3.3.3 Tietokanta 15
4 Sovelluksen toiminta, rakenne & toteutus 18
4.1 Käyttöliittymä 18
4.1.1 Etusivu, navigointipalkki ja uuden raportin luominen 18
4.1.2 Kappaleiden lisääminen raporttiin ja Raportit -sivu 19
4.1.3 Haku ja Top 100 23
4.1.4 Ohjelmat ja Käyttäjät 24
4.1.5 Omat tiedot, uloskirjautuminen ja aktiivinen raportti 25
4.2 Käyttäjätasot 26
4.3 Projektin kansiorakenne 27
4.4 Backend 28
4.5 Frontend 32
4.6 Redux tilanhallinta 34
4.7 Ulkoiset tietolähteet – DJOnline, Discogs 36
4.8 Kuukausiraportin tulostus 37
4.9 Julkaisu 38
5 Testaus 39
5.1 Yksikkö- & Integraatiotestit 39
5.2 Käyttöliittymän testaaminen 42
6 Sovelluksen TypeScript-versio 44
7 Pohdinta, aikataulun toteutuminen ja jatkokehitys 47

8 Lähteet..49

1 Johdanto

Opinnäytetyönä tuotetaan kappaletietojen raportointisovellus radiokanavalle. Kyseessä on moderneilla web-teknologioilla tuotettu sovellus, jota kanavan ohjelmantekijät käyttävät ohjelmissaan soittamansa musiikin raportointiin. Raportit toimitetaan muusikkojen oikeuksia Suomessa edustaville tekijänoikeusjärjestöille Teostoon ja Gramexiin, jotka tilittävät asiakkailleen korvauksen radiossa soineesta musiikista raportoitujen soittotapahtumien perusteella. Raportointisovelluksesta haetaan myös kappaletiedot radiokanavan verkkosivuille, josta kuulijat voivat halutessaan nähdä taajuudella kuulemansa kappaleet listattuna.

Sovelluksen toiminta pähkinänkuoressa on, että kanavan ohjelmantekijä aloittaa sovelluksen etusivulla uuden raportin täyttämisen. Ensin syötetään ohjelman nimi, päivämäärä ja kellonaika, jonka jälkeen syötetään ohjelmassa soineet kappleet. Käyttäjän on mahdollista hakea soittotapahtumia studion playout-ohjelmiston soittolokista, tai kappaletietokannasta. Mikäli soitettua kappaletta ei löydy tietokannasta tai soittolokista, kappaleen tiedot syötetään lomakkeeseen ja tallennetaan sekä raporttiin että tietokantaan. Kun ohjelman kaikki kappaleet on syötetty raporttiin, raportti kuitataan valmiiksi.

Inspiraationa sovellukselle toimi radiokanavalla aikaisemmin käytetty raportointityökalu. Vanhassa järjestelmässä ohjelmantekijät syöttivät kaikki soittamansa kappaleet raportteihin käsin. Musiikin digitalisaation myötä suuri osa kanavalla soivasta musiikista soitetaan studion playout-ohjelmasta, jonka soittotapahtumat on mahdollista hakea soittolokista. Uuden raportointijärjestelmän myötä manuaaliseen kappaletietojen syöttämiseen ei tarvitse juuri käyttää aikaa, sillä kappaletiedot saa haettua soittolokista silmän räpäyksessä. Käsin on enää tarpeen syöttää ainoastaan kappaleet, jotka on soitettu muualta kuin playout-ohjelmasta, kuten esimerkiksi cd- tai vinyylilevyiltä tai digitaalisina musiikkitiedostoina.

Opinnäytetyön tuloksena syntyy täysin tuotantovalmis kappaletietojen raportointisovellus, joka täyttää tekijänoikeusjärjestö Teoston ja Gramexin raportoinnille asettamat vaatimukset. Uudessa sovelluksessa käytetään vanhan raportointisovelluksen tietokannan kopiota, joten kanavan soittohistoria aina vuodesta 2001 alkaen on sovelluksen käyttäjien selattavissa.

Opinnäytetyössä käydään ensin läpi sovellukselle asetetut vaatimukset ja esitellään käyttäjälähtöisen suunnittelun teoriaa. Teknologiaosuudessa esitellään sovelluksen toteutukseen käytetyt ohjelmointikielet, kirjastot ja palvelut. Sovelluksen toteutus ja toiminta – osuudessa esitellään sovelluksen toteutus, rakenne ja toiminta, sekä kehitys- & julkaisu-

käytännöt. Lopuksi analysoidaan opinnäytetyön tuloksia ja pohditaan sovelluksen jatkokehitysmahdollisuuksia.

1.1 Vastaavien sovellusten markkinatilanne

Sovellusta on hankala lokeroida suoraan johonkin tiettyyn tuotekategoriaan. Toisaalta kyse on sisäiseen raportointiin käytettävä työkalu, toisaalta ulkoiseen, laissa määriteltyyn raportointiin käytettävä työkalu. Toimeksiantajan tapauksessa yhteen sovellukseen oli tarpeen yhdistää vanha tietokanta joka pitää sisällään soittotapahtumat kahdenkymmenen vuoden ajalta ja mahdollisuus hakea dataa moderneista REST-rajapinnoista.

Sovellus on ainoa laatuaan, eikä sille löydy suoranaisia kilpailijoita Suomen markkinoilta. Kaikki sovelluksessa käytettävät teknologiat ovat ohjelmistokehityksen valtavirtaa, eikä mikään sovellukseen toteutetuista ominaisuuksista poikkea kohtuuttomasti perinteisen luo-lue-päivitä-poista –sovelluksen toiminnasta. Ainutlaatuisuus tulee siitä, että sovelluksen ominaisuudet on täysin kustomoitu vastaamaan toimeksiantajan tarpeita.

Kappaletietojaan tekijänoikeusjärjestöille raportoivat radiokanavat ovat melko marginaalinen markkina. Kaikki radiokanavat pitävät kirjaa soittotapahtumistaan valitsemallaan tavalla. Ainoa vaatimus tekijänoikeusjärjestöjen taholta on kuukasittaisten soittotapahtumien raportointi vaaditussa formaatissa.

1.2 Käsitteet

Ohjelmaraportti	Raportti, johon on listattu kaikki yksittäisessä ohjelmassa soineet kappaleet.
Kuukausiraportti	Tekijänoikeusjärjestöille lähetetty raportti, johon on koottu yhden kuukauden kaikki ohjelmaraportit tekijänoikeusjärjestöjen vaatimassa raportointiformaatissa.
Playout-ohjelma	Radiokanavan studioiden tietokoneille asennettu ohjelmisto, josta suuri osa kanavalla soivasta digitaalisesta musiikista soitetaan. Playout-ohjelmaa käytetään myös muun ääniaineiston, kuten mainosten ja inserttien soittamiseen.
Soittoloki	Playout-ohjelman API endpoint, josta voi hakea tiedot ohjelman kautta soitetuista kappaleista. Endpoint-urlin parametrina annetaan haluttu päivämäärä, sekä studio jonka soittotapahtumia haetaan. Soittoloki palauttaa soittotapahtumat JSON-muodossa.

ISRC	ISRC (lyhenne sanoista International Standard Recording Code) on äänitteiden ja musiikkivideoiden tunnistamisessa käytetty koodi. ISRC -koodi määritetään ISO-standardissa ISO 3901. ISO on antanut koodien jakelun kansainvälisen äänitetuottajien järjestön IFPIn tehtäväksi. Suomessa ISRC -koodien antamisesta vastaa Ääni- ja Kuvatallennetuottajat ÄKT ry. ISRC -koodi annetaan erikseen kullekin teoksesta julkaistulle tallenteelle. Näin jokaisella teoksesta julkaistulla versiolla on oma koodinsa. Luokittelu luo perustan kansainväliselle tallenteiden EDI-rekisteröinnille. Sillä on tarkoitus helpottaa paitsi kirjastojen ja arkistojen luettelointia, myös teosten tunnistamista tekijänoikeusraportoinnin yhteydessä. ISRC ei kuitenkaan yksilöi itse teosta, joten se toimisi raportoinnissa lähinnä viiteavaimena teokset yksilöivään tietokantaan.
Levymerkki	Levymerkki eli label tai levy-yhtiö on musiikkia julkaiseva yritys. Levymerkki on usein vastuussa julkaisemansa musiikin tuotannosta, jakelusta, markkinoinnista ja promootiosta. Yhdellä levy-yhtiöllä voi olla allaan useita eri levymerkkejä.
Levykoodi	Tuottajan tai levymerkin julkaisemalleen äänitteelle antama tunnus. Tavallisesti koodi on kirjain-numeroyhdistelmä, jossa kirjaimet ovat levymerkin nimessä esiintyviä kirjaimia ja numero kertoo monesko levymerkin julkaisu on kyseessä. Tekijänoikeusjärjestöt toivovat että koodi on mahdollisimman omaperäinen ja pituudeltaan enintään 19 merkkiä. Levykoodi on koko äänitteen koodi, toisin kuin ISRC-koodi joka on kappalekohtainen.
Discogs	Suurin internetistä löytyvä käyttäjiensä ylläpitämä äänitetietokanta. Discogsin APIsta on mahdollista hakea julkaisujen metatietoja raportointia varten.
Spotify	Miljoonia musiikkikappaleita sisältävä digitaalinen streamauspalvelu. Spotifyn APIsta on mahdollista hakea julkaisujen metatietoja raportointia varten.
Tekijänoikeusjärjestö	Musiikin tekijänoikeuksia valvovat tahot. Radiokanavien raportoinnin kannalta oleellisia tahoja Suomessa ovat Teosto ja Gramex
Teosto	Säveltäjäin Tekijänoikeustoimisto Teosto r.y. on säveltäjien, sanoittajien, sovittajien ja musiikinkustantajien aatteellinen ja voittoa tavoittelematon tekijänoikeusjärjestö. Teoston tehtävä-

nä on kerätä ja tilittää musiikintekijöille ja kustantajille korvaukset heidän musiikkinsa julkisesta esittämisestä ja tallentamisesta. Järjestö huolehtii myös, että musiikin käyttäjät saavat helposti luvat musiikin esittämiseen ja tallentamiseen.

Gramex

Gramex ry on muusikoiden, solistien, kapellimestarien ja äänitteiden tuottajien oikeuksien valvomiseksi perustettu tekijänoikeusjärjestö. Se valvoo oikeuksia, joista säädetään tekijänoikeuslainsäädännössä. Yhdistys kerää musiikin esittämisestä ja muusta kuin yksityiskäyttöön kopioinnista korvauksia. Kerätyt varat Gramex välittää asiakkailleen muun muassa musiikin radiosoiton perusteella.

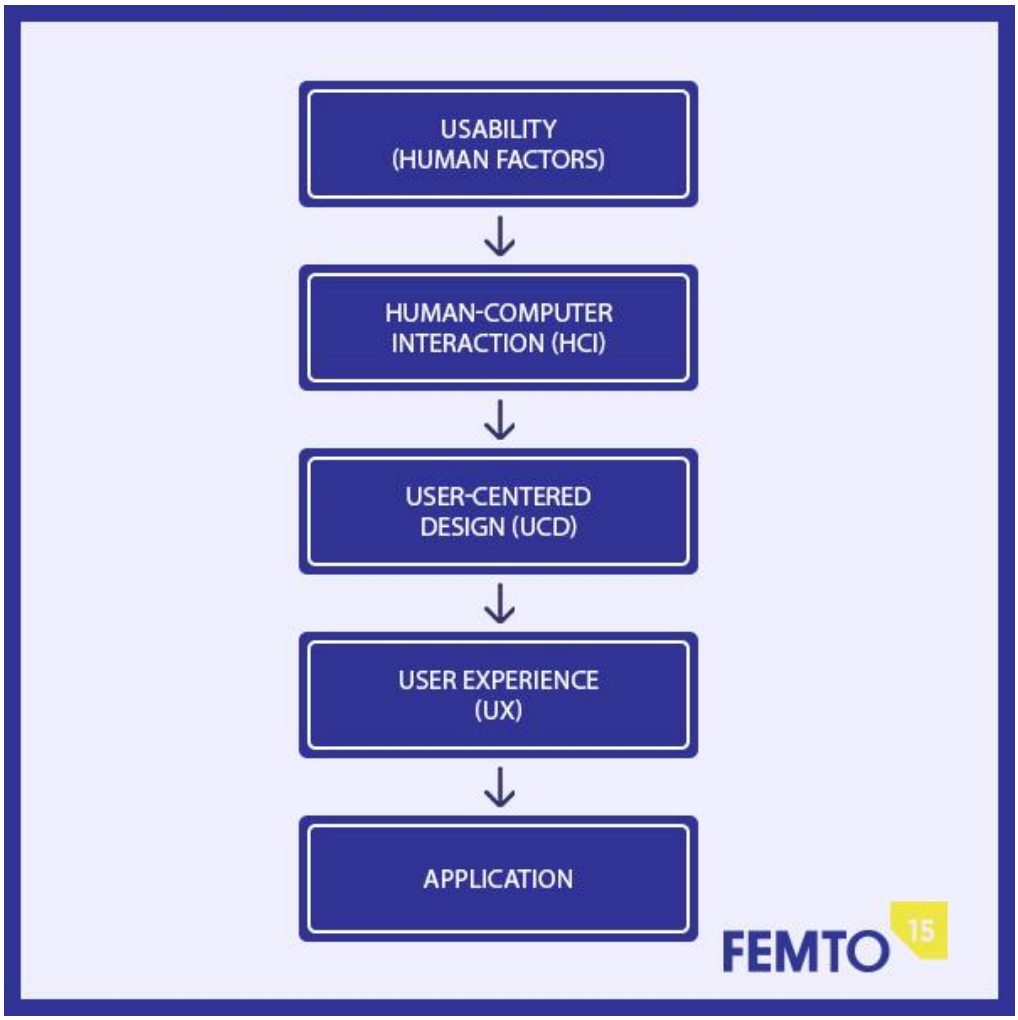
2 Käyttäjälähtöinen suunnittelu & käytettävyys

Hyvä käytettävyys voi olla näkymätöntä, mutta hyvän käytettävyyden puute ei missään nimessä ole. Joidenkin ohjelmistokehittäjien keskuudessa on havaittavissa ajatus, että hyvä käytettävyys on subjektiivinen kokemus. Näiden kehittäjien mielestä käytettävyysvalinnat ovat mielivaltaisia, kehittäjän omiin mieltymyksiin perustuvia. Tämän lisäksi joskus tehdään käytettävyysvalintoja, joilla ei ole mitään tekemistä käyttäjien kanssa.

Käyttäjälähtöinen suunnittelu ei ole sama asia kuin käytettävyys. Käytettävyys, johon viitataan myös inhimillisinä tekijöinä, on tutkimusta siitä, miten ihmiset toimivat suhteessa tuotteeseen. Käytettävyystutkimusta ja siitä saatuja oppeja voidaan soveltaa vaikka leivänpaahtimeen tai ovenkahvaan, tai niiden myyntipakkauksiin.

Ihmis-tietokonevuorovaikutustutkimuksen (englanniksi HCI, human-computer interaction) juuret on käytettävyystutkimuksessa, mutta se keskittyy enemmän siihen, miten ihmiset toimivat suhteessa tietokone- ja ohjelmistotuotteisiin. Käyttäjälähtöinen suunnittelu on HCI:n sivujuonne, ohjelmistosuunnittelun metodologia ohjelmistokehittäjille ja suunnittelijoille. Käyttäjälähtöisen suunnittelun on tarkoitus auttaa kehittäjiä tekemään sovelluksia, jotka kohtaavat käyttäjiensä tarpeet. Asettamalla käyttäjät suunnitteluprosessin keskiöön prosessiin saadaan selkeyttä ja suuntaa.

Käyttäjäkokemus (UX – user experience) puolestaan on termi, jota usein käytetään summaamaan kokemus koko ohjelmistotuotteesta. Käyttäjäkokemus ei rajoitu pelkästään ohjelmiston ulkonäköön ja toiminnallisuuteen, vaan pitää sisällään sen, kuinka mukaansatempaavaa ja miellyttävää ohjelmiston käyttäminen on. Sovelluksen käyttäjäkokemus on suurempi kuin osiensa summa, sisältäen ohjelmiston käyttämisen käyttäjässään herättävät fyysiset ja psyykkiset reaktiot. Soveltamalla käyttäjälähtöistä suunnittelua oikein voidaan varmistaa, että sovelluksen käyttäjäkokemus pysyy hyvänä. §
(Lowdermilk, Travis, User-Centered Design.)



Kuva 1. Käytettävyyden (Usability), ihmis-tietokonevuorovaikutuksen (HCI), käyttäjälähtöisen suunnittelun (UCD), käyttökokemuksen (UX) sekä sovelluksen (Application) välinen suhde (Femto15, User Experience and User Centered Design (UX and UCD)).

2.1 Vaatimusmäärittelyn tutkimusmetodit

Sovelluksen vaatimusmäärittelyn tutkimuksessa käytin laadullisia, eli kvalitatiivisia menetelmiä. Laadullisessa tutkimuksessa tietoa kerätään sanallisessa muodossa. Kvalitatiivista tietoa ovat esimerkiksi tutkittavien kertomukset itsestään tai vapaamuotoisten haastattelujen tuottama aineisto. Ilmiöiden mittaamisen ja selittämisen sijaan tutkijaa kiinnostaa kuvata, luokitella, ymmärtää ja tulkita tutkimusaineistoaan. Kvalitatiivista tutkimusta tehdään esimerkiksi silloin, kun tutkijaa kiinnostaa yksilön subjektiiviset kokemukset, jotka liittyvät tutkimuksen kohteena olevaan asiaan. Kvalitatiivisessa tutkimuksessa tietoa kerätään yleensä melko pieneltä ihmismäärältä, sillä tällaisen tiedon käsittely on hidasta, eikä sitä voi automatisoida.
(Paavilainen 2012; Jyväskylän Yliopiston Koppa 2015.)

Ennen sovelluksen toteutusta tulevaa sovellusta eniten käyttäviä henkilöitä haastateltiin siitä, millaisia toiveita heillä on uuden raportointityökalun suhteen. Vertailukohtana käytettiin vanhaa raportointisovellusta. Haastateltavilta tiedusteltiin, että mitkä vanhan sovelluksen ominaisuudet ovat heidän mielestään hyviä, mitkä ominaisuudet kaipaisivat parannusta, ja mitä ominaisuuksia puuttuu kokonaan.

Myös itselläni oli hyvä käsitys siitä, miten sovelluksen tulisi toimia. Sovelluksen kehitystyötä aloittaessa olin työskennellyt Radio Helsingin tekniikan päällikkönä noin kuuden vuoden ajan. Eräs työtehtävistäni oli huolehtia, että kaikkien ohjelmien soittotapahtumaraportit on täytetty, ja että raportit toimitetaan ajallaan tekijänoikeusjärjestöille. Raportoinnista huolehtiminen piti sisällään myös uusintaohjelmien raporttien monistamista, ilman juontajaa lähetettyjen soittolistojen raportoinnin, sekä rästiin jääneiden raporttien täyttämistä. Kuuden vuoden aikana ehdin muodostaa hyvän käsityksen siitä, miten raportointia voisi tekniikan päällikön näkökulmasta sujuvoittaa.

Sovelluksen tuotantokäytön aloituksen jälkeen käyttäjiltä kerättiin lisää dataa ominaisuu sista, jotka helpottaisivat heidän työskentelyään, sekä mahdollisista sovelluksen ulkoasumuutoksista tapaustutkimuksen mentelmin. Tapaustutkimukseksi kutsutaan tutkimusstrategiaa, jossa tarkoituksena on tutkia syvällisesti vain yhtä tai muutamaa kohdetta tai ilmiökokonaisuutta. Tapaustutkimuksessa pyritään tuottamaan valitusta tapauksesta yksityiskohtaista ja intensiivistä tietoa. Usein tapaustutkimuksen tavoitteena on tehdä muihin yksilöihin tai tapahtumiin yleistettävissä olevia päätelmiä tai löytää jotain yleisiä, kaikkien ihmisten toimintaan vaikuttavia lainalaisuuksia. Tapaustutkimuksen tulokset perustuvat usein myös tutkijan omiin tulkintoihin tutkimusaineistosta. Tapaustutkimusta pidetään arvokkaana, koska se antaa ideoita uudenlaisiin teorioihin ja tuo esiin uusia tutkimushypoteeseja. Käyttäjien tutkimus keskittyi pitkälti raportointiprosessin kulkuun, ja mikäli prosessia olisi mahdollista edelleen suoraviivaistaa ja automatisoida työskentelyn sujuvoittamiseksi.
(Paavilainen 2012; Jyväskylän Yliopiston Koppa 2015.)

3 Teknologiat

Kappaleessa esitellään sovelluksessa käytetyt teknologiat jaoteltuna yleisiin teknologioihin, palvelin- eli backend-teknologioihin, sekä käyttöliittymä- eli frontend-teknologioihin.

3.1 Yleiset

Yleiset teknologiat ovat käytössä laajasti koko sovelluksessa, eikä niiden käyttö rajoitu ainoastaan tiettyyn sovelluksen osa-alueeseen.

3.1.1 Javascript

Javascript on alunperin kehitetty Netscape-selaimeen 1990-luvulla. Javascript-kieli on nykyään standardoitu ECMAScript-standardissa. ECMAScript-kieltä kehitetään edelleen. Tällä hetkellä uusin versio on ECMAScript 2019 mutta käytännössä käytössä on ECMAScript 2015 (ES6). Javascript (ECMAScript) -kieltä käytetään useissa eri yhteyksissä, mutta erityisesti WWW-selaimissa. Selaimessa suoritetulla Javascriptilla voi muokata selainympäristöä ja siihen liittyviä objekteja.

JavaScriptiä käytetään myös palvelinten verkko-ohjelmoinnissa (esimerkiksi Node.js -ajoympäristössä), pelien kehityksessä ja työpöytä- sekä mobiilisovellusten luomisessa. JavaScript ei itsessään määrittele, miten tietoa voidaan syöttää tai tulostaa, vaan se tarjoaa ainoastaan tiedon käsittelymekanismit. Javascript ei liity millään tavalla Java-ohjelmointikieleen. Sekä React- että Nodejs –sovellukset kirjoitetaan JavaScriptillä. (JavaScript Perusteet Jyväskylän yliopisto.)

3.1.2 Tekstieditori, koodin formatointityökalut ja selainten kehittäjätyökalut

Visual Studio Code (VS Code) on avoimen lähdekoodin monialustainen tekstieditori. Visual Studio Code on tällä hetkellä saatavilla Linuxille, MacOS:lle ja Windowsille. Siinä on sisäänrakennettu tuki JavaScriptille, TypeScriptille ja Node.js:lle, ja tuki virheenkorjaukselle, Git-versionhallinnalle, syntaksin korostukselle, automaattiselle koodin täydennykselle, katkelmille ja refaktoroinille.

VS Code on laajalti mukautettavissa käyttäjän tarpeiden mukaan. Lisäksi siihen voi asentaa tai tehdä itse laajennuksia. Laajennukset voivat esimerkiksi tukea uutta ohjelmointikieltä tai tiettyjä koodin tyylikäytäntöjä. (Visual Studio Code, Visual Studio Code docs.)

Eslint on työkalu jolla määritellään tyylisäännöt JavaScript-koodille. Työkalun tavoitteena on tehdä koodista johdonmukaista ja välttää virheitä. Eslintissä on omat sisäänrakennetut

sääntönsä, joita sovelluskehittäjä voi tarpeen mukaan laittaa päälle tai pois päältä. Airbnb ylläpitää monen JavaScript-kehittäjän suosiossa olevaa tyylikirjastoa, jonka avulla kehittäjien on helpompi päästä kärryille muiden sovelluskehittäjien kirjoittamasta koodista.

Eslint on konfiguroitavissa toimimaan yhdessä VS Coden kanssa siten, että editori antaa kehittäjälle virheilmoituksia mikäli koodi poikkeaa määritellyistä Eslint-säännöistä. (Eslint dokumentaatio, Eslint.)

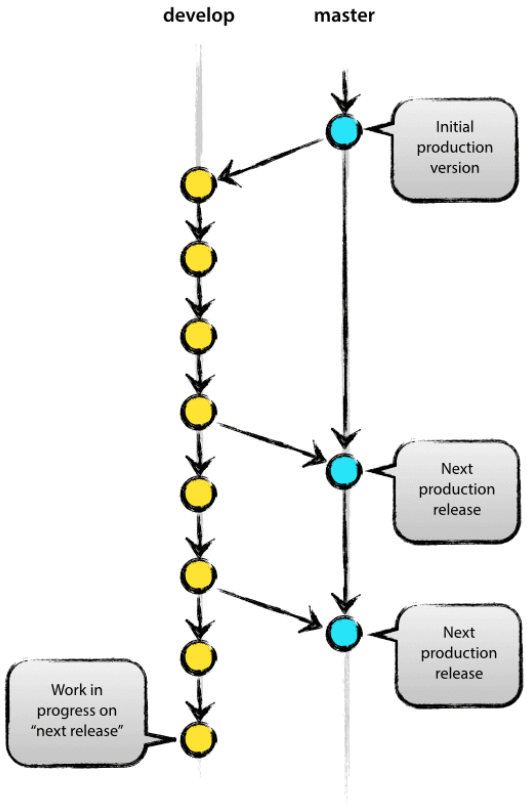
Prettier on VS Codeen asennettava koodinformatointityökalu. Prettier ylläpitää yhdenmukaista tyyliä mitä tulee esimerkiksi sulkujen, puolipisteiden ja sisennyksien käyttöön. Prettier uudelleenformatoi koodin tallennuksen yhteydessä vastaamaan sille määriteltyjä sääntöjä. Esimerkiksi puolipisteiden lisääminen koodirivin loppuun on mahdollista määritellä automaattiseksi Prettierin avulla. (Prettier Github-repositorio, Prettier)

Moderneihin web-selaimiin, kuten Google Chromeen ja Mozilla Firefoxiin, on sisäänrakennettu kehittäjätyökaluja, joilla sovelluskehittäjä pystyy tarkastelemaan web-sovelluksen rakennetta, toimintaa, suorituskykyä sekä diagnosoimaan ongelmia.

3.1.3 Git & Github

Git on versionhallintajärjestelmä, joka suunniteltiin tukemaan hajautettua työskentelyä, ja estämään datan virheellisyys sekä katoaminen. Git-projektia kutsutaan repositorioksi ja Gitiä käytetään useimmiten komentorivityökalulla. Vaikka Gitiä käytetään pääasiassa koodauksessa, Gitiä voisi käyttää minkä tahansa tiedostojen, kuten esimerkiksi Word-dokumenttien tai videoeditointiprojektien hallinointiin.

Github on Git-repositorioiden hostauspalvelu, jossa sovelluskehittäjät säilövät ja julkaisevat tekemäänsä koodia. Github mahdollistaa Gitin käytön graafisella käyttöliittymällä komentorivityökalun sijaan. Githubin käyttäjä voi kloonata toisen käyttäjän repositorion, tehdä siihen muutoksia, ja pyytää repositorion alkuperäistä omistajaa yhdistämään muutokset alkuperäiseen repositorioon. (What exactly is Github anyway, TechCrunch)



Kuva 2. Yksinkertaistettu Git develop-master –versionhallinta (A successful Git branching model, Nvie Blog).

3.2 Frontend

Suurella osalla verkkosivuista on yleisesti ottaen kaksi puolta, frontend ja backend. Näitä puolia voidaan myös kutsua nimillä selainpuoli ja palvelinpuoli. Frontend eli selainpuoli on kaikki se koodi, joka ajetaan verkkoselaimessa eli käyttäjän silmien edessä ja jonka kanssa käyttäjä voi olla tekemisissä.

Tyypillisesti verkkosivun frontend koostuu HTML-merkintäkielellä kirjoitetusta rakenteesta, CSS-merkintäkielellä kirjoitetusta ulkoasusta ja JavaScriptilla tai muulla vastaavalla ohjelmointikiellellä kirjoitetusta toiminnallisuudesta. (Mikä ihmeen frontend, Zaibatsu Interactive)

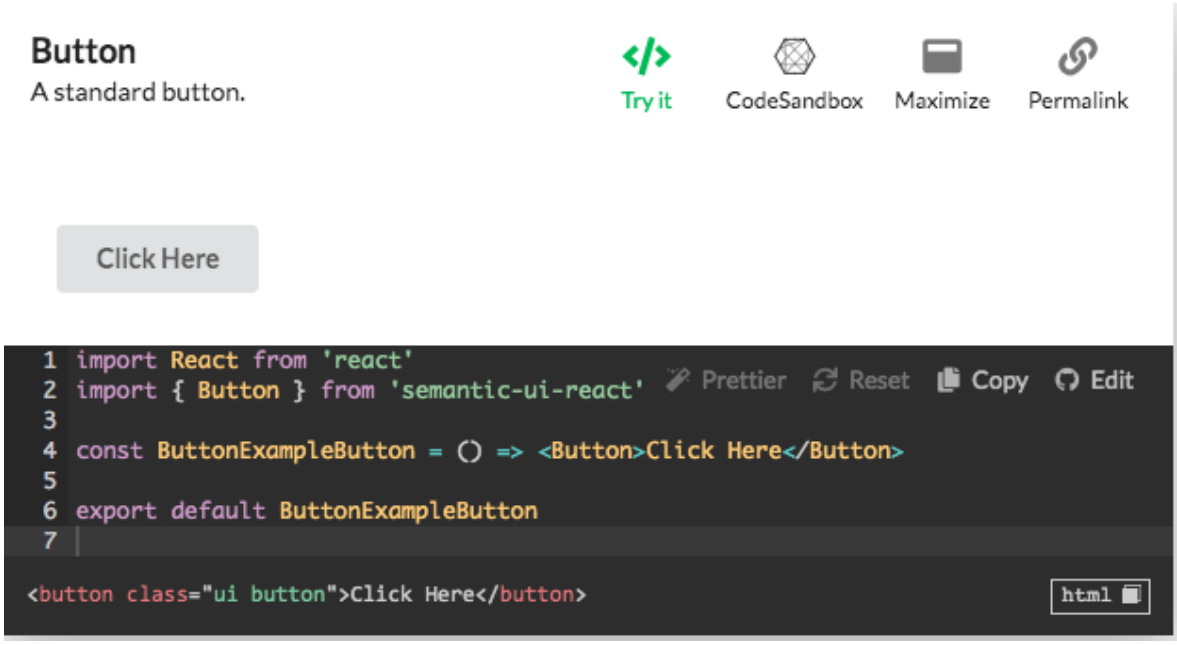
3.2.1 React

React on Facebookin kehittämä, komponenttipohjainen JavaScript-kirjasto web-käyttöliittymien tekemiseen. Reactin filosofiana on koostaa sovellus useista, pieneen asiaan keskittyvistä uudelleenkäytettävistä komponenteista. Komponentteja yhdistelemällä monimutkaisempikin sovellus on mahdollista pitää kohtuullisen ylläpidettävänä. (Web-ohjelmointi, Tampereen Yliopisto.)

React-komponentilla voi tarvittaessa olla tila, jossa säilytetään dataa. Mikäli tila muuttuu, niin komponentti renderöidään uudelleen. Yksinkertaisissa komponenteissa, kuten esimerkiksi lomakkeiden tekstikentissä, on perusteltua käyttää komponentin omaa tilaa datan käsittelyyn. Tila on kuitenkin vain kyseisen komponentin käytössä, eikä sovelluksen muilla komponenteilla ole suoraan pääsyä tilan dataan. Sovelluksen kasvaessa onkin syytä ottaa käyttöön jokin ratkaisu kattamaan koko sovelluksen tila. Sovelluksen tilaratkaisuna voidaan käyttää esimerkiksi Reactiin sisäänrakennettua Context APIa, tai ulkopuolista Redux-kirjastoa.

JavaScript –sovelluskehyksistä puhuttaessa keskustelu kääntyy yleensä kolmeen suurimpaan, eli Reactiin, Angulariin, ja Vueen. Jokaisella kehyksellä on omat hyvät ja huonot puolensa. Oikeaa vaihtoehtoa ei ole, vaan kyse on enemmänkin kehitystiimin preferenssistä. Raportointisovelluksen mittakaavassa yhdenkään sovelluskehyksen suorituskyky ei muodostunut kynnyskysymykseksi, joten React valikoitui käytettäväksi kehykseksi helppokäyttöisyytensä, sekä hyvien opiskeluresurssien perusteella. Muilla kehyksillä olisi varmasti päästy yhtä hyviin lopputuloksiin.
(React vs Angular vs Vue, What to Choose in 2020, Techmagic)

Käyttöliittymän tyylit ja komponentit on toteutettu käyttäen Semantic UI –kirjastoa. Semantic UI on moderni css-kirjasto, joka käyttää toiminnallisuudessaan LESS-CSSaa ja jQueryä. Semantic UI:n design on hienovaraista ja se tarjoaa kevyen käyttökokemuksen. Semantic UI:n dokumentaation mukaan kirjaston tavoite on helpottaa suunnittelijoiden työtä luomalla kieli käyttöliittymän jakamiseen. Tämä saavutetaan käyttämällä semanttista, kuvaavaa kieltä komponenttien nimeämiskäytännössä. Semantic UI:ta käyttäviä React-sovelluksia varten on olemassa Semantic UI React –kirjasto. (Introducing Semantic UI component library, Sitepoint.)



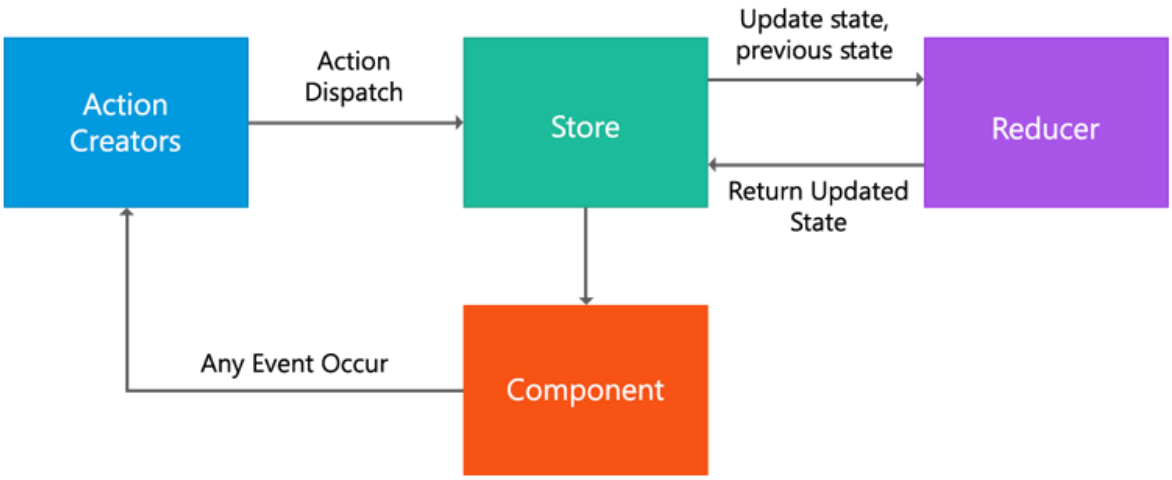
Kuva 3. Semantic UI Reactin Button-komponentin dokumentaatio (Button, Semantic UI).

3.2.2 Redux

Redux avoimen lähdekoodin JavaScript-kirjasto, joka on suunniteltu sovelluksen globaalin datan jakamiseen. Siitä on tullut erityisen suosittu React-sovelluksissa, mutta se toimii myös muiden JavaScript-sovellusten kanssa. Redux tarjoaa koko sovellukselle yhteisen tilasäilön, joka on kuvattu JavaScript-objektina. Sovelluksen kaikki komponentit voidaan liittää tilasäilöön riippumatta niiden sijainnista komponenttihierarkiassa.

Reduxin ytimessä on Reducer-funktio, joka pitää sisällään sovelluksen alustavan tilan. Tilasäiliön muuttamiseksi lähetetään toiminto, eli JavaScript-objekti jossa kuvataan haluttu muutos. Reducer-funktio ottaa vastaan toiminnon ja palauttaa sovelluksen päivitetyn tilan. Reducereita voi myös olla useita, jolloin reducerit yhdistetään yhdeksi juuri-reduceriksi. (Redux dokumentaatio, Redux.)

React-sovelluksissa Reduxia käytetään React-Redux-kirjaston kanssa. React-Redux on virallinen kirjasto, joka sitoo Reactin yhteen Reduxin kanssa. React-Redux edistää hyvää sovellusarkkitehtuuria ja auttaa optimoimaan sovelluksen suorituskykyä. (Why use React-Redux, React-Redux.)



Kuva 4. Reduxin datan kulku (Getting started with React Redux part 1, DOTNet Basic).

3.3 Backend

Backendillä tarkoitetaan web-sovellusten palvelinpuolta, eli sovelluksen pääkäyttäjälle näkymätöntä osaa. Backendissa tapahtuvat esimerkiksi lomakkeisiin syötetyn datan käsittely, käyttäjien kirjautuminen ja salasanojen tarkastaminen, järjestelmäintegraatiot ja tietojen tallennus ja haku tietokannasta. (Mitä markkinoijan tulee ymmärtää web-ohjelmoinnista, Dagmar)

3.3.1 Node.js & Express

Node.js on Googlen chrome V8 –Javascript-moottoriin perustuva Javascriptin suoritusympäristö. Node.js on suunniteltu helposti skaalautuvien web-sovellusten kehittämiseen. Käynnistyessään Node.js luo palvelimen, joka kykenee vastaamaan palvelupyyntöihin sekä jakamaan staattisia tiedostoja.

Palvelimen koodin tekeminen suoraan Noden sisäänrakennetun web-palvelimen http:n päälle on mahdollista, mutta työlästä, erityisesti jos sovellus kasvaa hieman isommaksi. Nodella tapahtuvaa web-sovellusten ohjelmointia helpottamaan onkin kehitelty useita http:tä miellyttävämmän ohjelmointirajapinnan tarjoamia kirjastoja. Näistä ylivoimaisesti suosituin on Express. (Node.js ja Express, Full Stack Open 2020)

Express on kevyt, suosittu ja monipuolinen Node.js –sovelluskehys, joka tarjoaa luotettavat perusteet web-sovellusten rakentamiseen. Express on Node.js:n päällä toimiva kerros, jonka avulla palvelin pystyy käsittelemään useita http-kutsuja useissa eri osoitteisiin.

Node.js:n yhteydessä asennetaan aina myös NPM (Node Packcage Manager), eli Node.js:n paketinhallintajärjestelmä. NPM on maailman suurin avoimen lähdekoodin oh-

jelmistokirjastojen rekisteri. Se tarjoaa mahdollisuuden tallentaa ja julkaista JavaScript-kirjastoja ja -ohjelmakomponentteja ilmaiseksi. Pakettejen ei tarvitse olla Node.js-ympäristön käyttöön tarkoitettuja, vaan esimerkiksi React-sovellusten kirjastot asennetaan NPM:aa käyttäen.

Kaikki sovelluksen riippuvuudet voivat löytyä samasta paketinhallinnasta JSON-tiedostosta package.json, jonka avulla esimerkiksi toinen sovelluskehittäjä voi helposti jatkaa sovelluksen kehitystyötä vain asentamalla sovelluksen riippuvuudet omaan järjestelmäänsä. (What is NPM, W3Schools.)

Node.js valikoitui palvelimen teknologiaksi, sillä se on helppo oppia, helppo käyttää ja se toimii kaikilla alustoilla. Node.js käyttää JavaScriptiä, joten JavaScriptiä osaamalla Node.js:n perusteet oppii parissa päivässä.
(Why is Node.js popular, Section.io)

Muita yleisiä ohjelmointikieliä palvelinten tekemiseen ovat esimerkiksi Java, C, C++, Ruby, PHP ja Python. Toisin kuin JavaScriptin kanssa, näiden ohjelmointikielien käyttö ei rajoitu pelkästään web-sovelluksiin,
(What Languages Are Used For Backend Development, Sagara Technologies)

3.3.2 REST & Postman

REST (Representational State Transfer) on yleinen arkkitehtuurimalli internetissä olevien palveluiden keskinäiseen viestintään. REST määrittelee, millaisilla operaatioilla palvelinten dataa pyydetään, lisätään ja käsitellään. REST-määritelmä koostuu joukosta ehtoja, jotka rajapinnan on täytettävä.

RESTin tärkeitä määrittäviä tekijöitä ovat tilattomuus ja palvelin-asiakas –malli. Tilattomuudella tarkoitetaan sitä, että kaksi erillistä pyyntöä eivät tiedä suoraan toisistaan. Kaikki pyyntöön liittyvä tieto siirretään jokaisella pyynnöllä ja tilan säilyttäminen on pyynnön tekijän, esimerkiksi selaimessa suoritettavan sovelluksen, vastuulla. Palvelin-asiakas –mallilla tarkoitetaan sitä, että asiakas (frontend) tekee pyyntöjä esimerkiksi tiedon hakemisesta, lisäämisestä tai poistamisesta, ja palvelin (backend) vastaa pyyntöjen toteuttamisesta. REST ei ota kantaa palvelimen sisäiseen rakenteeseen.

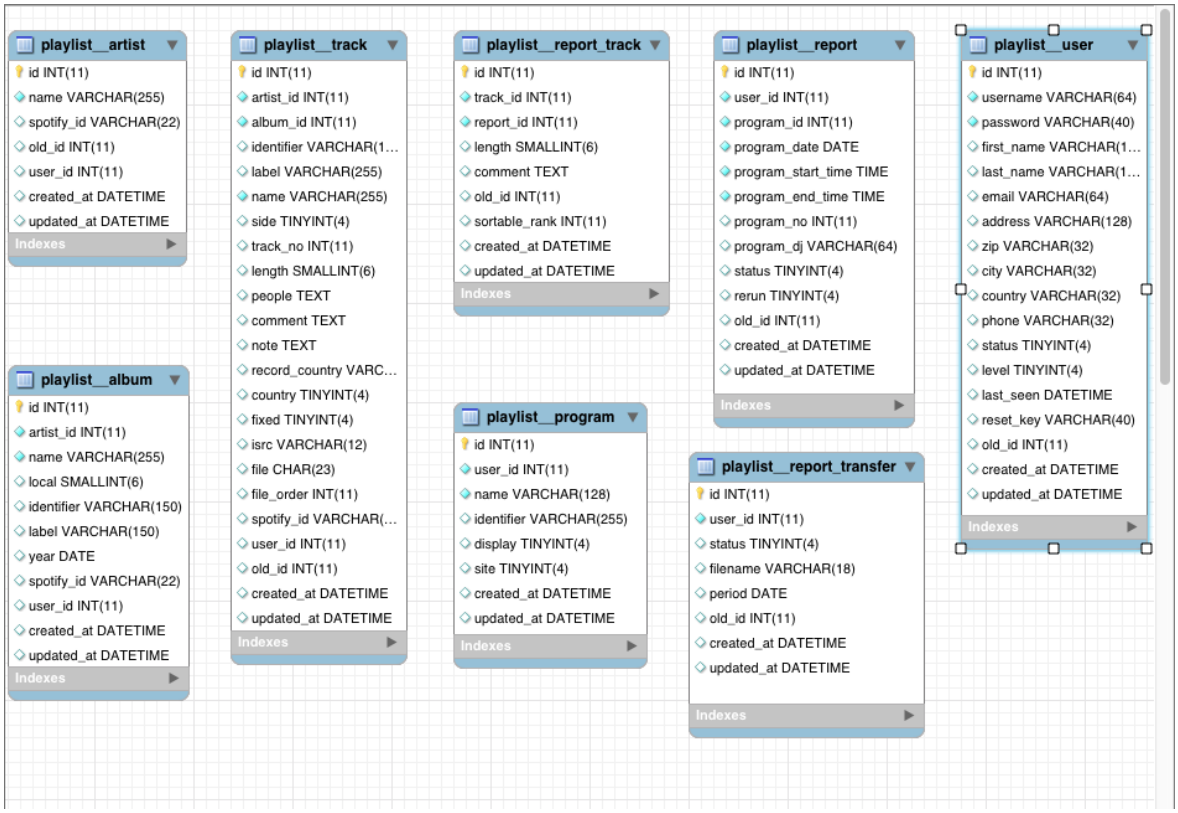
Rest pohjaa vahvasti http-protokollan ominaisuuksiin. Http:n yleisimpiä metodeja (GET, POST, PUT ja DELETE) sekä eri uri:ta käytetään kuvaamaan pyynnön luonnetta, jolloin

itse pyynnön dataan ei tarvitse sisällyttää metatietoja. Tämä tekee datasta helppolukuisempaa. (Rest on nettipalveluiden yhteinen kieli, Tivi)

Postman on REST APIen testaamiseen käytetty sovellus. Se tarjoaa yksinkertaisen käyttöliittymän jolla tehdä http-pyyntöjä APIlle ilman, että käyttäjän tarvitsee kirjoittaa riviäkään koodia. Postmaniin on myös mahdollista määritellä ympäristömuuttujia, kuten APIn endpoint-url tai http-pyynnön mukana lähetettävä käyttäjän kirjautumistoken. Käyttäjä voi myös tallentaa tekemänsä pyynnöt vapaasti määritteltyihin kokoelmiin, joista pyyntöjen tekeminen uudestaan onnistuu helposti. Postmaniin tallennetuista kokoelmista on myös helppo tuottaa APIn dokumentaatio HTML-formaatissa. (REST-rajapintojen testaus Postmanilla, Nemit Blog)

3.3.3 Tietokanta

Sovelluksen tietokanta on **MySQL**-tietokanta, tauluja sisältävä relaatiotietokanta, jota useimmiten käytetään verkkopalvelujen datan säilömiseen. Taulut koostuvat edelleen kentistä ja riveistä. Tässä sovelluksessa esimerkiksi artistit, albumit, kappaleet ja käyttäjät muodostavat jokainen oman taulunsa. Taulun kenttiä puolestaan ovat esimerkiksi albumitaululla yksilöivä id-numero, albumin nimi, albumin artistin id, vuosi, levymerkki ja levykoodi. Rivi puolestaan voisi sisältää edellä mainitut tiedot yhdestä albumista.



Kuva 5. Tietokannan taulut, Kuvankaappaus MySQL Workbench -ohjelmasta.

SQL-kielellä kirjoitetut kyselyt hakevat ja muuttavat tietokannan tietoja. SQL-kieltä käytetään myös monissa muissa tietokantajärjestelmissä. Eri järjestelmien SQL-kielet poikkeavat toisistaan, mutta suurin osa perusasioista on yhteisiä. (MySQL ja PHP: Osa 1 - Johdanto, Ohjelmointiputka)

Suosittu SQL-kieltä käyttävä vaihtoehto MySQL:lle olisi PostgreSQL, joka on nopea ja tukee JSON-formaattia. Vaihtoehtoina olisi myös lukuisia non-SQL –tietokantoja, kuten MongoDB, MariaDB tai OracleDB. Tietokannan valinta raportointisovellukseen oli kuitenkin helppo, sillä sovelluksessa voitiin käyttää suoraan vanhan raportointisovelluksen MySQL-tietokantaa. Tietokannan migraatiossa uudelle alustalle ei tässä tapauksessa olisi saatu mainittavaa hyötyä. Vaihtoehtoja olisi ollut syytä harkita siinä vaiheessa, jos sovellusta olisi alettu rakentaa ilman olemassaolevaa tietokantaa.
(Alternatives to MySQL, Slant)

Node.js –ympäristössä on yleistä käyttää SQL-tietokantoja ORM-kirjaston kanssa. ORM (Object Relational Mapping) muuntaa SQL-tietokannasta haetun datan objektimuotoiseksi, sekä kirjastolla voi tehdä kyselyjä tietokannan tauluihin pohjautuvia malleja käyttämällä ilman, että tarvitsee kirjoittaa riviäkään SQL:aa. Yksi suosituimpia ORM-kirjastoja on Postgres, MySQL, MariaDB, SQLite ja Microsoft SQL –yhteensopiva Sequelize, joka palauttaa MySQL-tietokannasta haetun datan JavaScript-objektina. (Sequelize, Sequelize.)

```
25  // @desc    Get one user
26  // @route   GET /:id
27  // @access  Private
28  export const getOneUser = asyncHandler(
29    async (req: Request, res: Response, next: NextFunction) => {
30      const user = await User.findOne({
31        where: { id: req.params.id },
32        attributes: {
33          exclude: ['password']
34        }
35      });
36      if (!user) {
37        return next(
38          new ErrorResponse(`no user found with the id ${req.params.id}`, 404)
39        );
40      }
41      res.status(200).json(user);
42    }
43  );
44
```

Kuva 6. Sequelizen User-mallin findOne-metodia käyttävä, yhden käyttäjän id-numeron perusteella hakeva kysely. Kuvankaappaus VSCode-ohjelmasta

SQL-kyselyjä on myös mahdollista kirjoittaa suoraan SQL-kielellä ilman, että kyselyssä käytetään Sequelizen mallia. Tämä on ehkä helpompaa tapauksissa, joissa tietoa joudutaan hakemaan useammasta taulusta.

```
// @desc    Get one album
// @route   GET /albumdetails/:id
// @access  Private
export const getOneAlbum = asyncHandler(
  async (req: Request, res: Response, next: NextFunction) => {
    const album = await db.query(
      `
      SELECT al.name as album_name
      , al.id as album_id
      , al.label
      , al.identifier as cat_id
      , al.spotify_id
      , al.year
      , ar.name as artist_name
      , ar.id as artist_id
      FROM playlist__artist as ar
      INNER JOIN playlist__album as al ON al.artist_id = ar.id
      WHERE al.id = ${req.params.id}
    `,
      {
        type: QueryTypes.SELECT,
      }
    );
    if (album.length === 0) {
      return next(
        new ErrorResponse(`no album found with the id ${req.params.id}`, 404)
      );
    }
    res.status(200).json(album[0]);
  }
);
```

Kuva 7. SQL-kielellä artist- ja album-tauluista haettu yhden albumin tiedot palauttava kysely. Kuvakaappaus VS Code -ohjelmasta.

Tietokantaa ylläpidetään Google Cloud Platformissa, joka on Googlen tarjoama pilvipalvelualusta. Google Cloud Platform käyttää samaa infrastruktuuria kuin esimerkiksi Googlen suositut palvelut Youtube, Gmail ja Google Search. Osana Googlen pilvipalvelualustaa on Cloud SQL for MySQL, täysin ylläpidetty tietokantapalvelu jonka avulla on helppo pystyttää, ylläpitää ja hallinnoida MySQL-relaatiotietokantoja. (Cloud SQL for MySQL documentation, Google).

4 Sovelluksen toiminta, rakenne & toteutus

Luvussa käydään läpi sovelluksen toiminta, projektirakenne ja toteutus. Luvussa esitellään myös tavanomainen navigaation kulku käyttäjien osalta. Lisäksi käydään läpi eri käyttäjäroolien toiminnot ja tekijänoikeusjärjestöille lähetettävän kuukausiraportin tulostus. Rakenneosuudessa esitellään koko projektin rakenne, ja avataan erikseen sekä backendin että frontendin rakennetta. Luvun lopussa esitellään miten soittolokista haettu data käsitellään ennen tietokantaan tallennusta, sekä miten tietokannasta haettu data muutetaan tekijänoikeusjärjestöjen vaatimaan formaattiin kuukausiraporttia tulostettaessa.

4.1 Käyttöliittymä

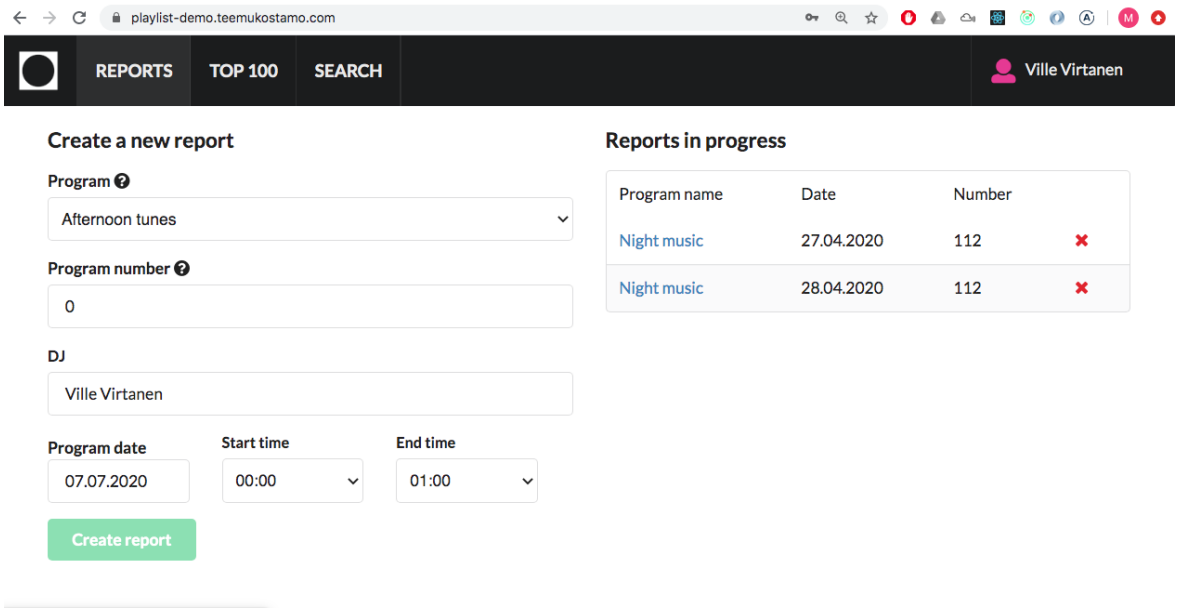
Sovellus on suunniteltu käytettäväksi desktop-laitteilla. Sovellus kuitenkin skaalautuu myös mobiililaitteille, sillä Semantic UI:lla toteutetut käyttöliittymät ovat oletuksena responsiivisia. Kuitenkin esimerkiksi navigointipalkissa ja yksittäisen raportin sivulla olevassa kappalelistauksessa on niin paljon tietoa, että mobiililaitteelle skaalatuessaan näkymä vaikuttaa täyteen ahdetulta.

4.1.1 Etusivu, navigointipalkki ja uuden raportin luominen

Navigointipalkista löytyy kirjautuneelle käyttäjälle sallitut näkymät. Peruskäyttäjälle nämä ovat Raportit, Top 100, ja Haku. Navigointipalkin oikeassa reunassa näkyy kirjautuneen käyttäjän nimi. Nimeä klikkaamalla käyttäjä voi kirjautua ulos, tai muokata omia tietojaan.

Etusivun näkymä on jaettu kahteen osaan. Vasemmalla on lomake, jolla käyttäjä luo uuden raportin. Oikealla on kyseisen käyttäjän keskeneräiset raportit. Ohjelman nimeä klikkaamalla käyttäjä pääsee jatkamaan keskeneräisen raportin täyttämistä. Keskeneräisen raportin voi poistaa rivin oikeassa reunassa olevaa punaista raksia klikkaamalla.

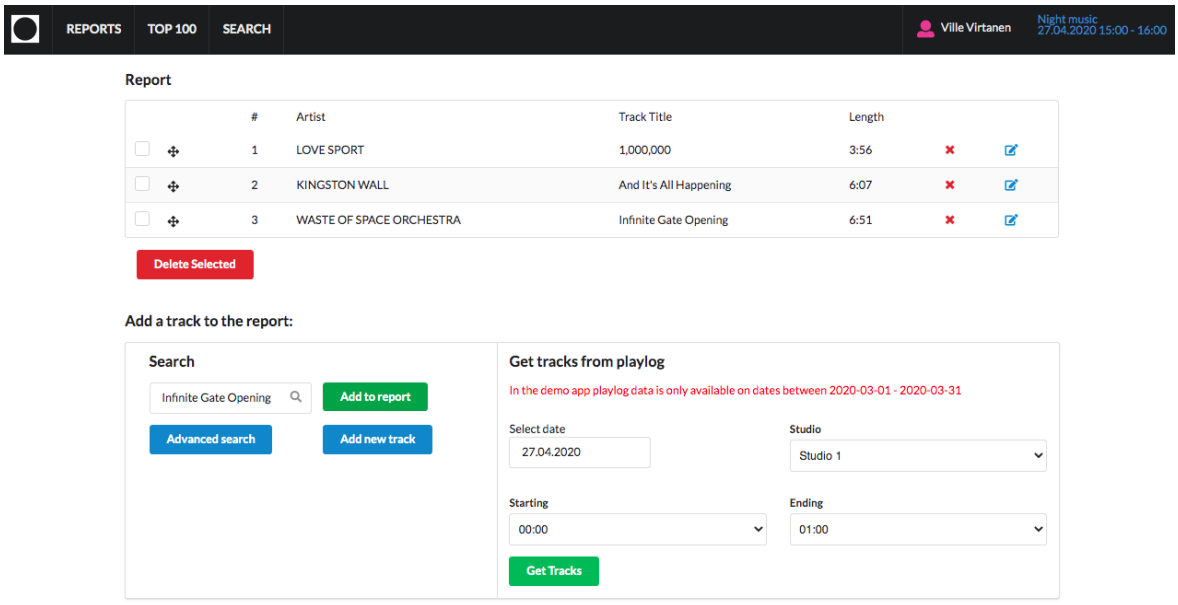
Uuteen raporttiin vaaditaan tiedot ainakin ohjelman nimestä, päivämäärästä ja kellonajasta. DJ-kenttään on oletuksena täytetty kirjautuneen käyttäjän etunimi ja sukunimi. Kentän nimeä voi vapaasti muuttaa, mikäli kyseisen ohjelman DJ on ollut joku muu kuin kirjautunut käyttäjä. Program number –kenttään täytetään kyseisen ohjelman numero, joka on juokseva kolmenumeroinen luku. Ohjelmanumero –kenttä on jäänne ajalta jolloin ohjelmat arkistoitiin CD-levyille, ja ohjelmien jaksot yksilöitiin ohjelmanumeroilla. Tieto ei enää ole pakollinen. Sovellus sallii käyttäjän siirtymisen raportin täyttämiseen vasta sitten kun pakolliset kentät on täytetty.



Kuva 8. Sovelluksen etusivun näkymä.

4.1.2 Kappaleiden lisääminen raporttiin ja Raportit -sivu

Täytettyään uuteen raporttiin vaadittavat tiedot, käyttäjä siirtyy lisäämään kappaleita raporttiin. Ensimmäisenä listataan raporttiin jo lisätyt kappaleet. Listan kappaleiden järjestystä voi muuttaa raahaamalla nuolikuvakkeesta. Yksittäisen kappaleen voi poistaa punaisesta raksista rivin oikeassa reunassa. Merkkaamalla rivin vasemmassa reunassa olevan laatikon käyttäjä voi valita useita kappaleita poistettavaksi. Rivin oiken reunan sinisestä muokkaa –kuvakkeesta käyttäjä pääsee muokkaamaan yksittäisen kappaleen tietoja.



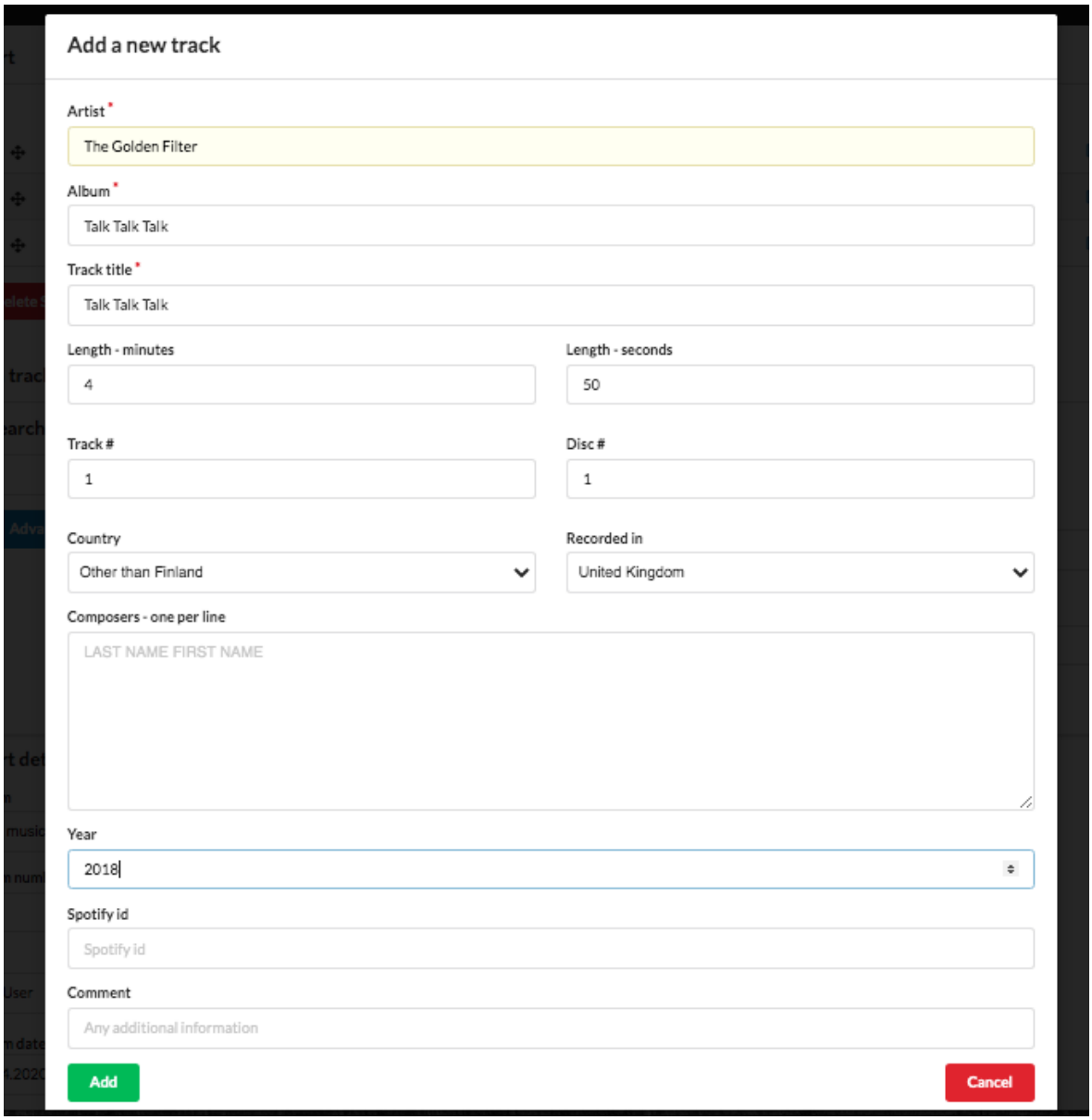
Kuva 9. Yksittäisen raportin sivu.

Lisää kappale raporttiin –kohdasta käyttäjä voi lisätä raporttiin jo aiemmin kanavalla soineita kappaleita sovelluksen pikahakua käyttäen, lisätä kokonaan uuden kappaleen raporttiin ja sovelluksen tietokantaan, tai hakea soittotapahtumat studion playout-ohjelmiston soittolokista päivämäärän ja kellonajan perusteella.

Mikäli käyttäjä soitti musiikin ainoastaan playout-ohjelmistoa käyttäen, riittää että soittotiedot haetaan soittolokista. Playlog-haussa käyttäjä voi myös valita, haetaanko soittotapahtumat Studio 1:sta vai Studio 2:sta. Studio 1 on pääasiassa käytössä suorissa lähetyksissä, kuin Studio 2:ta puolestaan käytetään ennakkoäänityksissä. Hakutulokset saa oikein, kun valitsee päivämääräksi ja kellonajaksi sen aikavälin, jolloin kappaleet soitettiin ulos playout-ohjelmistosta.

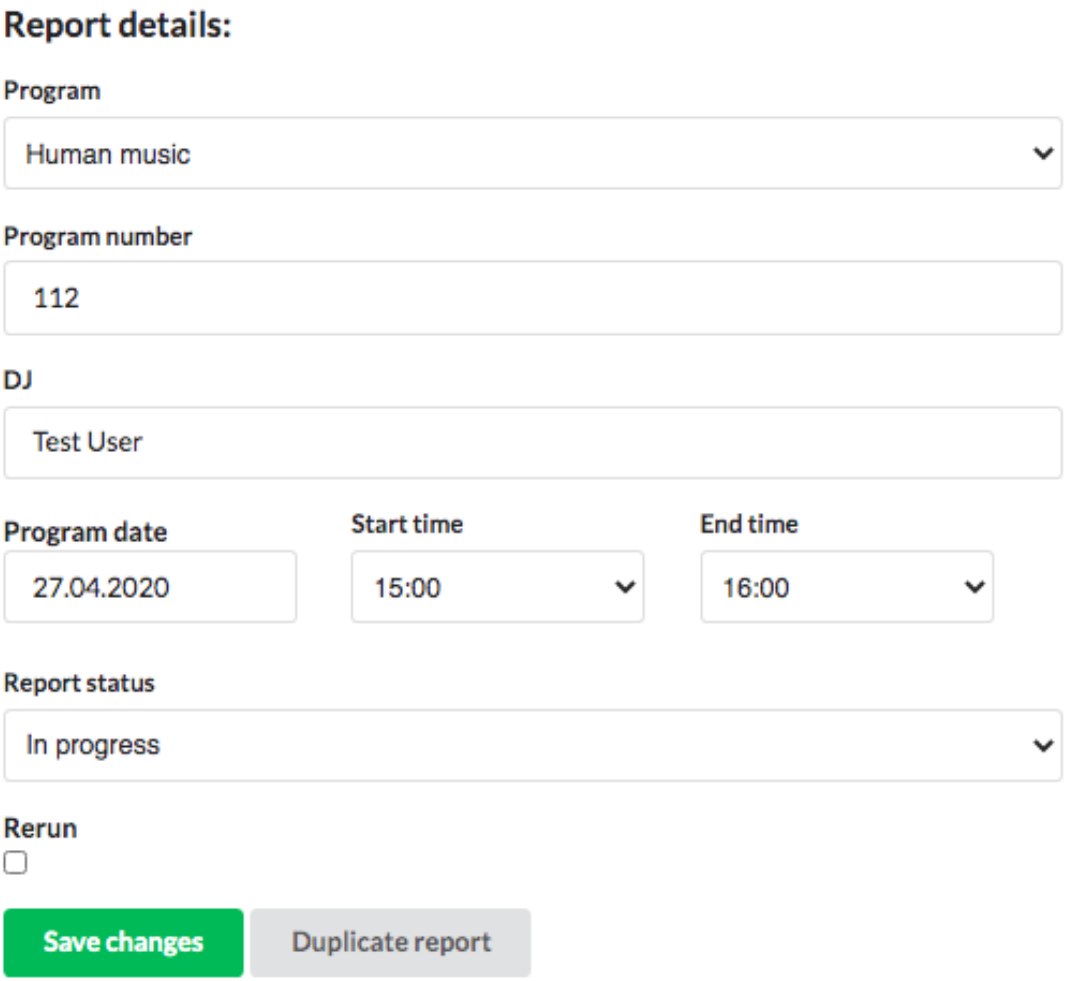
Pikahaku etsii tietokannasta hakusanaa vastaavat artistit ja kappaleet. Mikäli käyttäjän etsimä kappale löytyy pikahaun tuloksista, kappale lisätään raporttiin klikkaamalla ensin haun tulosta, ja sen jälkeen vihreää ”Lisää raporttiin” –nappulaa.

Mikäli soitettuja kappaleita ei löydy pikahaulla eikä soittolokista, käyttäjän täytyy lisätä uusi kappale. Uuden kappaleen lisäävä lomake vaatii pakollisina tietoina samat tiedot mitä tekijänoikeusjärjestöt vaativat radioita raportoimaan. Lisäksi käyttäjän on mahdollista kirjoittaa kappaleesta lisätietoa, tai lisätä kappaleen Spotify-linkki.



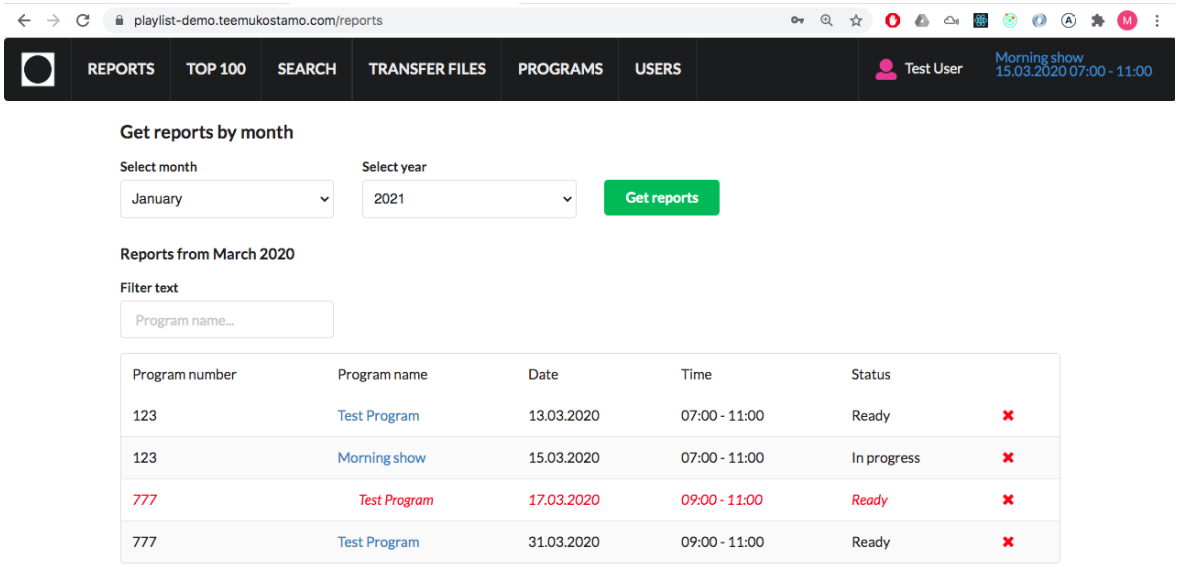
Kuva 10. Lomake uuden kappaleen lisäämiseen.

Raportti-sivun alimpana on lomake, jossa on tiedot raportin päivämäärästä ja kellonajasta. Käyttäjän on mahdollista muuttaa raportin tietoja mikäli vaikka raporttia lisätessä joitakin tietoja olisi syötetty järjestelmään väärin. Kun kaikki soitetut kappaleet on raportoitu, käyttäjä vaihtaa raportin tilaksi Valmis ja tallentaa muutokset. Kaikki raporttiin syötetyt kappaleet tallentuvat automaattisesti. Vain raportin yksityiskohtiin tehdyt muutokset täytyy tallentaa erikseen.



Kuva 11. Raportin yksityiskohdat –lomake.

Raportit –sivulla käyttäjä näkee tekemänsä raportit kuukausittain. Raportit on mahdollista suodattaa joko tekstisuodatuksella tai tilan (Valmis / Kesken) mukaan. Klikkaamalla punaista ruksia rivin oikeassa reunassa kyseinen raportti poistetaan. Punaisella tekstillä merkityt raportit ovat uusintaohjelmien raportteja. Sinisellä tekstillä merkityt ovat uusien ohjelmien raportteja.

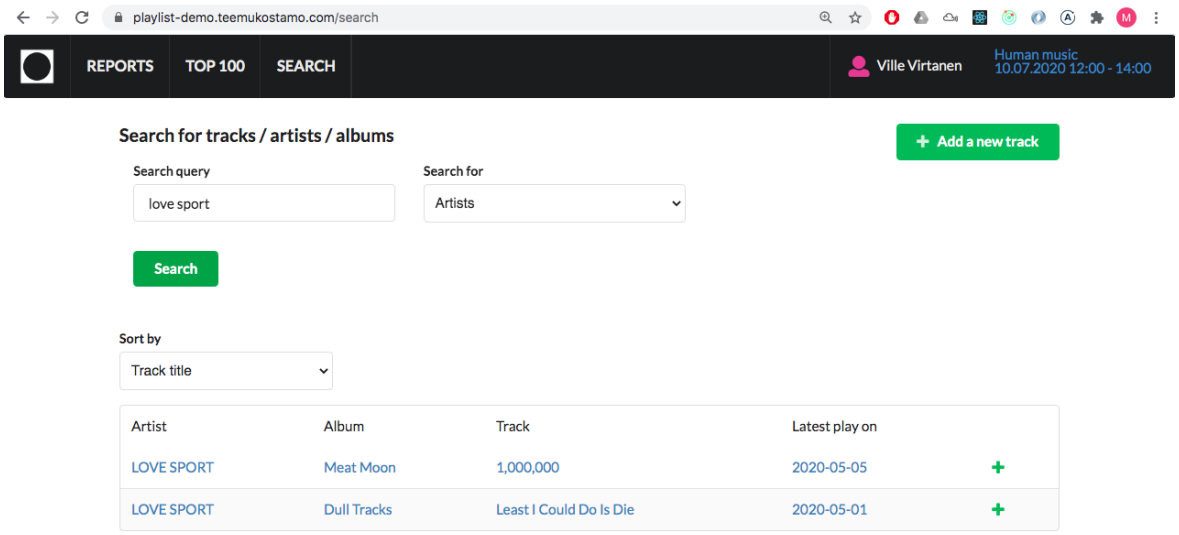


Kuva 12. Raportit-sivu.

4.1.3 Haku ja Top 100

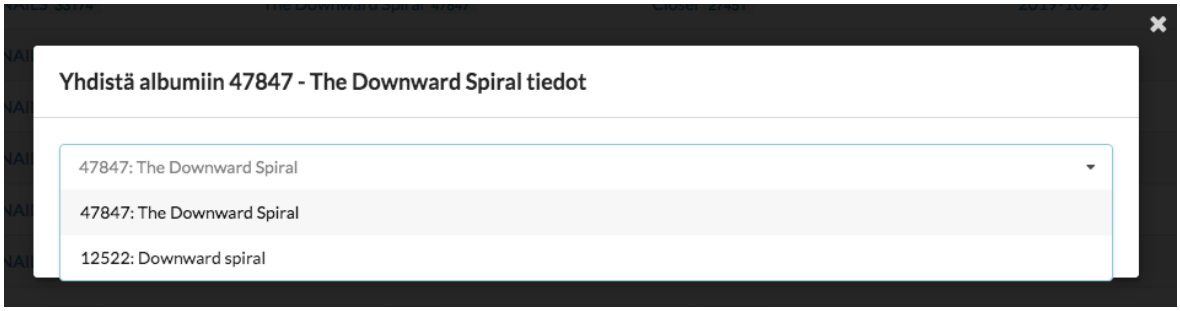
Haku -näkymässä käyttäjä voi hakea artisteja, albumeja tai kappaleita sanahaulla. Klikkaamalla artistin, albumin tai kappaleen nimeä voidaan niiden tietoja muokata. Klikkaamalla vihreää + -symbolia hakutuloksen rivin oikeassa reunassa ko. kappale lisätään aktiiviseen raporttiin. Hakutuloksen rivillä on nähtävissä myös päivämäärä, jolloin kyseinen kappale on soinut edellisen kerran.

Top100 -näkymässä käyttäjä näkee sata eniten soinutta artistia, albumia tai kappaletta tietyllä aikavälillä. Klikkaamalla vihreää + -symbolia hakutuloksen rivin oikeassa reunassa ko. kappale lisätään aktiiviseen raporttiin.



Kuva 13. Haku –sivu.

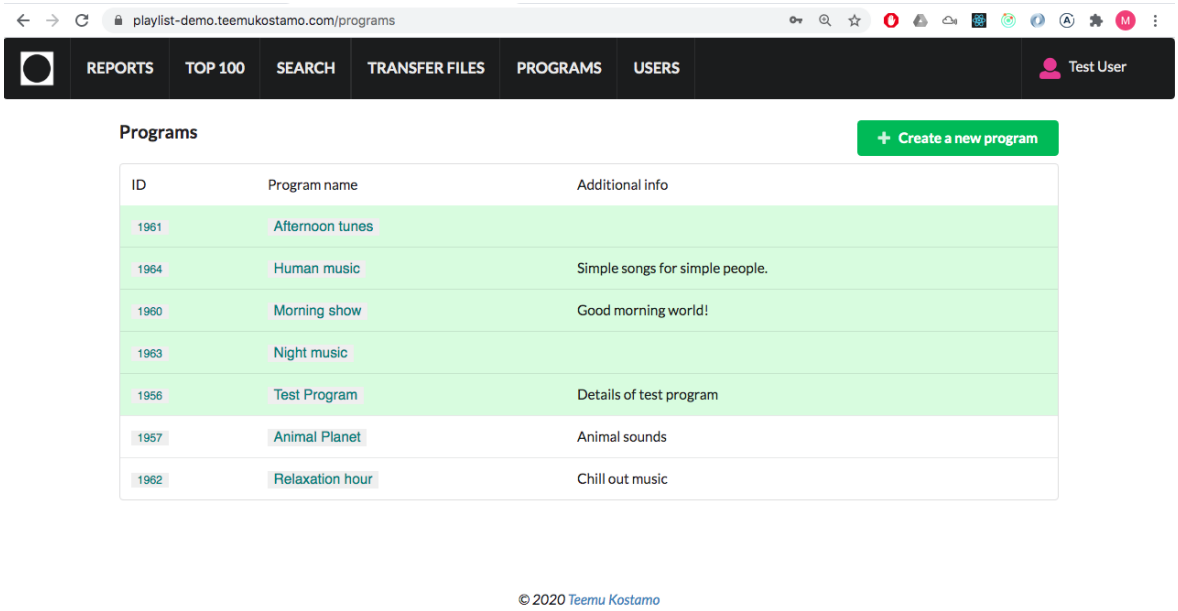
Sekä artistin, albumin ja biisin nimen perässä lukee vihreällä id-numero. Mikäli tuloksissa on duplikaatteja, klikkaamalla id-numeroa artistin, biisin tai albumin voi yhdistää toiseen ja näin poistaa duplikaatin tietokannasta.



Kuva 14. Yhdistä albumi –näkymä.

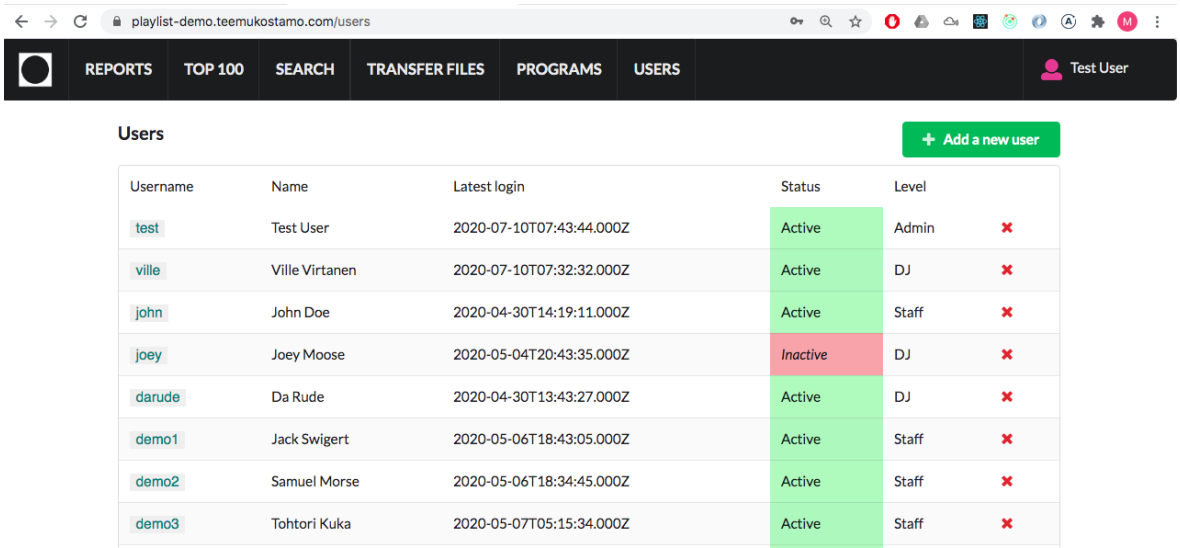
4.1.4 Ohjelmat ja Käyttäjät

Ohjelmat -sivulla käyttäjä voi muokata ohjelman tietoja. Aktiiviset ohjelmat näkyvät vihreällä pohjalla. Vaihtamalla ohjelman tilaksi aktiivinen ko. ohjelma saadaan näkyviin listalla josta valitaan uuden raportin ohjelma. Klikkaamalla listan vasemmassa reunassa olevaa ID-numeroa duplikaattiohjelma voidaan yhdistää toiseen ohjelmaan. Luo uusi ohjelma -painikkeella luodaan uusi ohjelma.



Kuva 15. Ohjelmat –sivu.

Käyttäjät -sivulla on lista sovelluksen käyttäjistä. Klikkaamalla käyttäjänimeä voi muokata käyttäjän tietoja. Käyttäjän tiedoista voidaan määrittää käyttäjätunnuksen olevan hyllyllä. Hyllyllä olevilla käyttäjillä ei ole pääsyä sovellukseen. Rivin oikeassa reunassa olevasta punaisesta ruksista käyttäjä voidaan poistaa. Lisää uusi käyttäjä -napilla luodaan uusi käyttäjä.

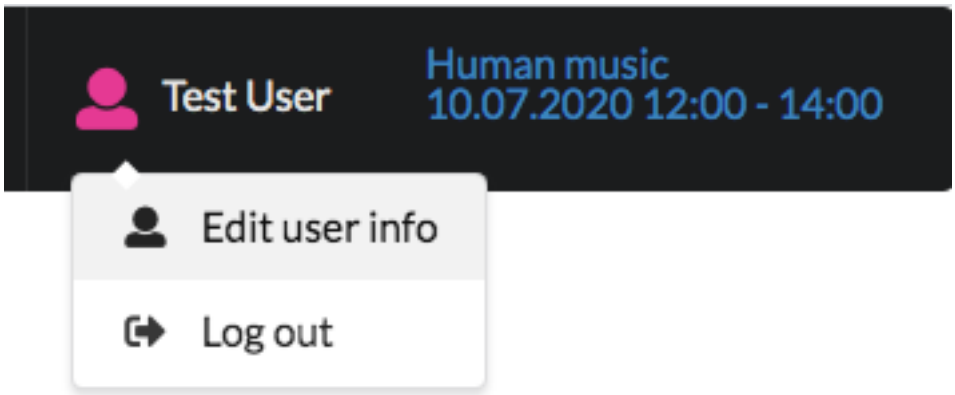


Kuva 16. Käyttäjät –sivu.

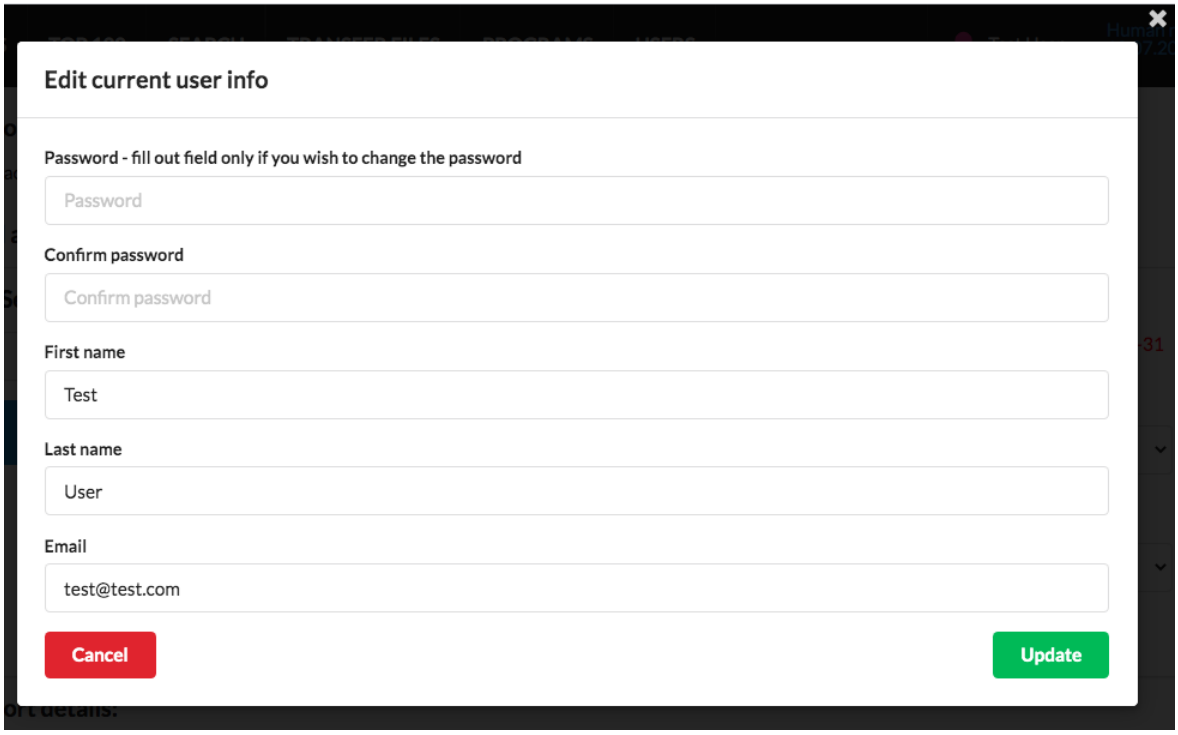
4.1.5 Omat tiedot, uloskirjautuminen ja aktiivinen raportti

Klikkaamalla omaa nimeään navigointipalkissa käyttäjän on mahdollista kirjautua ulos, tai muokata omia tietojaan. Tietojen muokkaussivulla on mahdollista myös vaihtaa kirjautumissalasanaa.

Navigaatiopalkista löytyy linkki aktiiviseen, eli viimeismpänä avattuun raporttiin. Ko. raportin ohjelman nimi ja päivämäärä näkyvät navigointipalkin oikeassa reunassa. Top100- ja Haku -näkymien kautta lisätyt kappaleet päätyvät tähän raporttiin.



Kuva 17. Muokkaa käyttäjän tietoja- uloskirjautuminen ja aktiivinen raportti –linkit.



Kuva 18. Käyttäjän tietoja muokkaava lomake.

4.2 Käyttäjätasot

Sovelluksessa on tarpeen olla kolme eri käyttäjätasoa:

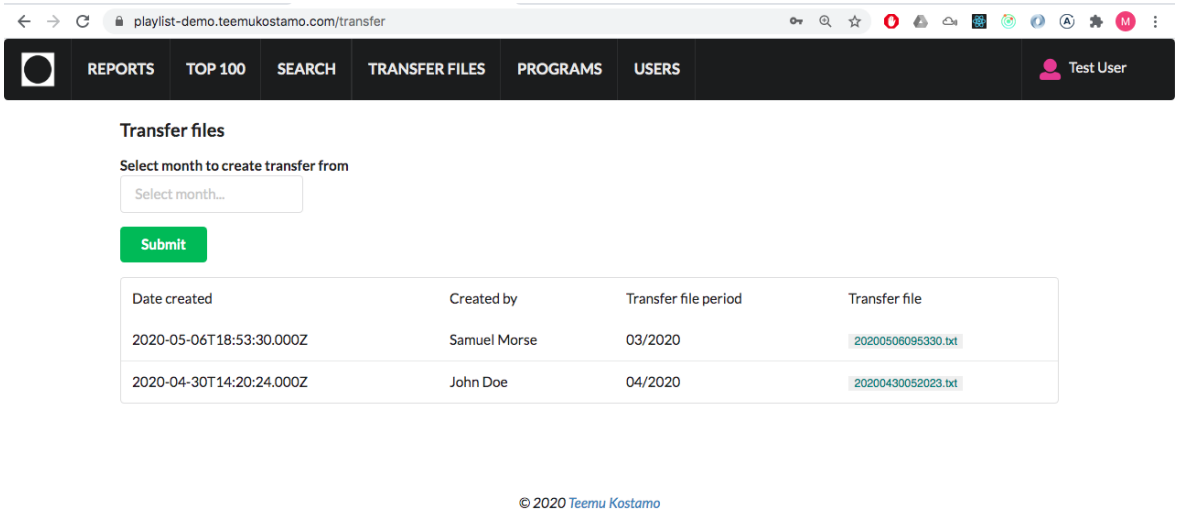
- Peruskäyttäjä (DJ)

- Toimitus (Staff)
- Admin

Peruskäyttäjällä on oikeus täyttää raportteja, selata Top 100 –soitetuimpia kappaleita tietyllä aikavälillä, hakea kappaleita, levyjä ja artisteja tietokannasta, sekä muokata näiden tietoja.

Toimitus –käyttäjätaso on suunniteltu kanavan työntekijöille, joiden vastuulla on joidenkin ohjelmien raporttien täyttämisen lisäksi monistaa uusintana lähetettävien ohjelmien raportit vastaamaan uusinnan päivämäärää ja kellonaikaa. Toimitus-tason käyttäjälle ovat siis näkyvissä kaikkien käyttäjien kaikki raportit. Toimitus-tason käyttäjä voi myös lisätä uusia ohjelmia.

Admin-käyttäjillä on samat oikeudet kuin DJ- ja Toimitus –tason käyttäjillä, oikeudet hallinnoida käyttäjiä, sekä pääsy Siirtotiedostot -sivulle. Siirtotiedostot -sivulla koostetaan tekijänoikeusjärjestöille lähetettävä raportti kuukauden soittotapahtumista. Käyttäjä valitsee kuukauden ja klikkaa HAE-nappia. Kun raportti on valmis, linkki tekstitiedostoon ilmestyy alla olevaan listaan. Listalla näkee ko. raportin luontipäivämäärän, raportin luoneen käyttäjän, raportin ajankohtan (MM/YYYY) ja linkin itse tiedostoon.

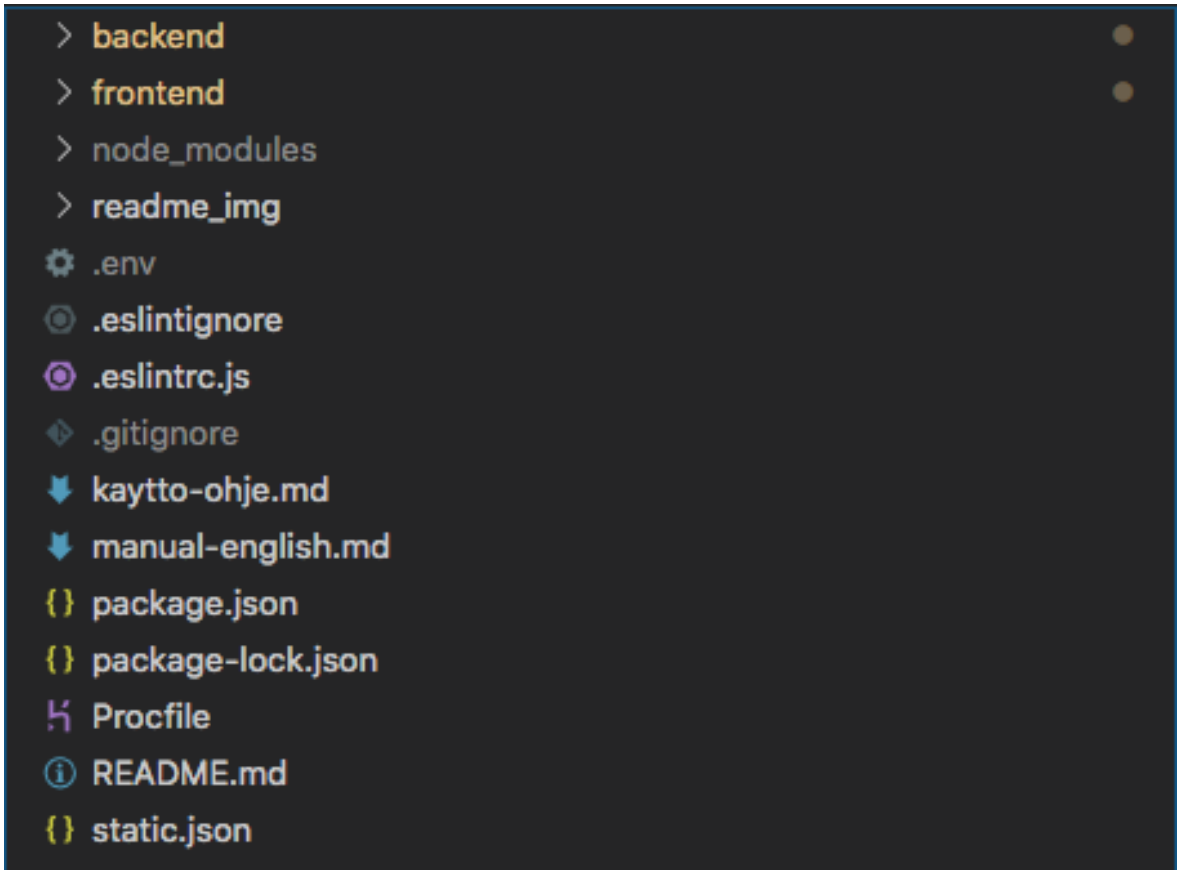


Kuva 19. Siirtotiedostot –sivu.

4.3 Projektin kansiorakenne

Koko sovellus sijaitsee yhdessä repositoriossa. Sovelluksen juuresta löytyy sovelluksen käyttö-ohjeet suomeksi ja englanniksi, asennus- ja käynnistysohjeet, sekä konfiguraatiotiedostot. Juuressa olevan package.json –tiedoston komennot ja riippuvuudet viittaavat

backendin komentoihin ja riippuvuuksiin. Backendin ja frontendin kooditiedostot löytyvät omista kansioistaan.

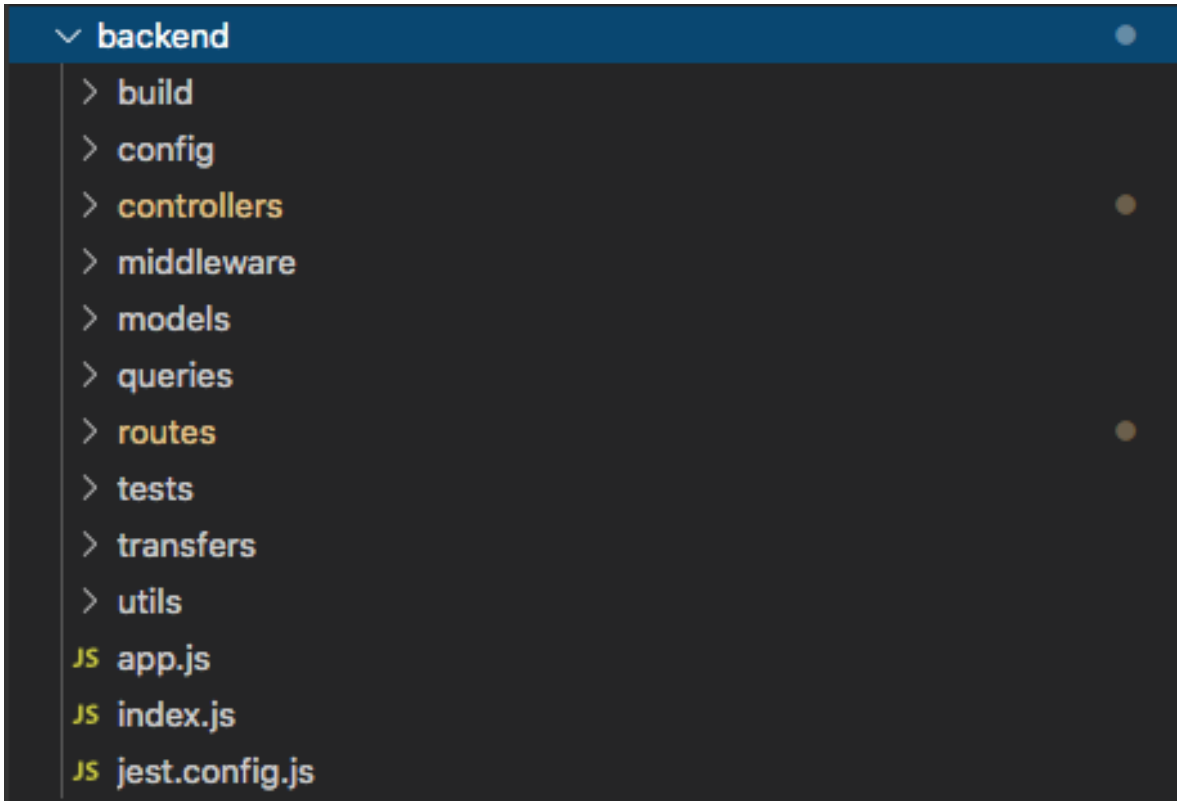


Kuva 20. Koko sovelluksen juurihakemisto.

Eslintin asetukset määritellään .eslintrc.js –tiedostossa. Tiedostot ja kansiot, joiden tyylimäärittelyyn ei haluta käyttää Eslintiä määritellään .eslintignore –tiedostossa. Procfile –tiedostossa määritellään Herokulle komento, jolla sovellus käynnistetään. Ympäristömuuttujat määritellään .env –tiedostossa. Ympäristömuuttujia ovat esimerkiksi tietokannan käyttäjätunnus ja salasana, sekä muut arkaluontoiset tiedot joita ei tietoturvasyistä voi julkaista sovelluksen muiden tiedostojen kanssa Githubissa. Tiedostot, joita ei haluta lisätä sovelluksen repositorioon Githubiin, määritellään .gitignore –tiedostossa. Tällaisia ovat esimerkiksi edellä mainittu .env –tiedosto, sekä sovelluksen riippuvuudet sisältävä node_modules –kansio. Tiedostot ja kansiot jotka poissuljetaan git-repositoriosta, näkyvät VS Coden kansionäkymässä harmaana.

4.4 Backend

Backendin tehtävänä on käsitellä frontendista palvelimelle tulevat pyynnöt, tallentaa dataa tietokantaan ja palauttaa sitä edelleen frontendiin käyttäjän nähtäväksi, sekä tarjoilla frontendin tuotantoversion staattiset tiedostot käyttäjän saataville.



Kuva 21. Backendin kansiorakenne.

Frontendin tuotantoversion tiedostot löytyvät *build* –kansiosta. Config-kansioon on määritelty asetukset, joilla sovellus ottaa yhteyden tietokantaan.

Controllers –kansiossa sijaitsevat funktiot, jotka huolehtivat erityyppisen datan tallennuksesta tietokantaan, sekä datan hausta tietokannasta. Kuvassa n nähtävillä listan aktiivisista ohjelmista palauttava funktio. Koko funktio on asyncHandler –middlewaren sisällä, jonka tehtävä on huolehtia siitä että asynkroninen koodi suoritetaan oikessa järjestyksessä.

```
// @desc    Get all active programs
// @route   GET /active
// @access  Private
exports.getAllActivePrograms = asyncHandler(async (req, res, next) => {
  const programs = await db.query(
    'SELECT * FROM playlist__program WHERE display = 1 order by name asc',
    {
      type: db.QueryTypes.SELECT
    }
  );
  if (programs.length === 0) {
    return next(new ErrorResponse('no programs found', 404));
  }
  res.status(200).json(programs);
});

```

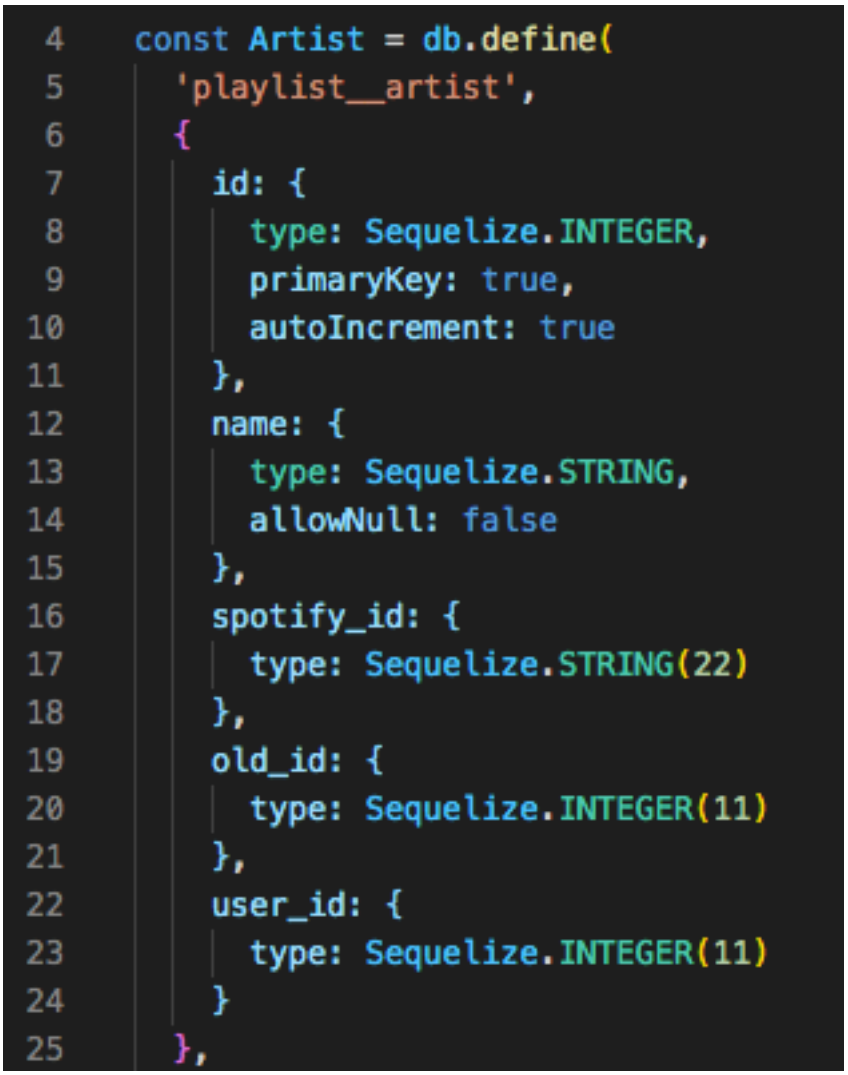
Kuva 22. Listan aktiivisista ohjelmista palauttava funktio.

Middleware-kansiossa on apufunktioita, joita käytetään laajalti ympäri sovellusta. Näitä ovat esimerkiksi backendiin tulevat http-pyynnöt konsoliin loggaava funktio, käyttäjän autentikoinnin varmistava funktio, sekä virhetapahtumia vastaavat http-koodit lähettävä funktio.

```
const logger = (req, res, next) => {
  console.log('Method:', req.method);
  console.log('Path:  ', req.path);
  console.log('Body:  ', req.body);
  console.log('---');
  next();
};

module.exports = logger;

```

Kuva 23. Http-pyynnöt konsoliin loggaava middleware-funktio.

Models-kansioon on määritelty datamallit tietokantaan tallennettavalle datalle. Malliin määritellään esimerkiksi tietokantaan tallennettavien kenttien nimet ja tietotyypit. Myös tiedon validointia on mahdollista ja suotavaa suorittaa malleissa. Esimerkiksi siten, että palvelin palauttaa virheviestin, mikäli tietokantaan tallennettavan käyttäjän sähköposti ei ole oikeassa formaatissa.



```
const Artist = db.define(
  'playlist__artist',
  {
    id: {
      type: Sequelize.INTEGER,
      primaryKey: true,
      autoIncrement: true
    },
    name: {
      type: Sequelize.STRING,
      allowNull: false
    },
    spotify_id: {
      type: Sequelize.STRING(22)
    },
    old_id: {
      type: Sequelize.INTEGER(11)
    },
    user_id: {
      type: Sequelize.INTEGER(11)
    }
  },
```

Kuva 24. Artist-malli.

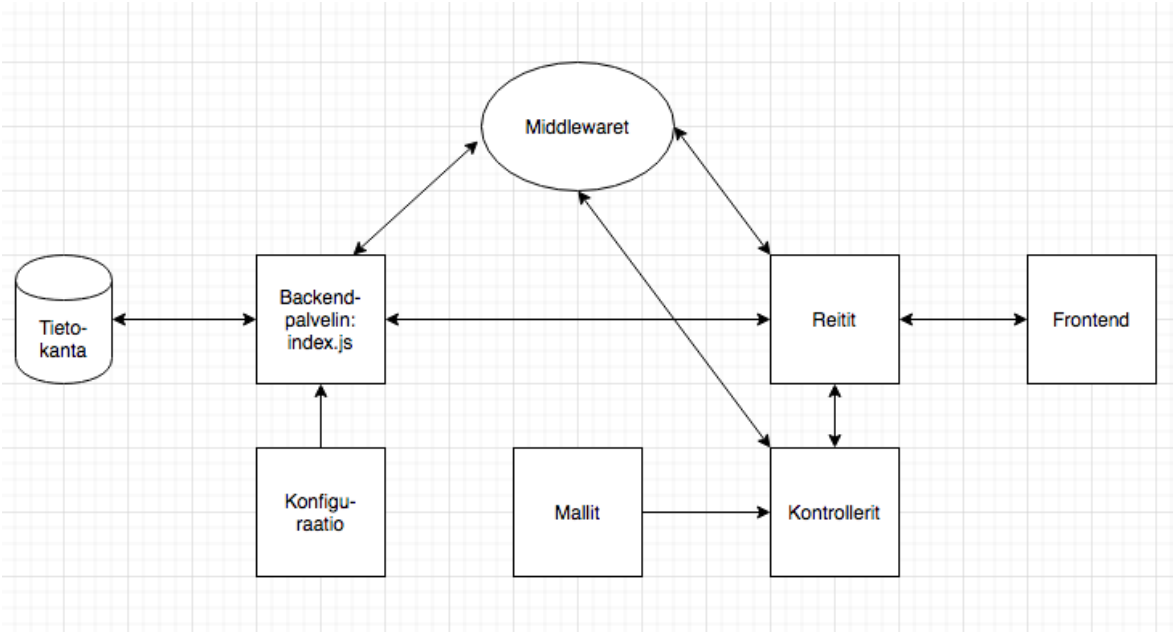
Routes-kansiossa ovat reitit, joita frontend kutsuu hakiessaan tai lähettäessään dataa backendiin. Reitit puolestaan kutsuvat asianmukaista kontrolleria ja tarpeen mukaan middleware-funktiota. Reitteihin myös määritellään, mistä http-protokollan mukaisesta reitistä on kyse. Tässä sovelluksessa on käytetty ainoastaan http-protollan yleisimpiä GET, POST, PUT ja DELETE –kutsuja. Samaa reittiä on myös mahdollista kutsua useammalla eri http-kutsulla. Esimerkiksi tiettyyn reittiin tehty GET-kutsu hakee dataa tietokannasta, kun taas POST-kutsu tallentaa sinne dataa.

Alla olevassa kuvassa on yhden albumin tiedot hakeva reitti. Mikäli selain lähettää reittiin GET-kutsun, reitti kutsuu *verifyUser*-middlewarea, jolla varmistetaan kirjautuneen käyttäjän validius. Tämän jälkeen kutsutaan *getOneAlbum*-funktiota, joka hakee tietokannasta reitin urlin parametrina välitetyn id-numeron mukaisen albumin. PUT-kutsu puolestaan päivittää id-numeron mukaisen albumin tiedot.

```
13  router
14    .route('/albumdetails/:id')
15    .get(verifyUser, getOneAlbum)
16    .put(verifyUser, updateAlbum);
```

Kuva 25. Yhden albumin tiedot hakeva reitti.

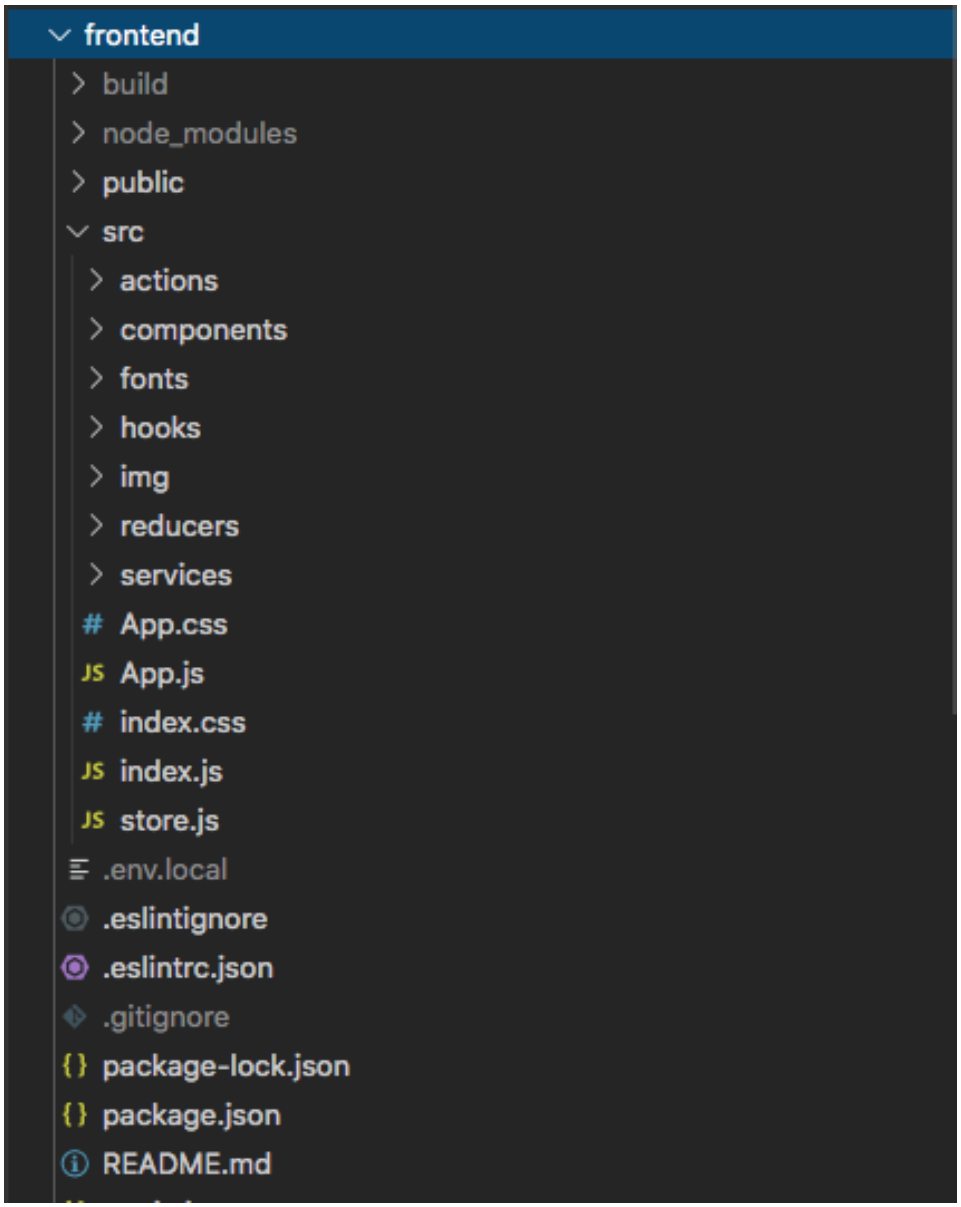
Sovellus kootaan yhteen app.js –tiedostossa, jossa Express –palvelimeen liitetään sovelluksen reitit ja middlewaret. Samassa tiedostossa hoidetaan myös tietokantaan kirjautuminen. Express-palvelin tuodaan index.js –tiedostoon, joka toimii sovelluksen käynnistyspisteenä. Alla olevassa diagrammissa on havainnollistettu backendin rakennetta lisää.



Kuva 26. Datan kulkua palvelimella havainnollistava diagrammi.

4.5 Frontend

Frontend tarjoaa sovelluksen käyttäjille käyttöliittymän, jolla he voivat syöttää, lukea ja muokata palvelimen hallinnoimaa dataa. Sovelluksen repositorion juuressa on *frontend*-kansio, jossa käyttöliittymän tiedostot sijaitsevat.



Kuva 27. Kuvankaappaus sovelluksen käyttöliittymän tiedosto- ja kansiorakenteesta.

Build-kansiossa on sovelluksen koottu, tuotantoon optimoitu versio. Node_modules –kansiossa on frontendin käyttämien kirjastojen asennustiedostot. Kaikki sovelluksen Git-repositorion ulkopuolelle jätetyt tiedostot näkyvät VS Coden tiedostoselaimessa harmaana. Yleinen käytäntö on, ettei koottuja sovellusversiota tai kirjastojen asennustiedostoja sisällytetä repositorioihin, vaan sovelluksen käyttön ottavat kehittäjät asentavat kirjastot aina uudelleen paikalliselle laitteelleen.

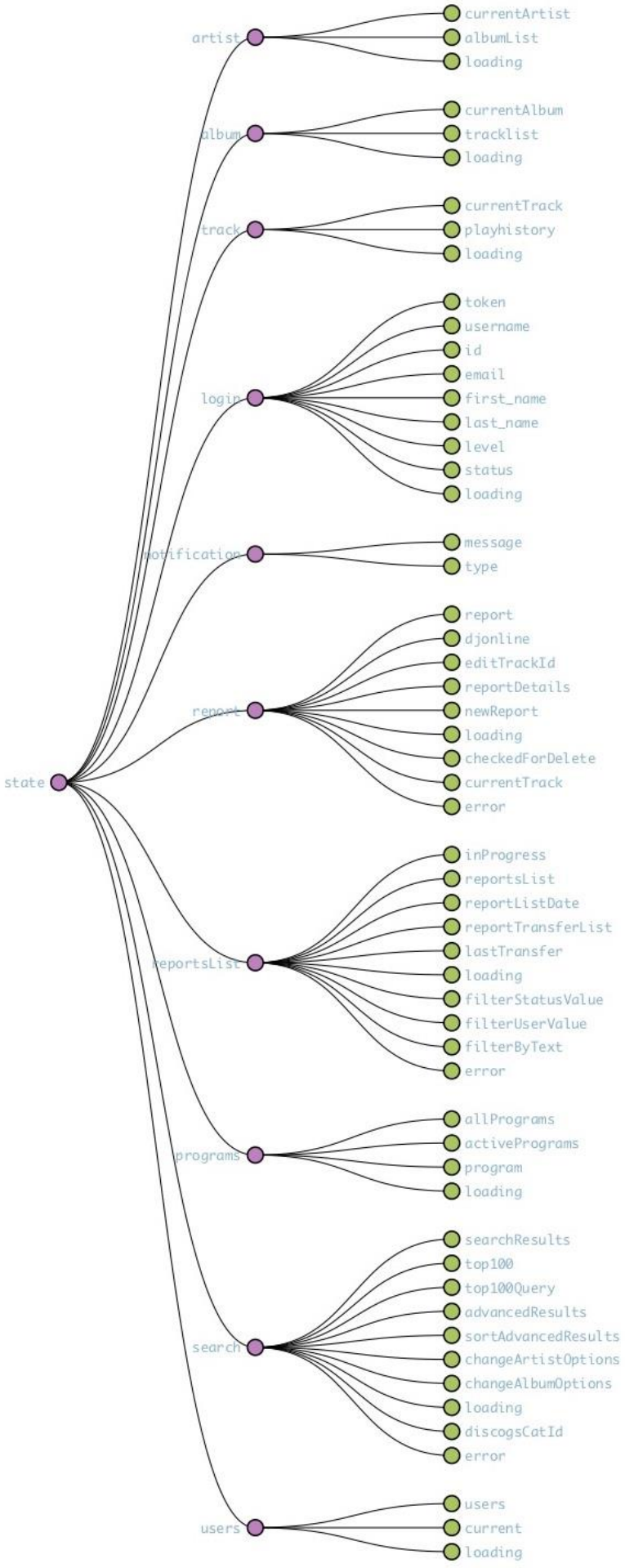
Public-kansiossa on html-tiedosto, jolla selaimet saavat sovelluksen auki. Frontend-kansion juuresta löytyy vielä koodin tyylimäärittelyt sisältävät eslint-tiedostot, yleistä tietoa sovelluksesta sisältävä README-tiedosto, sekä tiedot asennetuista kirjastoista sisältävä package.json –tiedosto.

Src-kansio puolestaan sisältää itse sovelluksen frontendin koodin. Src-kansion juuressa on index.js –tiedosto, joka käynnistää sovelluksen. Index-tiedosto puolestaan avaa App.js –tiedoston, johon on tuotu kaikki sovelluksen komponentit. Src-kansion alakansioista components pitää sisällään käyttäjälle näkyvät sovelluskomponentit kuten esimerkiksi eri sivut, lomakkeet ja navigointipalkin. Actions-, hooks-, reducers-, ja services –kansioiden tiedostot huolehtivat logiikasta jolla käyttöliittymä juttelee palvelimen kanssa, ja Redux-tilanhallinnasta huolehtivat tiedostot.

4.6 Redux tilanhallinta

Koko sovellukselle saatavissa oleva globaali tila on toteutettu Redux-kirjastolla. Reduxin ”tilapuun” haarat vastaavat pitkälti sovelluksen eri näkymiä. Artist-, Album-, ja Track –haaroissa ovat yksittäisen artistin, albumin ja kappaleen tiedot. Login-haara sisältää kirjautuneen käyttäjän tiedot ja kirjautumistokenin. Report-haara sisältää tiedot yksittäisen raportin kappaleista ja ajankohdasta. ReportList-haara sisältää tiedot haetun ajankohdan kaikista raporteista, sekä tulostettavan kuukausiraportin tiedoista. Search-haara sisältää tarkennetun haun hakutulokset, top100 –näkymän hakutulokset, sekä Discogsista haetun levykoodin. Programs- ja users –haarat sisältävät tiedot kaikista ohjelmista ja käyttäjistä. Notification-haara sisältää vain käyttäjälle lähetettävän huomautusviestin, jolla kerrotaan esimerkiksi onnistuneesta kappaleen lisäyksestä tai raportin päivämäärän muokkauksesta. Huomautuksen tyyppinä on joko succes tai fail, jolloin viesti käyttäjälle näkyy tyyppiä vastaavalla värillä.

Lähes jokaisella haaralla on erikseen loading-haara, jonka arvona on totuusarvo true tai false. Loading-tila on oletuksena false. Silloin kun sovellus hakee tai tallentaa tietoja tietokannasta, loading-tilaksi muuttuu true, ja käyttöliittymä näyttää käyttäjälle latauskuvakkeen. Kun lataus on valmis, loading-tila vaihtuu takaisin falseksi ja latauskuvake häviää näkymästä. Sovellusta refaktoroidessa olisi syytä harkita voitaisiinko tällaiselta toistolta välttyä lisäämällä tilaan erikseen oma loading –haara, joka hoitaisi koko sovelluksen lataustilasta huomauttamisen.


state
artist
currentArtist
albumList
loading
album
currentAlbum
tracklist
loading
track
currentTrack
playhistory
loading
login
token
username
id
email
first_name
last_name
level
status
loading
notification
message
type
report
report
djonline
editTrackId
reportDetails
newReport
loading
checkedForDelete
currentTrack
error
reportsList
inProgress
reportsList
reportListDate
reportTransferList
lastTransfer
loading
filterStatusValue
filterUserValue
filterByText
error
programs
allPrograms
activePrograms
program
loading
search
searchResults
top100
top100Query
advancedResults
sortAdvancedResults
changeArtistOptions
changeAlbumOptions
loading
discogsCatId
error
users
users
current
loading

Kuva 28. Kuvakaappaus Reduxin ”tilapuusta” kehittäjätyökaluissa.

4.7 Ulkoiset tietolähteet – DJOnline, Discogs

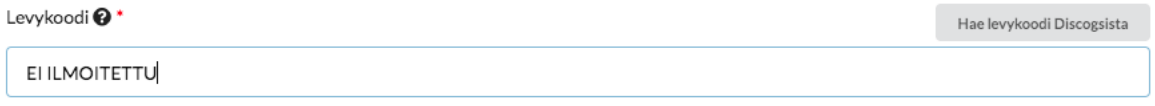
Tuotantoon mennessään sovellus on integroitu toimimaan kahden ulkoisen tietolähteen kanssa. Radio Helsingin playout-ohjelmistona toimivan DJOnlinen soittimen rajapinnasta voidaan hakea soittotapahtumat päivämäärän ja kellonajan mukaan rajattuna. Tämä toiminnallisuus on suurin syy siihen, miksi sovellus haluttiin alun perin tehdä.


```
- 76: {
      date: "2019-01-20 01:43:13",
      id: "243468",
      composer: "",
      artist: "Hearing, The",
      artist_fi: null,
      additional_artist: null,
      conductor: null,
      song: "Backwards",
      song_fi: null,
      label: "Solina Records",
      matrix: "SOL-048",
      isrc: "FISR61600006",
      album: "Backwards",
      side: "1",
      tracknumber: "1",
      length: "4:37",
      year: "2016",
      recording_country: "FI"
  },
```


Kuva 29. DJOnlinen rajapinnan palauttamaa dataa.

Käyttäjä syöttää päivämäärän ja kellonajan joiden ajalta soittotapahtumat halutaan hakea. Sovellus parsii yllä olevan kaltaisesta json-datasta määritellylle aikavälille sopivat kappaleet, lisää ne avoinna olevaan raporttiin, sekä tarpeen mukaan tallentaa tiedot uusista kappaleista tietokantaan.

Uutta kappaletta lisätessään käyttäjällä on mahdollisuus hakea lisättävän kappaleen albumin levykoodi Discogsista. Frontend lähettää Discogsin rajapintaan kyselyn, joka saa parametreikseen haettavan albumin nimen ja esittäjän. Mikäli kyselyllä on tuloksia, levykoodi lisätään uuden kappaleen lisäävään lomakkeeseen. Mikäli kyseistä albumia ei löydy Discogsin tietokannasta, lomakkeen levykoodi-kenttään täytetään EI ILMOITETTU.



Kuva 30. Lisää kappale –lomakkeen Hae levykoodi Discogsista –kenttä.

4.8 Kuukausiraportin tulostus

Kuukausiraportin koostava funktio hakee tietokannasta kaikki valmiisiin raportteihin lisätyt kappaleet tietyn kuukauden mittaiselta aikaväliltä. Tekijänoikeusjärjestöt haluavat raportin yhtenä tekstitiedostona, jossa jokainen soitettu kappale on merkattu omalle rivilleen. Rivin mitan täytyy olla tasan 460 merkkiä. Raportin formaattia on helpompi hahmoittaa kuvalla:

```
2010010000  0600  2592CLOSE                                                0003270001
2010010000  0600  2592SLEEPSTORM                                           0003500001
2010010000  0600  2592LOPPUSOITTO                                          0001220001
2010010000  0600  2592HELSINKI ‹BER ALLES                                  0003080001
2010010000  0600  2592GRAVION                                              0004230001
2010010000  0600  2592HOIDA PUUT                                           0003040001
2010010000  0600  2592JOO, JOO, Mƒ RAKASTAN SUA                            0004000001
2010010000  0600  2592BEACON                                               0005140001
2010010000  0600  2592EPILOGI (REMASTERED 2016)                            0002540001
2010010000  0600  2592KUKA Sƒ OOT?                                         0004260001
2010010000  0600  2592SIKALANHOITAJA                                       0004120001
2010010000  0600  2592HIHNALLA                                             0004290001
2010010000  0600  2592JACKPOT                                              0003020001
2010010000  0600  2592EI SANKARIAINESTA                                    0003420001
2010010000  0600  2592HIIDEN HURTTA                                        0003410001
2010010000  0600  2592VALEHTELE MULLE (FEAT. YEBOYAH)                      0002570001
2010010000  0600  2592TOAST WORLD                                          0003260001
```

Kuva 31. Kuvankaappaus tulostetusta kuukausiraportista tekijänoikeusjärjestöjen vaatimassa formaatissa.

Kuvankaappauksessa rivin ensimmäiset kuusi merkkiä kertovat lähetysajankohdan vuoden, kuukauden ja päivämäärän kaksinumeroisessa formaatissa. Seuraavat kuusi merkkiä kertovat kellonajan ohjelman alkaessa, ja seuraavat kuusi merkkiä kellonajan ohjelman päättyessä. Rivin positiot 11-12 ja 17-18 jätetään tyhjiksi. Rivin positioista 19-22 käy ilmi kanavakohtainen esitysryhmätunnus.

Rivin positioissa 23-72 tulee olla kappaleen nimi. Mikäli kappaleen nimi on vähemmän kuin 50 merkkiä, tulee riville lisätä välilyöntejä positioon 72 asti, jotta seuraava tietue alkaa oikeassa kohdassa positiossa 73. Positioon 73-74 lisätään apukentän arvo 00. Positiossa 75-78 ilmoitetaan kappaleen kesto muodossa MMSS. Positioissa 79-82 ilmoitetaan soittokerrat nelinumeroisena lukuna. Yllä olevassa esimerkissä jokainen kappale on soinut kerran, eli raporttiin on merkattu 0001.

Tietojen lisääminen tekstitiedostoon jatkuu samaan tyyliin, kunnes jokaisella rivillä on tasan 460 merkkiä. Jokaisen rivin loppuun tulee lisätä uutta riviä merkitsevät loppumerkit CR LF. Tekstitiedosto tulee enkoodata Windows 1252-merkistöllä.
(Teosto/Gramex raportointiohje)

4.9 Julkaisu

Sovellus on julkaistu Herokussa, joka on suosittu pilvipalvelu web-sovellusten julkaisuun. Gitin tapaan Heroku käyttää komentorivityökaluja. Gitiä käyttävät repositoriot voidaan julkaista Herokussa melko vähäisellä konfiguraatiolla. (Heroku, Heroku.)

Sovellus julkaistaan komennolla npm run deploy:full. Komento suorittaa ensin frontendin koodin koostamisen tuotantomuotoon. Tämän jälkeen koodi lisätään git-repositorioon, sekä Heroku-haaraan. Heroku suorittaa automaattisesti build komennon, joka tekee seuraavat asiat:

- komentorivillä siirtyminen backend –kansioon
- luodaan transfers-kansio, jonne kuukausiraporttien tekstitiedostot tallennetaan
- siirrytään takaisin projektin juureen, ja siellä frontendin kansioon.
- Frontend-kansiossa asennetaan projektin riippuvuudet, ja suoritetaan build komento, jolla koodi koostetaan tuotantomuotoiseksi.

```
"scripts": {
  "build": "cd backend && mkdir transfers && cd .. && cd frontend && npm install --only=dev && npm
  install && npm run build",
  "build:ui": "cd frontend && npm run build",
  "deploy": "git push heroku master",
  "deploy:full": "npm run build:ui && git add . && git commit -m uibuild && git push -u origin
  master && npm run deploy",
  "start": "cross-env NODE_ENV=production node backend",
  "start:test": "cross-env NODE_ENV=test node backend",
  "test": "cross-env NODE_ENV=test jest --verbose --detectOpenHandles
  --testPathIgnorePatterns=frontend/cypress/integration/examples --runInBand",
  "watch": "cross-env NODE_ENV=development nodemon backend"
```

Kuva 32. Kuvankaappaus skripteistä package.json –tiedostossa.

Kun build-komentoon sisällytetään transfers-kansion luominen, sekä frontendin build-kansion luominen, ne voidaan jättää hyvien tapojen mukaisesti pois Git-repositoriosta. (What not to commit, dart.dev)

5 Testaus

Luvussa kerrotaan ohjelmien testauksesta yleisesti, raportointisovelluksen palvelimen integraatiotesteistä, sekä käyttöliittymän end-to-end -testeistä.

Kaikki ohjelmiston viat syntyvät tuotekehityksen aikana, tosin myös ohjelmiston käyttöympäristössä tapahtuvat muutokset voivat aiheuttaa ohjelmiston ”vikaantumisen” sen käyttöaikana toiminnallisten riippuvuuksien kautta. Ohjelmiston testaamisen tavoitteita ovat esimerkiksi ohjelman suorittaminen virheiden löytämiseksi, ohjelman suorittaminen laadun mittaamiseksi, ja ohjelman suorittaminen luottamuksen lisäämiseksi. Myös ohjelmiston rakenteiden ja koodin staattinen analyysi voi olla osa testausta. Raportointisovelluksen kehityksessä staattinen analyysi tapahtuu ESLint-kirjaston avulla, joka varoittaa kehittäjää virheistä ennalta määrättyjen sääntöjen mukaan.
(Ohjelmistotestauksen perusteita, Helsingin Yliopisto, Antti-Pekka Tuovinen)

Koodin toiminnallisuuden varmistamisen lisäksi testeistä on muutakin hyötyä. Ohjelmistoprojektiin uutena kehittäjänä mukaan tuleva voi testejä lukemalla päätellä helpommin miten ohjelman tulisi toimia. Jotta koodi olisi helposti testattavissa, tulee sen olla riittävän yksinkertaista. Esimerkiksi monimutkaiselle funktiolle on vaikeaa, ellei mahdotonta kirjoittaa kattavia testejä. Ennen uuden ominaisuuden koodaamista kehittäjän on hyvä miettiä miten ominaisuutta voi testata järkevästi, niin toteutuskin pysynee selkempänä ja yksinkertaisempana.

5.1 Yksikkö- & Integraatiotestit

Palvelimen testaukseen käytetään Facebookin kehittämää Jest –testauskirjastoa, joka on toiminnaltaan ja syntaksiltaankin hyvin samankaltainen kuin testikirjastojen entinen kuningas Mocha. Jest on raportointisovellukseen luonteva valinta, sillä se soveltuu hyvin palvelinten testaamiseen, mutta on erinoimainen React-sovellusten testaamisessa.

Yksikkötesteissä testataan ohjelmiston pienien yksittäisten osien, kuten funktioiden toimintaa. Raportointisovelluksessa ei ole paljoakaan järkevää yksikkötestattavaa, mutta testikirjaston toimivan asennuksen testaamiseksi sovellukseen on kirjoitettu pari yksikkötestiä Helsingin yliopiston Full Stack Open –kurssin oppeja noudattaen.
(Full Stack Open – Node-sovellusten testaaminen, Helsingin yliopisto)

```
const palindrome = string => {
  return string
    .split('')
    .reverse()
    .join('');
};
```

Kuva 33. Palindromi –funktio.

```
const { palindrome } = require('../utils/for_testing');

test('palindrome of react', () => {
  const result = palindrome('react');

  expect(result).toBe('tcaer');
});
```

Kuva 34. Palindromi –funktiota testaava testi.

Ensin kirjoitetaan funktio nimeltä palindromi, joka saa parametrinaan merkkijonon. Funktio muuntaa merkkijonon yksittäiset kirjaimet sisältäväksi taulukoksi, kääntää järjestyksen ja palauttaa uudelleen yhteen liitetyn merkkjonon. Itse testissä käytetään Jestin test-funktiota. Funktion ensimmäisenä parametrina kuvaillaan mitä testin oletetaan tekevän. Toisena parametrina on funktio, joka määrittää testin toiminnallisuuden. Expect-metodissa kerrotaan mitä testin tuloksen odotetaan olevan. Testien suorituksen jälkeen tuloksen näkee konsolissa.

```
PASS backend/tests/palindrome.test.js
  ✓ palindrome of a (2ms)
  ✓ palindrome of react (1ms)
  ✓ palindrome of saippuakauppias (1ms)
```

Kuva 35. Testien läpäisystä kertova viesti konsolissa.

Sovelluksen palvelimen rajapintojen ja tietokannan toimintaa yhdessä testataan useita sovelluksen komponentteja yhtaikaa käyttävillä integraatiotesteillä. Tätä varten tietokannasta luodaan testiversio, joka vastaa rakenteeltaan ja osin sisällöltäänkin sovelluksen varsinaista tietokantaa. Ennen integraatiotestien suorittamista on tärkeää varmistaa, että tietokanta alustetaan aina samalla tavalla, jotta testien tulokset pysyvät johdonmukaisina. Jest-kirjastolla on tätä varten esimerkiksi *beforeEach* –metodi, joka suoritetaan aina ennen testien suorittamista. Metodin voi käskeä ensin tyhjentämään testitietokanta, ja lisätä sinne dataa jota testien oletetaan käyttävän.

```
11  beforeAll(async () => {
12    await User.destroy({
13      where: {},
14      truncate: true,
15    });
16
17    await User.bulkCreate(users);
18
19    const loginCredentials = {
20      username: 'teemu',
21      password: 'test',
22    };
23    const req = await api.post('/api/login').send(loginCredentials);
24    token = req.body.token;
25  });
```

Kuva 36. Ennen testejä suoritettava beforeAll –metodi, joka tyhjentää testitietokannan käyttäjistä, lisää tietyt käyttäjät uudelleen, sekä kirjaa käyttäjän sisään.

Kun tietokanta on alustettu testien odottamalla tavalla, voidaan suorittaa itse testit. Esimerkkitapauksissa tehdään kutsu */users* –rajapintaan, jonka ilman kirjautumistokenia odotetaan palauttavan virhekoodi 401. Toinen testi tekee kutsun samaan rajapintaan kirjautumistokenin kanssa, jolloin tulokseksi odotetaan lista käyttäjistä, sekä onnistuneesta kutsusta kertova tilakoodi 200.

```
40  it('should require authorization', async () => {
41    await api.get('/api/users').expect(401);
42  });
43
44  it('should return list of users', async () => {
45    await api
46      .get('/api/users')
47      .set('Authorization', `bearer ${token}`)
48      .expect(200);
49  });
```

Kuva 37. /users-rajapintaan tehtäviä testikutsuja.

```
PASS backend/tests/1_user.test.js (7.236s)
  ✓ successful login returns 200 (141ms)
  ✓ should require authorization (30ms)
  ✓ should return list of users (36ms)
  ✓ should return an user with an existing id (61ms)
  ✓ should return an error message and status 404 with a non-existing user id (68ms)
  ✓ should return an error message and status 400 when adding an existing username (63ms)
  ✓ a user can be deleted (98ms)
  ✓ user can be added (154ms)
```

Kuva 38. Konsoliviesti testien läpäisystä.

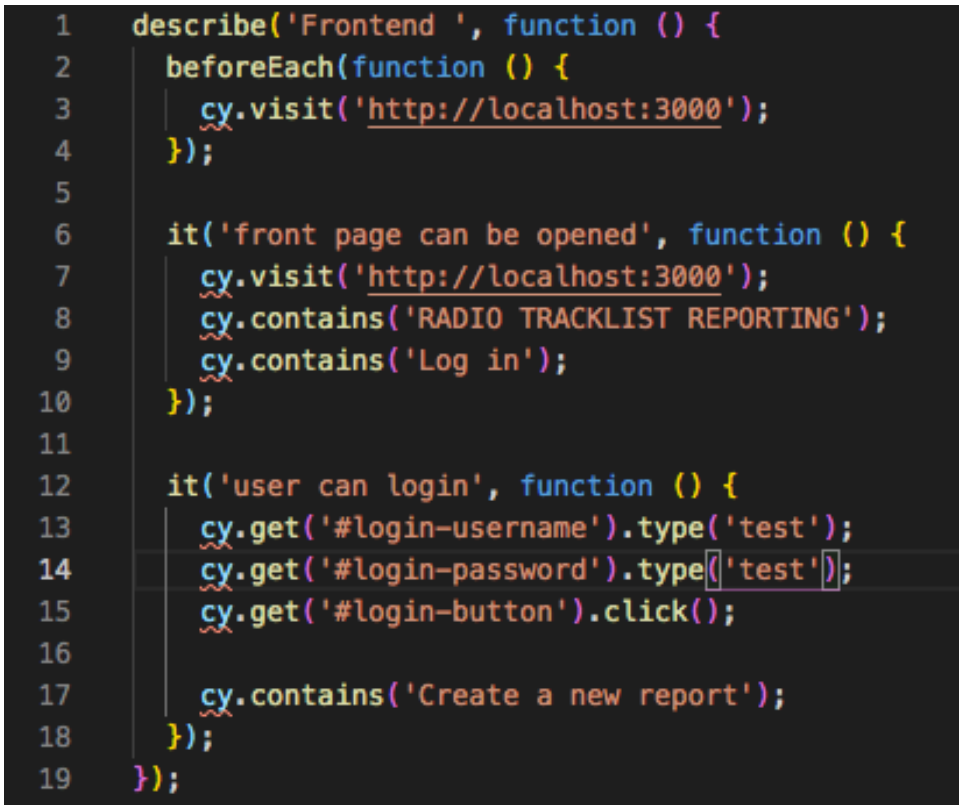
5.2 Käyttöliittymän testaaminen

Reactilla tehtyjen käyttöliittymien testaamiseen on monia tapoja. Facebookin tekemän sovelluskehyksen testaaminen onnistuu luonnollisesti kätevästi Facebookin tekemällä Jest-kirjastolla. Yksittäiset komponentit voidaan renderöidä react-testing-library –kirjaston avulla. Jest toimii käyttöliittymätesteissä samaan tyyliin palvelimen testien kanssa, eli test-metodille annetaan parametreina testin nimi, ja React-komponentin renderöivä funktio. Expect-lohkossa testille kerrotaan mitä sen odotetaan sisältävän.

Käyttöliittymää, ja järjestelmää kokonaisuutena on mahdollista testata end-to-end –testeillä. E2E testit ovat potentiaalisesti kaikkein hyödyllisin testikategoria, sillä ne tutkivat järjestelmää saman rajapinnan kautta kuin todelliset käyttäjät. E2E-testeihin liittyy myös ikäviä puolia. Niiden konfigurointi on haastavampaa kuin yksikkö- ja integraatiotestien. E2E-testit ovat tyypillisesti myös melko hitaita ja isommassa ohjelmistossa niiden suoritusaika voi helposti nousta minuutteihin, tai jopa tunteihin. Tämä on ikävää sovelluskehityksen kannalta, sillä sovellusta koodatessa on erittäin hyödyllistä pystyä suorittamaan testejä mahdollisimman usein koodin regressioiden varalta.
(Full Stack Open – End to end –testaus, Helsingin yliopisto)

Raportointisovelluksen end-to-end –testit on tehty Cypress –kirjaston avulla. Cypress on kasvattanut nopeasti suosiotaan viimeisen reilun vuoden aikana. Cypressin toimintaperiaate poikkeaa radikaalisti useimmista E2E-testaukseen sopivista kirjastoista, sillä Cypress-testit ajetaan kokonaisuudessaan selaimen sisällä. Muissa lähestymistavoissa testit suoritetaan Node-prosessissa, joka on yhteydessä selaimeen ohjelmointirajapintojen kautta. Toisin kuin yksikkötestit, Cypress-testit voidaan sijoittaa joko frontendin tai backendin repositorioon, tai vaikka kokonaan omaan repositorioonsa. Testejä suoritettaessa sekä palvelimen että käyttöliittymän täytyy olla käynnissä, jotta selaimeen aukeava Cypress pääsee niihin käsiksi.

Testien kirjoittaminen on hyvin samankaltaista kuin palvelimen Jest-kirjastoa käyttävien testien. Describe-lohkon sisään kirjoitetaan, mitä testien odotetaan tekevän. Cy.visit –metodilla kerrotaan mihin osoitteeseen Cypressin halutaan menevän, cy.contains –metodille kerrotaan mitä sivulla odotetaan lukevan. Cy.get –metodilla on mahdollista etsiä tekstinsyöttökenttiä ja nappuloita. Testi menee läpi, mikäli kirjautumisen jälkeen testi löytää sivulta sovelluksen etusivulla esiintyvän Create a new report –tekstin.



```
describe('Frontend ', function () {
  beforeEach(function () {
    cy.visit('http://localhost:3000');
  });

  it('front page can be opened', function () {
    cy.visit('http://localhost:3000');
    cy.contains('RADIO TRACKLIST REPORTING');
    cy.contains('Log in');
  });

  it('user can login', function () {
    cy.get('#login-username').type('test');
    cy.get('#login-password').type('test');
    cy.get('#login-button').click();

    cy.contains('Create a new report');
  });
});
```

Kuva 39. Kirjautumista käyttöliittymässä testaavan end-to-end –testin koodi.

6 Sovelluksen TypeScript-versio

TypeScript on Microsoftin luoma ja ylläpitämä johdannainen JavaScriptista. TypeScript on itse asiassa kaksi erillistä, mutta toisilleen sukua olevaa teknologiaa - ohjelmointikieli ja kielen kääntäjä. Ohjelmointikielenä TypeScript on ominaisuuksiltaan rikas, staattisesti tyypitetty kieli joka lisää olio-ohjelmoinnista tuttua kapasiteettia JavaScriptiin. Koodin kääntäjä muuntaa TypeScript-koodin natiiviksi JavaScriptiksi, ja tukee ohjelmoijaa vähävirheisemmän koodin kirjoituksessa. TypeScriptiä käyttämällä ohjelmoija voi kirjoittaa koodia, jota on helpompi ymmärtää ja refaktoroida. TypeScriptin käyttö lisää kurinalaisuutta ohjelmistoprojektiin pakottamalla kehittäjät korjaamaan virheet jo kehitysvaiheessa. TypeScript on kehitysaikainen teknologia. TypeScriptissä ei ole ajonaikaisia komponentteja, eikä TypeScript-koodia ikinä suoriteta sellaisenaan. TypeScript-kääntäjä kääntää koodin tavalliseksi JavaScriptiksi, joka puolestaan suoritetaan joko selaimessa tai palvelimella.

JavaScript-ohjelmonintikieli kehittettiin vuonna 1995, alun perin käytettäväksi osana Netscape-selainta. Siitä lähtien JavaScriptin suosio on kasvanut tasaisesti ja nykyään sitä käytetään myös palvelinten ja työpöytäsovellusten kehitykseen. JavaScriptin joustavuus on kuitenkin osoittautunut sekä eduksi että ongelmaksi. Kun sovellusten koko kasvaa suuremmaksi ja suuremmaksi, JavaScriptin rajoitukset ja heikkoudet nostavat päätään. Suurempien sovellusten kehityksessä on suuremmat tarpeet kuin selainpohjaisessa JavaScript-kehityksessä. Toisin kuin JavaScript, lähes kaikki suurempien sovellusten kehitykseen käytetyt ohjelmointikielet kuten Java, C++ ja C# sisältävät staattisen tyypityksen ja olio-ohjelmointimahdollisuudet.

Jokainen ohjelmointikieli sisältää ja käyttää tyyppejä. Tyyppi voidaan mieltää joukoksi sääntöjä, jotka kuvaavat tapoja joilla objekteja koodissa voidaan käyttää uudelleen. JavaScript on dynaamisesti tyypitetty kieli. Dynaamisesti tyypitetyissä ohjelmointikielissä luoduille muuttujille ei tarvitse antaa tyyppiä, ja muuttujan voi osoittaa uudelleen olemaan jotain muuta kuin sille alun perin annettua tyyppiä. Tämä ominaisuus lisää suuresti joustavuutta JavaScriptiin, mutta on samalla lukuisien virheiden lähde. TypeScript puolestaan käyttää staattista tyypitystä. Staattinen tyypitys pakottaa kehittäjän ilmoittamaan kunkin muuttujan tyypin jo muuttujan alustusvaiheessa. Tämä vähentää epäselvyyttä ja eliminoi kahden eri tyypin välillä tapahtuvat käännösvirheet.
(Choi 19.4.2021)

Raportointisovelluksen TypeScript-versiossa määritellään tyypit, joita tietokannasta haettavalla sekä sinne tallennettavalla datalla oletetaan olevan. Useita kenttiä sisältävää tyyppimäärittelyä kutsutaan interfaceksi.

```
68  export interface ReportDetails {
69    program_name: string;
70    program_no: number | null;
71    program_dj: string;
72    program_date: string;
73    program_start_time: string;
74    program_end_time: string;
75    id: number;
76    program_id: number;
77    rerun: number | null | undefined;
78    status: number;
79    user_id: number;
80    username: string;
81    first_name: string;
82    last_name: string;
83  }
```

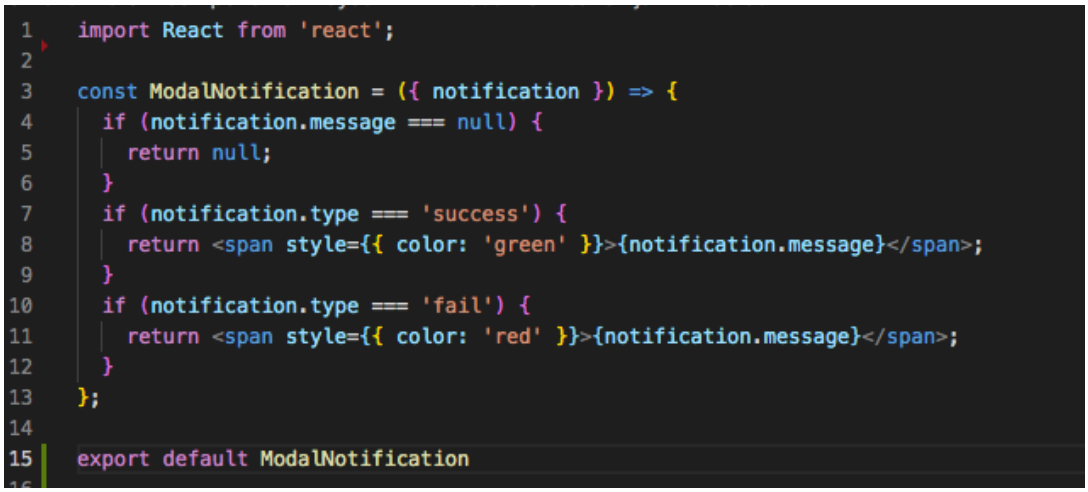
Kuva 40. ReportDetails-interfacen määritys. Kuvankaappaus VS Code –ohjelmasta.

Esimerkiksi tietokannasta haettavan yksittäisen raportin tietojen odotetaan sisältävän tiedot muun muassa ohjelman nimestä, numerosta, dj:sta, päivämäärästä, aloitus- ja lopetusajasta ja raportin tilasta. Mikäli tietueella on useampi | -merkillä eroteltu tyyppi, kyseisen tietueen tyyppi voi olla mikä vain annetuista. Mikäli sovelluksen saama data tai datan tyyppi poikkeaa ilmoitetusta, saa käyttäjä virheilmoituksen. Tyypin voi myös ilmoittaa valinnaisena laittamalla tyypin nimen eteen kysymysmerkin.

```
Failed to compile.

/Users/MacBookPro/Sites/playlist_typescript/frontend/src/components/users/EditUserModal/index.tsx
TypeScript error in /Users/MacBookPro/Sites/playlist_typescript/frontend/src/components/users/EditUserModal
/index.tsx(23,43):
Property 'username' does not exist on type 'User'.  TS2339
```

Kuva 41. Tekstieditori ilmoittaa tahallaan tuotetusta virheestä, jossa interfacesta User on poistettu tyyppi username. Kuvankaappaus VS Code –ohjelmasta.

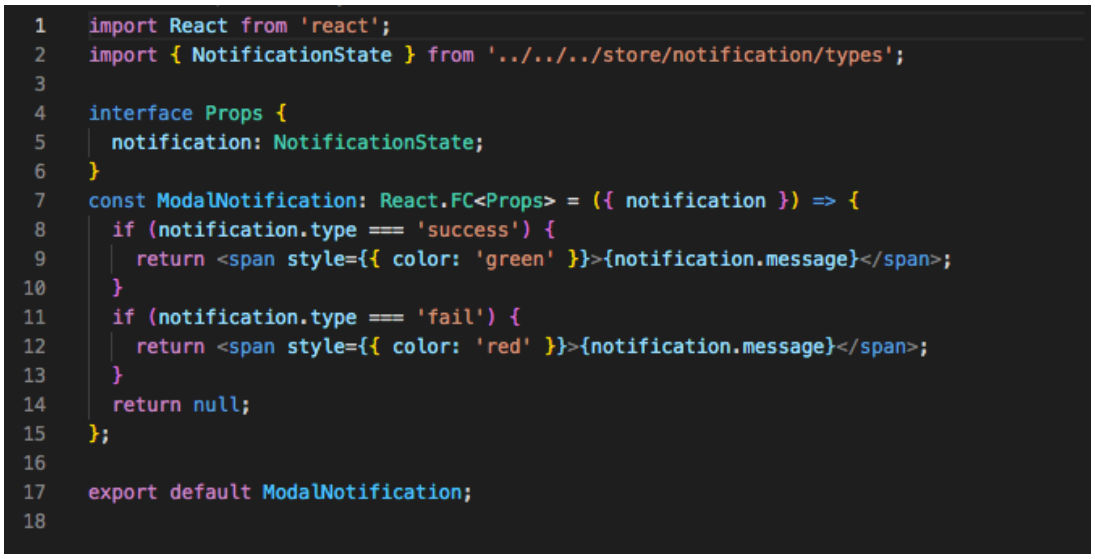
Alla olevassa kuvankaappauksessa on sovelluksen JavaScript-version yksinkertaisemmasta päästä oleva, lomakkeiden täyttöön liittyvissä ilmoituksissa käytetty ModalNotification –komponentti. Komponentille välitetään notification-olio, jonka tyypistä riippuen käyttäjälle näytetään ilmoitus onnistuneesta tai epäonnistuneesta toiminnosta.



```
import React from 'react';

const ModalNotification = ({ notification }) => {
  if (notification.message === null) {
    return null;
  }
  if (notification.type === 'success') {
    return <span style={{ color: 'green' }}>{notification.message}</span>;
  }
  if (notification.type === 'fail') {
    return <span style={{ color: 'red' }}>{notification.message}</span>;
  }
};

export default ModalNotification
```

Kuva 42. Sovelluksen JavaScript-version ModalNotification-komponentti. Kuvankaappaus VS Code –ohjelmasta.

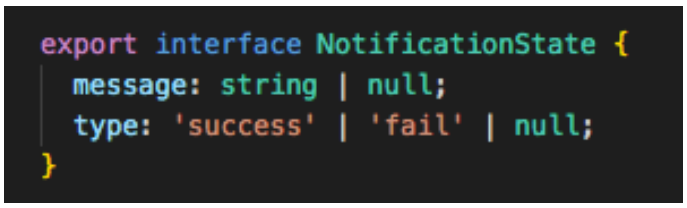
Komponentin TypeScript –versio näyttää lähes samalta muutamaa riviä lukuunottamatta.



```
import React from 'react';
import { NotificationState } from '../../../store/notification/types';

interface Props {
  notification: NotificationState;
}
const ModalNotification: React.FC<Props> = ({ notification }) => {
  if (notification.type === 'success') {
    return <span style={{ color: 'green' }}>{notification.message}</span>;
  }
  if (notification.type === 'fail') {
    return <span style={{ color: 'red' }}>{notification.message}</span>;
  }
  return null;
};

export default ModalNotification;

```

Kuva 43. Sovelluksen TypeScript-version ModalNotification-komponentti. Kuvankaappaus VS Code –ohjelmasta.

Komponentin TypeScript-versiossa komponentille kerrotaan propsina välitettävän datan olevan NotificationState –tyyppiä. Mikäli data poikkeaa määritellystä tyypistä, TypeScript antaa kehittäjälle virheilmoituksen.



```
export interface NotificationState {
  message: string | null;
  type: 'success' | 'fail' | null;
}
```

Kuva 44. Erillisestä tiedostosta löytyvä, ilmoituksen tilan määrittelevä NotificationState -tyyppimäärittely. Kuvankaappaus VS Code –ohjelmasta.

7 Aikataulun toteutuminen, pohdinta & jatkokehitys

Toimeksiantosopimusta tehdessä vuoden 2018 lopulla sovelluksen valmistumistavoitteeksi sovittiin vuoden 2019 loppuun mennessä. Sovellus oli pääpiirteissään valmis vuoden 2019 joulukuun puoliväliin mennessä. Koodi oli siinä vaiheessa vielä hieman viimeistelemätöntä. Myöskään laajempaa testausta sovellukselle ei vielä tuohon mennessä ollut tehty. Koska toimeksiantajan vanha kappaletietojen raportointisovellus oli edelleen toimintakunnossa, ei ollut perusteltua kiirehtiä uuden sovelluksen julkaisun kanssa. Toimeksiantaja oli asiasta samaa mieltä.

Käytin siis vielä vuoden 2020 ensimmäiset kuukaudet sovelluksen viimeistelyyn ottamalla käyttöön AirBnb:n Eslint-kirjaston, jonka avulla koodista tuli luettavampaa ja yhdenmukaisempaa. Myös julkaisuputkea tuli virtaviivaistettua, ja sovelluksen kehitysversiolle luotiin oma Heroku-alusta, jolla koodimuutoksia on mahdollista testata riskeeraamatta tuotannossa olevan sovelluksen toimivuutta. Sovelluksen ensimmäinen tuotantoversio julkaistiin 13.3.2020.

Sovelluksen testikattavuutta olisi voinut lisätä paljonkin. Parhaimmillaan / pahimmillaan sovelluksen testien kirjoittamiseen saa kulumaan enemmän aikaa kuin itse sovelluksen tekemiseen. Julkaisuvaiheessa sovellukseeen oli tehty vain muutama yksinkertainen palvelimen integraatiotesti, sekä sisäänkirjautumista käyttöliittymän kautta testaava end-to-end –testi. Testejä kirjoittaessa tuli ilmi, että palvelimen koodia tulisi refaktoroida yksinkertaisemmaksi, jotta testauksen kohteena oleva toiminnallisuus pysyisi mahdollisimman yksinkertaisena. Esimerkiksi useista tietokantatapahtumista riippuvaiset controller-funktiot ovat sellaisenaan melko monimutkaisia testattavia. Käyttöliittymän toiminnallisuudessa on myös vielä paljon end-to-end –testattavaa.

Kappaleiden metatietojen etsimisessä voisi hyödyntää entisestään kolmansien osapuolien rajapintoja. Esimerkiksi Spotifyn rajapinnasta löytyy kattavasti kappaleiden ISRC-koodeja, sekä julkaisujen levykoodeja, jotka molemmat ovat Teostolle ja Gramexille tärkeitä tietoja korvausten tilityksessä. Sovelluksen palvelin voisi tutkia taustalla, mikäli lisättävästä kappaleesta löytyy tietoja kolmannen osapuolen rajapinnasta. Uusien, julkaisemattoimien kappaleiden osalta tosin ongelmaksi muodostuu, että niiden tietoja ei ole lisätty mihinkään tietokantaan siinä vaiheessa kun kappaletta syötetään Radio Helsingin järjestelmään. Tällöin sovellus voisi kuukausiraporttia koostaessaan tutkia mikäli jonkin Teostoon ja Gramexiin lähetettävän kappaleen metatietoja puuttuu, ja tarpeen mukaan tehdä uuden haun Spotifyn ja Discogsin rajapintoihin puuttuvien tietojen löytämiseksi. On myös olemassa Shazamin kaltaisia, nettitstreamia kuuntelevia palveluja, joista listan soittotapah-

tumista voisi koostaa automaattisesti. Automatiikka näissä ei tosin ole täysin vedenpitävää, ja esimerkiksi mainosten taustamusiikkia saattaisi päätyä raportoitavaksi.

Radio Helsingin nettisivujen ohjelmakalenteria hyödyntämällä vaadittaville raporteille voisi tehdä valmiit pohjat. Ohjelmakalenteria voisi myös käyttää tutkimaan sitä, minkä ohjelmien raportti puuttuu. Järjestelmän voisi koodata lähettämään puuttuvien raporttien ohjelmantekijöille muistutusviestejä.

Jotkut kuuijat ovat toivoneet Radio Helsingin nettisivuille ominaisuutta, jossa kuulijat voivat hakea tilastoja soitetuista kappaleista tai artisteista, tai hakea suosikkiohjelmansa soitetuimpia kappaleita tietyillä aikaväleillä. Myös dj:t ovat toivoneet ominaisuutta nähdä omat top-kappaleensa raportointisovelluksen käyttöliittymässä.

Tuotantoon mennyt sovellus otettiin käyttäjiensä parissa vastaan melko positiivisesti. Ohjelmantekijät, jotka hakevat suuren osan soittamistaan kappaleista DJOnlinen rajapinnasta arvostivat erityisesti vähentynyttä tarvetta copy-pastelle. Muualta kuin playout-ohjelmistosta musiikkinsa soittavat dj:t joutuvat käyttämään enemmän aikaa raportointiin, sillä toisin kuin vanhassa sovelluksessa, uudessa sovelluksessa kappaletta ei voinut tallentaa järjestelmään ilman, että kaikki vaadittavat tiedot on syötetty.

Tuotantoversion julkaisun jälkeen sovelluksesta ilmeni joitakin bugeja ja ulkoasumuutostoiveita. Esimerkiksi kappaleiden järjestyksen muuttaminen raahaamalla ei toiminut, ja kappalehakunäkymää muutettiin hieman. Näiden heti julkaisun jälkeen toteutettujen korjausten jälkeen sovellus on toiminut ilman suurempia muutoksia ainakin tekstin kirjoittamishetkeen 24.4.2021 asti.

8 Lähteet

JavaScript Perusteet Jyväskylän yliopisto. Luettavissa: http://appro.mit.jyu.fi/tiea2120/luennot/javascript_basics/. Luettu 20.5.2020

Kemiantekniikan ohjelmointikurssi, Aalto-yliopisto. Luettavissa: https://mycourses.aalto.fi/mod/book/view.php?id=238495&chapterid=571&lang=fi. Luettu 20.5.2020

Visual Studio Code, Visual Studio Code docs. Luettavissa: https://code.visualstudio.com/docs. Luettu 21.5.2020

Eslint dokumentaatio, Eslint. Luettavissa: https://eslint.org/docs/user-guide/getting-started. Luettu 21.5.2020

Prettier Github-repositorio, Prettier. Luettavissa: https://github.com/prettier/prettier-vscode. Luettu 21.5.2020

What exactly is Github anyway, TechCrunch. Luettavissa: https://techcrunch.com/2012/07/14/what-exactly-is-github-anyway/. Luettu 21.5.2020.

Heroku, Heroku. Luettavissa: https://www.heroku.com/. Luettu 21.5.2020

What is NPM, W3Schools. Luettavissa: https://www.w3schools.com/whatis/whatis_npm.asp. Luettu 8.6.2020

Cloud SQL for MySQL documentation, Google. Luettavissa https://cloud.google.com/sql/docs/mysql. Luettu 21.5.2020

Mikä ihmeen frontend, Zaibatsu Interactive. Luettavissa https://zaibatsu.fi/mika-ihmeen-frontend-frontti-selvaksi-alle-200-sanan/. Luettu 1.6.2020

Web-ohjelmointi, Tampereen yliopisto. Luettavissa http://www.cs.tut.fi/~seitti/2015/kalvot/react/all.html. Luettu 2.6.2020

Redux dokumentaatio, Redux. Luettavissa https://redux.js.org/introduction/getting-started. Luettu 8.6.2020

An Introduction to Redux's Core Concepts, DigitalOcean. Luettavissa https://www.digitalocean.com/community/tutorials/redux-redux-intro. Luettu 8.6.2020

Why use React-Redux, React-Redux. Luettavissa https://react-redux.js.org/introduction/why-use-react-redux. Luettu 8.6.2020

Getting started with React Redux part 1, DOTNet Basic. Luettavissa http://blogs.kansiris.org/2018/12/getting-started-with-react-redux-part1.html. Luettu 8.6.2020

Introducing Semantic UI component library, Sitepoint. Luettavissa https://www.sitepoint.com/introducing-semantic-ui-component-library/. Luettu 8.6.2020

Semantic UI React, Semantic UI React. Luettavissa https://react.semantic-ui.com/. Luettu 8.6.2020

Button, Semantic UI. Luettavissa https://react.semantic-ui.com/elements/button/#types-button. Luettu 8.6.2020

Mitä markkinoijan tulee ymmärtää web-ohjelmoinnista, Dagmar. Luettavissa https://www.dagmar.fi/verkkopalvelukehitys/mita-markkinoijan-tulee-ymmartaa-web-ohjelmoinnista/. Luettu 8.6.2020

Node.js ja Express, Full Stack Open 2020. Luettavissa https://fullstackopen.com/osa3/node_js_ja_express. Luettu 9.6.2020

Node.js – Express framework, TutorialsPoint. Luettavissa https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm. Luettu 10.6.2020

Rest on nettipalveluiden yhteinen kieli, Tivi. Luettavissa https://www.tivi.fi/uutiset/rest-on-nettipalveluiden-yhteinen-kieli/23703ab5-dd19-383e-a422-ebfc3d910583. Luettu 10.6.2020

REST-rajapintojen testaus Postmanilla, Nemit Blog. Luettavissa https://blog.nemit.fi/rest-rajapintojen-testaus-postmanilla-44c02cbb4661. Luettu 6.7.2020

A successful Git branching model, Nvie Blog. Luettavissa https://nvie.com/posts/a-successful-git-branching-model/. Luettu 6.7.2020

MySQL ja PHP: Osa 1 - Johdanto, Ohjelmointiputka. Luettavissa: https://www.ohjelmointiputka.net/oppaat/opas.php?tunnus=mysqlphp01. Luettu 6.7.2020

Sequelize, Sequelize. Luettavissa https://sequelize.org/. Luettu 6.7.2020.

Tietokannan taulut, MySQL Workbench. Screenshot MySQL Workbench –ohjelmasta

Teosto/Gramex raportointiohje. Luettavissa: https://www.gramex.fi/wp-content/uploads/2020/02/Raportointiohje-Kaupalliset-radiot-1.1.2020.pdf. Luettu 10.12.2020

Why is Node.js popular, Section.io. Luettavissa https://www.section.io/engineering-education/why-node-js-is-popular/. Luettu 21.4.2021

What Languages Are Used For Backend Development, Sagara Technologies. Luettavissa https://sagaratechnology.medium.com/what-languages-are-used-for-back-end-development-71a8a10c135c. Luettu 21.4.2021

What not to commit, dart.dev. Luettavissa: https://dart.dev/guides/libraries/private-files. Luettu 10.12.2020

React vs Angular vs Vue, What to Choose in 2020, Techmagic. Luettavissa https://medium.com/techmagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d. Luettu 21.4.2021

Alternatives to MySQL, Slant. Luettavissa https://www.slant.co/options/4216/alternatives/~mysql-alternatives. Luettu 21.4.2021

Choi, David: Full Stack React, TypeScript & Node. Julkaissut Packt Publishing Ltd. Julkaistu 2020.

Ohjelmistotestauksen perusteita, Antti-Pekka Tuovinen, Helsingin Yliopisto. Luettu 22.4.2021. Luettavissa https://www.cs.helsinki.fi/u/aptuovin/testaus/Ohj_testaus_2013_1.pdf

Full Stack Open – Node-sovellusten testaaminen, Helsingin yliopisto. Luettu 23.4.2021. Luettavissa

https://fullstackopen.com/osa4/sovelluksen_rakenne_ja_testauksen_alkeet#node-sovellusten-testaaminen

Full Stack Open – End to end –testaus, Helsingin yliopisto. Luettu 24.4.2021. Luettavissa https://fullstackopen.com/osa5/end_to_end_testaus

Paavilainen Petri: Psykologian tutkimustyöopas. Julkaissut Edita Prima Oy. Julkaistu 2012

Jyväskylän Yliopisto. Jyväskylän Yliopiston Koppa, Laadullinen tutkumus. Luettavissa https://koppa.jyu.fi/avoimet/hum/menetelmapolkuja/menetelmapolku/tutkimusstrategiat/laadullinen-tutkimus. Luettu 24.4.2021

Jyväskylän Yliopisto. Jyväskylän Yliopiston Koppa, Tapaustutkimus. Luettavissa https://koppa.jyu.fi/avoimet/hum/menetelmapolkuja/menetelmapolku/tutkimusstrategiat/tapaustutkimus. Luettu 24.2.2021

Lowdermilk Travis: User-Centered Design. Julkaissut O'Reilly Media, Inc. Julkaistu 2014

Femto15, User Experience and User Centered Design (UX and UCD). Luettavissa: https://www.femto15.com/user-experience-and-user-centered-design-ux-and-ucd. Luettu 4.5.2021.