Tommi Vainio

Tilausvideopalvelu farmasia-alan yritykselle

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

18.4.2023

Tiivistelmä

Tekijä:	Tommi Vainio
Otsikko:	Tilausvideopalvelu farmasia-alan yritykselle
Sivumäärä:	47 sivua
Aika:	18.4.2023

Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Mediatekniikka
Ohjaaja:	Lehtori Toni Spännäri

Insinöörityön tarkoituksena oli luoda video on demand -striimauspalvelu eli tilausvideopalvelu lääkkeiden koneelliseen annosjakeluun keskittyvälle yritykselle. Tavoitteena oli, että palvelun kautta yritys voisi myydä asiakkailleen katseluoikeuksia sen tuottamiin koulutustallenteisiin ja webinaareihin. Palvelu tuli liittää osaksi yrityksen kehitettävänä olevaa ekstranetportaalia. Kehitettävän video on demand -palvelun tuli jakautua ekstranetissä verkkokauppaosioon, materiaalipankkiosioon ja hallintaosioon. Koulutusten katseluoikeuksien myynnin tuli tapahtua verkkokauppaosiossa, katselun materiaalipankkiosiossa ja niiden hallinnan yrityksen sisällönhallitsijoiden toimesta hallintaosiossa.

Palvelun selainpuoli toteutettiin React-JavaScript-kirjaston ja yrityksen oman komponenttikirjaston avulla ja palvelinpuoli taas Node.js-suoritusympäristön ja NestJS-sovelluskehyksen avulla. Molemmissa ohjelmointikielenä käytettiin TypeScriptiä. Videoiden säilyttämiseen ja striimaamiseen käytettiin Vimeo-videopalvelua ja sen Advanced-paketin tuomia ominaisuuksia. Palvelun ja Vimeon välinen kommunikointi toteutettiin Vimeon avoimen rajapinnan kautta. Videot ladataan Vimeoon tus-protokollan avulla, ja koulutuksiin liitetyt kuvat tallennetaan palvelun palvelinpuolen kautta Azuren Blob Storage -säilöön. Palvelun selainpuolen näkymät toteutettiin yrityksen käytössä olleen ulkoisen käyttöliittymäsuunnittelijan Figma-palveluun luomien mallien mukaisiksi.

Työn lopputuloksena saatiin täysin responsiivinen tilausvideopalvelu, jonka kautta yrityksen asiakkaat voivat selailla koulutuksia, lisätä niille tykkäyksiä, katsella koulutuksiin mahdollisesti liitettyjä teaser-videoita, ostaa kuukauden kestäviä tilauksia koulutuksiin ja katsella ostettuja ja ilmaisia koulutuksia. Yrityksen sisällönhallitsijat voivat taas luoda, muokata ja poistaa koulutuksia. Suurimmiksi jatkokehityskohteiksi jäivät maksutoiminnallisuuden toteuttaminen sekä katseluoikeuksien myyminen tulossa oleviin webinaareihin ja niiden näyttäminen palvelussa.

Avainsanat: tilausvideo, TypeScript, React, Node.js, NestJS

Abstract

Author:	Tommi Vainio
Title:	Video on demand streaming service for a pharmaceutical company
Number of Pages:	47 pages
Date:	18 April 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Media Technology
Supervisor:	Toni Spännäri, Senior Lecturer

The purpose of this final year project was to create a video on demand streaming service for a company which produces an automated dose dispensing of medicines. The aim was that using this service the company could sell viewing rights to the training records and webinars it produces. The streaming service had to be integrated into the company's extranet portal under development. In the extranet this streaming service had to be divided into three sections: online store, material bank and management section.

The frontend of the service was done using React JavaScript library and the company's own style component library. The backend was done using Node.js runtime environment and NestJS framework. TypeScript was used as programming language for both the frontend and the backend. Vimeo video hosting platform is used for hosting and streaming the videos.

The result of the project was a fully responsive video on demand service through which the customers of the company can browse trainings, like them, watch teasers possibly added to them, buy month-long subscriptions for trainings and watch both bought and free trainings. The company's content managers can create, modify and delete trainings. The biggest further development targets are payment functionality, selling viewing rights for upcoming webinars and displaying them in the service.

Keywords: video on demand, TypeScript, React, Node.js, NestJS

Sisällys

Lyhenteet

AVOD	Advertising-based Video on Demand. Mainostuloihin perustuva video on demand -malli, jossa videosisältö on ilmaiseksi katsottavissa.
CDN	Content Delivery Network. Sisällönjakeluverkko, joka koostuu maantieteellisesti hajautetuista konesaleista ja välityspalvelimista.
CSS	Cascading Style Sheets. Tyylikieli, jonka avulla määritellään verkkosivun ulkoasu.
CSSOM	CSS Object Model. Malli, joka määrittää DOM-puun elementeille tyylit.
DOM	Document Object Model. Puuta muistuttava malli, joka muodostaa HTML-dokumentin.
HTML	Hypertext Markup Language. Merkintäkieli, jolla luodaan verkkosivun rakenne.
JSX	JavaScript XML. React-kirjaston komponenttien palauttama HTML:n syntaksia muistuttava kieli, jos ohjelmointikielenä on käytetty JavaScriptiä.
NPM	Node Package Manager. Node.js-suoritusympäristön oma JavaScript-pakettikirjasto.
SVOD	Subscription Video on Demand. Tilauspohjainen video on demand -malli, jossa käyttäjä ostaa katseluoikeuden video on demand -palvelun sisältöön tietyksi ajaksi.

TLS TypeScript Language Service. TypeScript-ohjelmointikieleen kuuluva sovelluskerros, joka auttaa koodieditoreja muun muassa koodin muotoilussa, värittämisessä ja automaattisessa täydennyksessä.

TSX TypeScript XML. React-kirjaston komponenttien palauttama HTML:n syntaksia muistuttava kieli, jos ohjelmointikielenä on käytetty TypeScriptiä.

TVOD Transactional Video on Demand. Video on demand -malli, jossa käyttäjä ostaa katseluoikeuden yhteen videoon kerrallaan.

VDOM Virtual Document Object Model. Malli, jota React-kirjasto käyttää DOM-puun elementtien päivittämiseen.

VOD Video on Demand. Sisällönjakeluteknologia, jossa video striimataan internetyhteyden välityksellä käyttäjän laitteelle.

1 Johdanto

Insinöörityön tavoitteena oli luoda video on demand -striimauspalvelu eli tilausvideopalvelu lääkkeiden koneellista annosjakelua tuottavalle Pharmac Finland Oy:lle. Tämä palvelu tuli liittää osaksi yrityksen kehitteillä olevaa asiakkaille tarkoitettua portaalia. Koska lääkkeiden koneellisen annosjakelun kannattavuus laskee, yrityksen tavoitteena on laajentua uusille liiketoiminta-alueille, ja yksi tällaisista alueista on koulutustallenteiden ja webinaarien katseluoikeuksien myyminen. Tätä varten yritys tarvitsi alustan, jonka kautta se voi myydä ja näyttää tuottamiaan koulutustallenteita ja webinaareja.

Palvelun toiminnallisuudet tuli integroida osaksi yrityksen laajaa ekstranet-web-sovellusta hyödyntäen tuon sovelluksen kehityksessä käytettyjä teknologioita. Näihin teknologioihin kuuluivat muun muassa TypeScript-ohjemointikieli, React-kirjasto, selainpuolen rakennustyökalu Vite, Node.js-ajoympäristö, NestJS-sovelluskehys ja Microsoft SQL Server Management Studio -tietokannanhallintatyökalu. Videoiden säilytys- ja striimausalustaksi oli jo aiemmin valittu Vimeo, jonka avointa rajapintaa tuli käyttää videoiden hallitsemiseen.

Insinöörityön toisessa luvussa esitellään video on demand käsitteenä ja luodaan lyhyt katsaus sen merkitykseen, historiaan, siihen liittyvään teknologiaan ja Vimeoon. Kolmannessa luvussa käsitellään modernia web-kehitystä yleisesti ja esitellään lyhyesti tärkeimmät insinöörityön teknisessä toteutuksessa käytetyt teknologiat. Insinöörityön neljännessä luvussa käydään läpi työn tavoitteet tarkemmin ja esitellään, millaisista lähtökohdista työtä lähdettiin toteuttamaan. Viides luku käsittelee työn varsinaista toteutusta, sen tuloksia ja jatkokehityskohteita. Kuudennessa luvussa arvioidaan työn onnistumista ja alkuperäisten tavoitteiden toteutumista.

2 Video on demand -sisällönjakeluteknologia

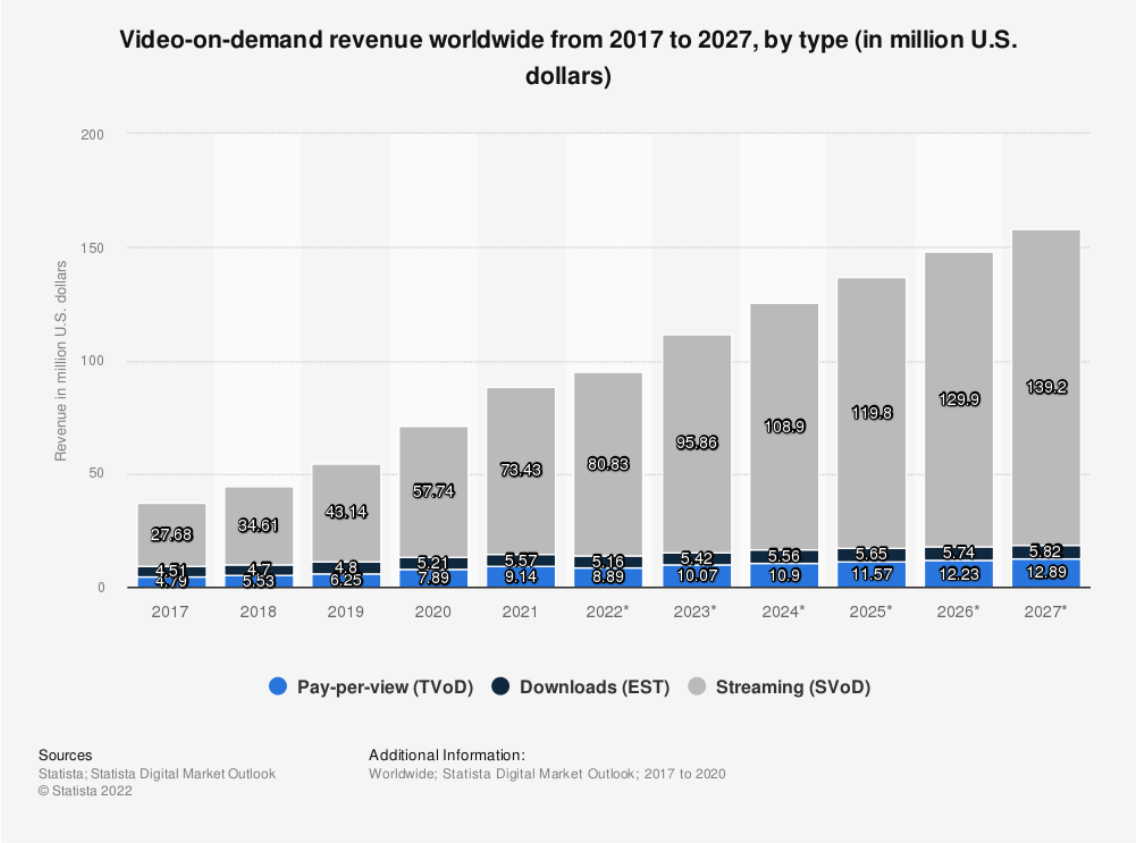
Video on demand eli VOD on sisällönjakeluteknologia, jonka avulla videosisältöön pääsee käsiksi kaikkialla, kunhan käytössä on videoiden toistamiseen soveltuva laite ja internetyhteys, jossa on riittävästi kaistanleveyttä. Perinteisen television näyttämästä sisällöstä VOD-sisältö eroaa siinä, että sitä voi katsoa usealla eri laitteella, esimerkiksi älypuhelimella, tabletilla, älytelevisiolla ja tietokoneella, eikä se vaadi antenni- tai kaapeliverkkoa. Lisäksi striimattavat videot ovat aina saatavilla ja niitä pystyy kelaamaan edestakaisin, keskeyttämään ja jopa lataamaan, kun taas perinteisen television sisältö on saatavilla ainoastaan silloin, kun sitä lähetetään televisiovastaanottimiin. (1; 2.)

2.1 Video on demand -mallit

Video on demand -palvelut voidaan karkeasti jakaa kolmeen malliin, jotka ovat SVOD (Subscription Video on Demand), TVOD (Transactional Video on Demand) ja AVOD (Advertising-based Video on Demand). Näistä malleista yleisin on SVOD. Siinä käyttäjä ostaa tilauksen, joka antaa katseluoikeuden kyseisen palvelun sisältöön tietyksi ajaksi. Tällaiset tilaukset voivat olla esimerkiksi kuukausittain, vuosineljänneksittäin tai vuosittain uusittavia. SVOD-palvelut tarjoavat myös usein erilaisia hintapaketteja, esimerkiksi opiskelijoille ja perheille. SVOD-mallia käyttäviä palveluita ovat esimerkiksi Netflix, Amazon Prime Video ja HBO Max. TVOD-mallissa käyttäjä maksaa vain siitä sisällöstä, jonka hän katsoo. Esimerkiksi, jos käyttäjä haluaa nähdä tällaisesta palvelusta tietyn elokuvan, hän ostaa katseluoikeuden vain kyseiselle elokuvalle eikä koko palvelun sisällölle niin kuin SVOD-mallissa. TVOD-mallia käyttäviä palveluita ovat esimerkiksi iTunes ja Sky Box Office. AVOD-palveluissa suurin osa videoista, jos ei kaikki, on ilmaiseksi katsottavissa. Palvelun ylläpito rahoitetaan mainoksilla, joita saattaa olla upotettuna itse palvelun käyttöliittymään sekä videomuotoisena striimattavan videosisällön yhteydessä. Monet tällaiset palvelut tarjoavat kuitenkin myös maksullisia tilauksia, joilla palveluun aukeaa uusia ominaisuuksia, kuten videoiden katselu ilman mainoksia. Ylivoimaisesti

suosituin AVOD-mallia käyttävä palvelu on Youtube. Muita esimerkkejä ovat DailyMotion ja IBM Video Streaming. (1.)

Video on demand -palveluiden suosio on noussut maailmanlaajuisesti jyrkästi viime vuosina. Tästä kertoo myös kuvan 1 diagrammi, joka kuvaa VOD-palveluiden tuottamia tuloja vuosina 2017−2027. Kun vuonna 2017 VOD-palvelut tuottivat maailmanlaajuisesti noin 35 miljardia dollaria, oli sama arvo vuonna 2021 jo lähes 90 miljardia dollaria. Diagrammi ennustaa VOD-palveluiden tuottamien tulojen nousevan vuonna 2027 jo yli 150 miljardiin dollariin. Huomionarvoista on se, että näihin lukuihin eivät sisälly AVOD-palvelut ja niiden tuottamat mainostulot.



Kuva 1. Video on demand -palveluiden tuottamat tulot maailmanlaajuisesti tyypeittäin Yhdysvaltain dollareina. Näihin tuloihin ei ole laskettu mukaan AVOD-palveluiden tuottamia tuloja, vaan niiden tilalla on ladattavista videoista saatavat tulot. (3.)

VOD-palveluiden valtavasta suosiosta kertoo myös se, että striimattu video-sisältö ohitti historiallisesti kaapelitelevision kautta näytetyn sisällön katsotuimpana sisältönä Yhdysvalloissa heinäkuussa 2022. Tähän vaikutti tosin se, että heinäkuu on televisiosisällön osalta hiljainen kuukausi, sillä suuri osa televisio- ja urheilusarjoista on tauolla tuolloin, kun taas moniin suoratoistopalveluihin tuli tuolloin uutta sisältöä. Striimauspalveluita ei kuitenkaan ollut koskaan aiemmin katsottu Yhdysvalloissa kuukaudessa enemmän kuin kaapelitelevision sisältöä, joten kyseessä oli merkittävä muutos. (4.)

2.2 Historia

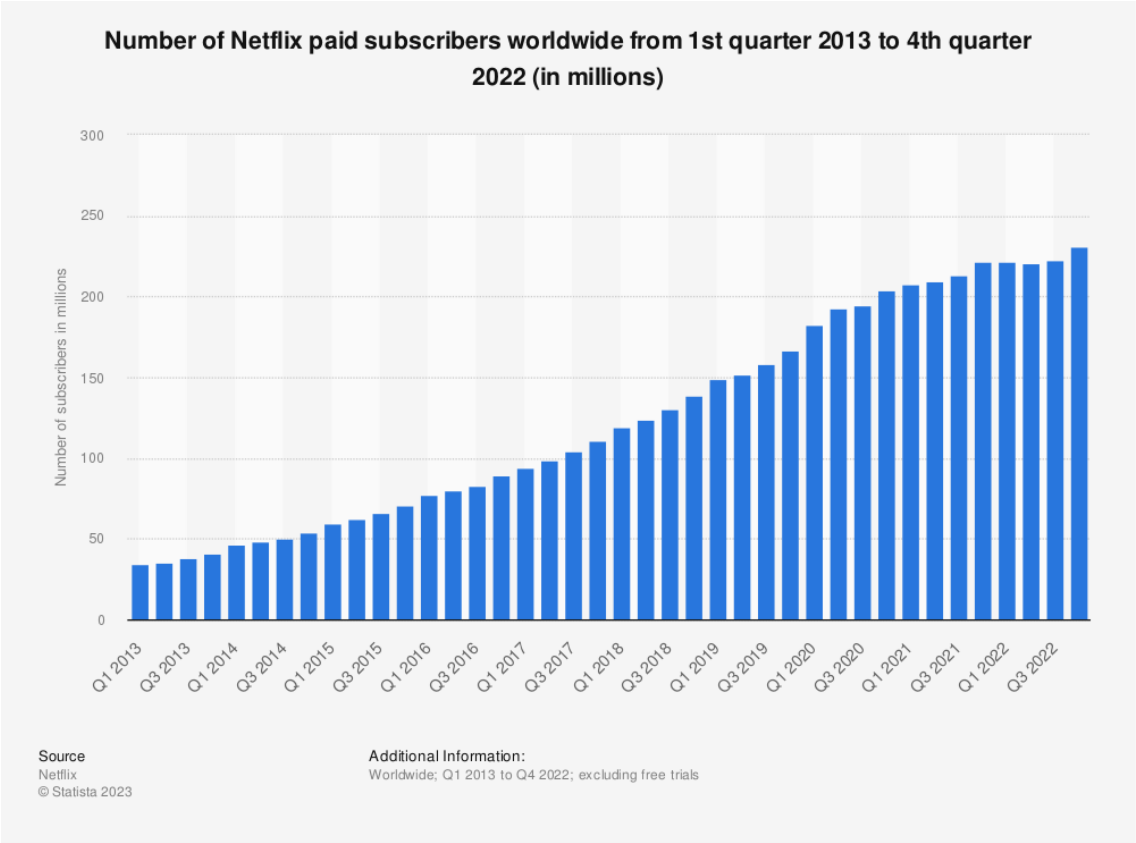
Ensimmäiset video on demand -palvelut luotiin jo 1990-luvulla, mutta niiden suosio jäi hyvin vähäiseksi. 2000-luvun puolivälissä parannukset internetyhteyksien nopeuksiin ja kaistojen leveyteen olivat osaltaan vaikuttamassa ensimmäisten valtavirtaan nousseiden VOD-palveluiden syntyyn ja suosioon. Youtube perustettiin vuonna 2005, ja vain vuosi sen perustamisen jälkeen Google osti sen 1,65 miljardilla dollarilla. Tämä oli selvä merkki muille alan toimijoille, että VOD-palveluiden markkinat tulisivat kasvamaan räjähdysmäisesti. Amazon julkaisi oman striimauspalvelunsa Prime Videon vuonna 2006 ja BBC iPlayer-palvelunsa vuonna 2007. (5.)

Netflix perustettiin jo vuonna 1997, mutta se aloitti DVD-videoita postin välityksellä vuokraavana yrityksenä. Netflix oli alusta alkaen tilauspohjainen palvelu, jossa tilauksen ostettuaan sai tilauksen kestoajaksi katseluoikeuden käytännössä kaikkiin Netflixin omistamiin videoihin. Aluksi videoita ei kuitenkaan striimattu, vaan ne lähetettiin DVD-muodossa tilaajien koteihin. Kun tilaaja palautti tilaamansa videon, hän sai tilalle uuden. 10 vuotta perustamisensa jälkeen vuonna 2007 Netflix alkoi myös striimata videoita. Striimauspalvelu tuli lisänä DVD-vuokrauspalvelun oheen ja sisältyi tilausmaksuun. Vuonna 2010 Netflix alkoi keskittymään pääasiallisesti videoiden striimaukseen, mutta piti DVD-palvelunsa myös osana toimintaansa. Se vuokraa DVD-videoita vielä tänäkin päivänä. (6.)

Merkittävä käännekohta Netflixin ja ylipäänsä VOD-palveluiden historiassa tapahtui vuonna 2012, kun Netflix julkaisi ensimmäisen alkuperäissarjansa Lilyhammerin. Sitä ennen striimauspalvelut olivat näyttäneet vain vanhoja TV-sarjoja ja elokuvia, jotka oli jo julkaistu televisiossa tai elokuvateattereissa. Alkuperäissarjansa myötä Netflix muutti koko televisio- ja elokuva-alan dynamiikkaa näyttämällä, että myös VOD-palvelut voivat tuottaa sisältöä, jota miljoonat ihmiset haluavat katsoa ja maksaa siitä. Tämän jälkeen Netflix on tuottanut yli 1 900 alkuperäissarjaa ja -elokuvaa. Näistä esimerkiksi Orange is the New Black, Stranger Things ja Narcos kuuluivat 2010-luvun suosituimpien sarjojen joukkoon. Myös muut striimauspalvelut ovat lähteneet tämän jälkeen tuottamaan omaa alkuperäissisältöään. (5.)

Vastatakseen kilpailuun televisio- ja elokuvayhtiöt ovat perustaneet omia VOD-palveluitaan. Esimerkiksi A&T, NBC, Disney ja Isossa-Britanniassa Channel 4 tarjoavat sisältöään myös omien suoratoistopalveluidensa kautta. Suomalaisia esimerkkejä televisioyhtiöiden suoratoistopalveluista ovat Nelonen Median Ruutu ja MTV:n Katsomo. Televisio- ja elokuvayhtiöiden perustamat VOD-palvelut ovat johtaneet siihen, että perinteisten VOD-palveluiden on pitänyt panostaa entistä enemmän alkuperäissarjoihinsa ja -elokuviinsa. Tämä johtuu siitä, että televisio- ja elokuvayhtiöiden omaa sisältöä on alettu näyttää näiden omissa palveluissa yksinoikeudella. (5.)

Tällä hetkellä maailman suosituin VOD-palvelu on Youtube, jolla on yli kaksi miljardia käyttäjää. Netflix taas pitää hallussaan maailman suosituimman SVOD-palvelun paikkaa. (5.) Viimeisimpien tietojen mukaan sillä on yli 220 miljoonaa tilaajaa (7). Kuvasta 2 voidaan havaita, että Netflixin tilaajamäärän kasvu on ollut melko tasaista vuosien varrella lukuun ottamatta vuoden 2020 ensimmäisellä vuosineljänneksellä tapahtunutta melko jyrkkää nousua. Tämä tilaajamäärän äkillinen jyrkkä kasvu johtui tuolloin alkaneesta maailmanlaajuisesta koronaviruspandemiasta, jonka vuoksi ihmisten täytyi viettää aiempaa enemmän aikaa kotonaan. Myös muiden merkittävimpien SVOD-palveluiden tilaajamäärät kokivat tuolloin vastaavanlaisen pyrähdyksen. (7.)



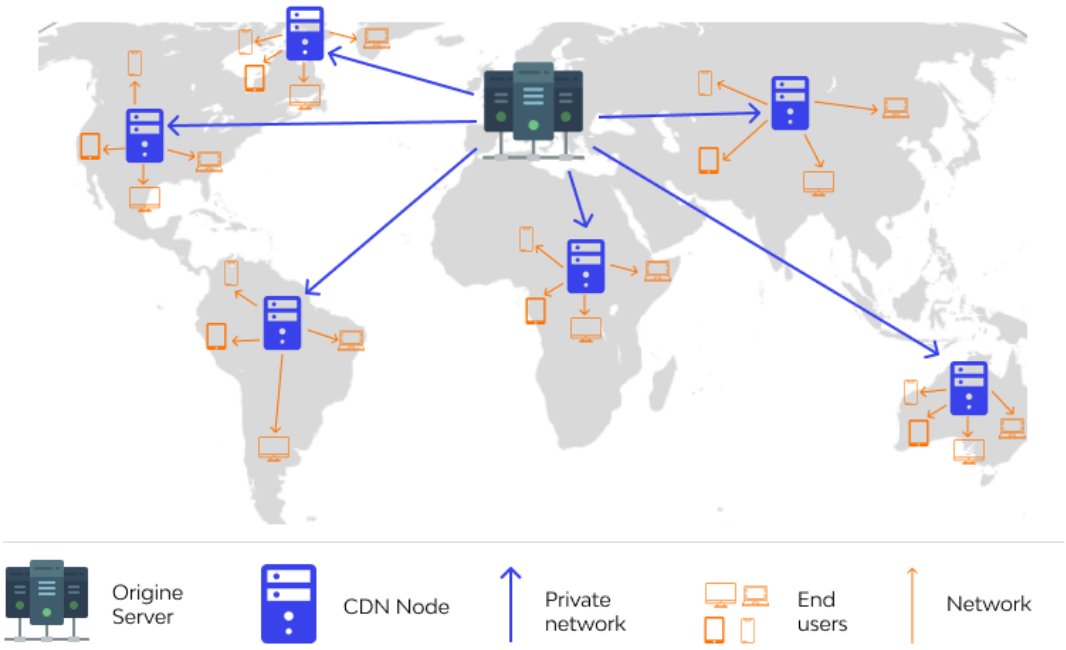
Kuva 2. Netflixin maailmanlaajuinen tilaajamäärä vuoden 2013 ensimmäiseltä vuosineljännekseltä vuoden 2022 neljännelle vuosineljännekselle (miljoonaa tilaajaa) (8).

2.3 Striimausteknologia

Videon striimaaminen voidaan jakaa neljään prosessiin: prosessointiin (processing), säilytykseen (storage), toimitukseen (delivery) ja toistamiseen (playback). Ensin video pitää prosessoida, sillä raaka videotiedosto itsessään sisältää liikaa dataa eikä sitä siksi voida sellaisenaan striimata. Vain viisi sekuntia raakaa videota vie noin 1,7 gigatavua tallennustilaa, joten sen striimaaminen veisi valtavasti kaistaa. Pakkausprosessia kutsutaan koodaukseksi (encoding). Koodekki taas on ohjelma tai algoritmi, joka koodaa videon pakatuiksi videotiedostoiksi videon siirtoa varten ja käyttäjän päässä dekoodaa tiedostot takaisin alkuperäiseksi videoksi. Koodekki voi pienentää videotiedoston kokoa jopa 850 kertaa alkuperäistä pienemmäksi, joten se on ratkaisevan tärkeä osa videon striimausta. (9.)

Koodauksen jälkeen seuraava vaihe videon prosessoinnissa on transkoodaus. Siinä digitaalisen videon kolme elementtiä, tiedostoformaatti, video ja ääni, dekoodataan pakkaamattomaan tilaan ja koodataan uudelleen eri formaatteihin. Tämä tehdään siksi, että kaikki eivät muuten pystyisi välttämättä katsomaan videota, sillä alkuperäisellä formaatilla olevan videon striimaus saattaisi esimerkiksi vaatia enemmän kaistaa kuin kaikilla käyttäjillä olisi käytössään. Lisäksi kaikki laitteet eivät tue kaikkia videoformaatteja. Näin ollen, jotta video tavoittaisi mahdollisimman suuren yleisön, se on transkoodattava useisiin eri formaatteihin, resoluutioihin ja bittinopeuksiin käyttämällä eri koodekkeja. Transkoodauksen jälkeen sen tuloksena syntyneet videotiedostot kääritään manifestitiedostoon, josta löytyy myös videon metadata ja tiedot tuetuista bittinopeuksista sekä ohjeet videosoittimille videotiedostojen lataamiseen. (9; 10.)

Seuraavaksi video täytyy tallentaa niin, että se on saatavilla koko ajan niille, joilla on oikeus nähdä se. Videon tallentamiseen ja striimaamiseen loppukäyttäjille käytetään sisällönjakeluverkkoa (CDN). Sisällönjakeluverkolla tarkoitetaan maantieteellisesti hajautettua välityspalvelimien ja konesalien verkkoa. Kun video tallennetaan sisällönjakeluverkossa olevalle palvelimelle, sitä kopioidaan myös muille verkon palvelimille. Näin eri puolilla maailmaa olevat käyttäjät voivat striimata videota heitä lähimmältä palvelimelta, jolloin mahdollisuus videon pätkimiseen ja heikkoon laatuun pienenee. (9.) Kuvassa 3 on esimerkki yksinkertaisesta sisällönhallintaverkosta, jonka pääpalvelin sijaitsee Euroopassa ja jokaisessa muussa kuvan kartalla näkyvässä maanosassa on yksi kyseisen verkon välityspalvelin. Sisällönjakeluverkon palvelimet tallentavat dataa tietyksi ajaksi, ja jos videota ei löydy käyttäjää lähimpänä sijaitsevalta palvelimelta, tämä palvelin pyytää videon joltain muulta verkon palvelimelta, josta kyseinen video löytyy (10). Tämän jälkeen käyttäjän lähin palvelin tallentaa videon itseensä, jolloin seuraavalla katselukerralla videon lataaminen on nopeampaa (9).



Kuva 3. Esimerkki maailmanlaajuisesta sisällönhallintaverkosta (11).

Jotta käyttäjä voi toistaa videota, striimauspalvelussa tulee olla videosoitin. Videosoitin dekoodaa pakatun videon pakkaamattomaan muotoon ja näyttää sen ruudulla. Lisäksi videosoitin tarjoaa käyttäjälle rajapinnan, jonka avulla hän pystyy hallitsemaan videon toistoa. Tämän rajapinnan avulla käyttäjä pystyy muun muassa selaamaan videota edestakaisin, muuttamaan sen nopeutta ja valitsemaan näytetäänkö mahdolliset tekstitykset. Tarkemmin tarkasteltuna videosoitin lataa ennen striimauksen aloittamista manifestitiedoston ja pyrkii tunnistamaan, millaisella laitteella sitä käytetään ja kuinka paljon kaistanleveyttä on käytössä. Näiden tietojen perusteella videosoitin valitsee manifestitiedostosta olosuhteisiin parhaiten sopivan version videosta striimattavaksi. Esimerkiksi käyttäjille, joilla on hidas verkkoyhteys, videosoitin valitsee heikompilaatuisen version videosta, sillä sen striimaaminen vie vähemmän kaistaa. Näin videon toistaminen on sulavampaa ja luotettavampaa. Tätä kutsutaan Adaptive Bitrate (ABR) -algoritmiksi. Soittimet antavat myös käyttäjien itse valita videon laadun. Striimattavaa videota toistavat soittimet lataavat tyypillisesti kolme videon osasta eli segmenttiä ennen, kuin alkavat toistaa videota. Näin soitin pyrkii varmistamaan, että video ei pysähdy kesken

kaiken lataamaan seuraavaa segmenttiä, vaan se olisi jo valmiiksi ladattu, kun soittimen on tarkoitus näyttää se. (9; 10.)

2.4 Vimeo-sisällönjakelualusta

Vimeo on vuonna 2004 perustettu videosisällönjakelualusta. Nykyään sillä on yli 260 miljoonaa käyttäjää. Youtuben tavoin Vimeota voisi luonnehtia käyttäjäkeskeiseksi VOD-palveluksi, eli sen tarkoituksena on toimia alustana, jonka kautta sen käyttäjät voivat jakaa omia videoitaan ja pitää livelähetyksiä. Vimeo ei kuitenkaan ole AVOD-palvelu niin kuin Youtube, sillä se ei näytä mainoksia sivuillaan. Sen sijaan Vimeon toiminta perustuu tilauspohjaisten pakettien myymiseen sisällöntuottajille. Näitä paketteja on eri tasoisia, ja mitä kalliimman paketin käyttäjä ostaa, sitä enemmän ominaisuuksia hän saa käyttöönsä ja sitä enemmän videoita hän voi ladata Vimeoon. (12; 13.)

Tällä hetkellä Vimeo tarjoaa neljä erilaista maksullista tilausvaihtoehtoa, jotka ovat kahdeksan euroa kuukaudessa maksava Starter, 20 euroa kuukaudessa maksava Standard, 49 euroa kuukaudessa maksava Advanced ja Enterprise-tilaus, jolle käyttäjän täytyy erikseen sopia hinta Vimeon kanssa. Vimeota voi käyttää myös ilmaiseksi, mutta tällöin käyttäjä voi ladata Vimeoon vain kaksi videota kuukaudessa ja yhteensä enintään 25 videota. Vertailun vuoksi, halvimman Starter-tilauksen valinnut käyttäjä voi ladata Vimeoon enintään 60 videota vuodessa ilman kuukausittaista rajoitusta. Vimeota ilmaiseksi käyttävät eivät voi myöskään muun muassa muokata videosoitinta tai rajoittaa videon näkyvyyttä. Livelähetyksien pitäminen onnistuu vain Advanced- ja Enterprise-tilauksilla, ja näistä vain Enterprise-tilauksen valinneet saavat käyttöönsä Vimeon Live Api -rajapinnan, jonka avulla livelähetyksiä voi hallita muualtakin kuin Vimeon omien sivujen kautta. (14.)

Vimeo säätelee Youtubea tarkemmin, minkälaisia videoita sinne ladataan. Kun Youtubeen voi ladata lähes mitä tahansa, Vimeo pyrkii pitämään huolen siitä, että sen sivuilla näytettävät videot olisivat mahdollisimman laadukkaita. (12.) Vimeo sopii Youtubea paremmin ammattimaiseen video on demand -käyttöön

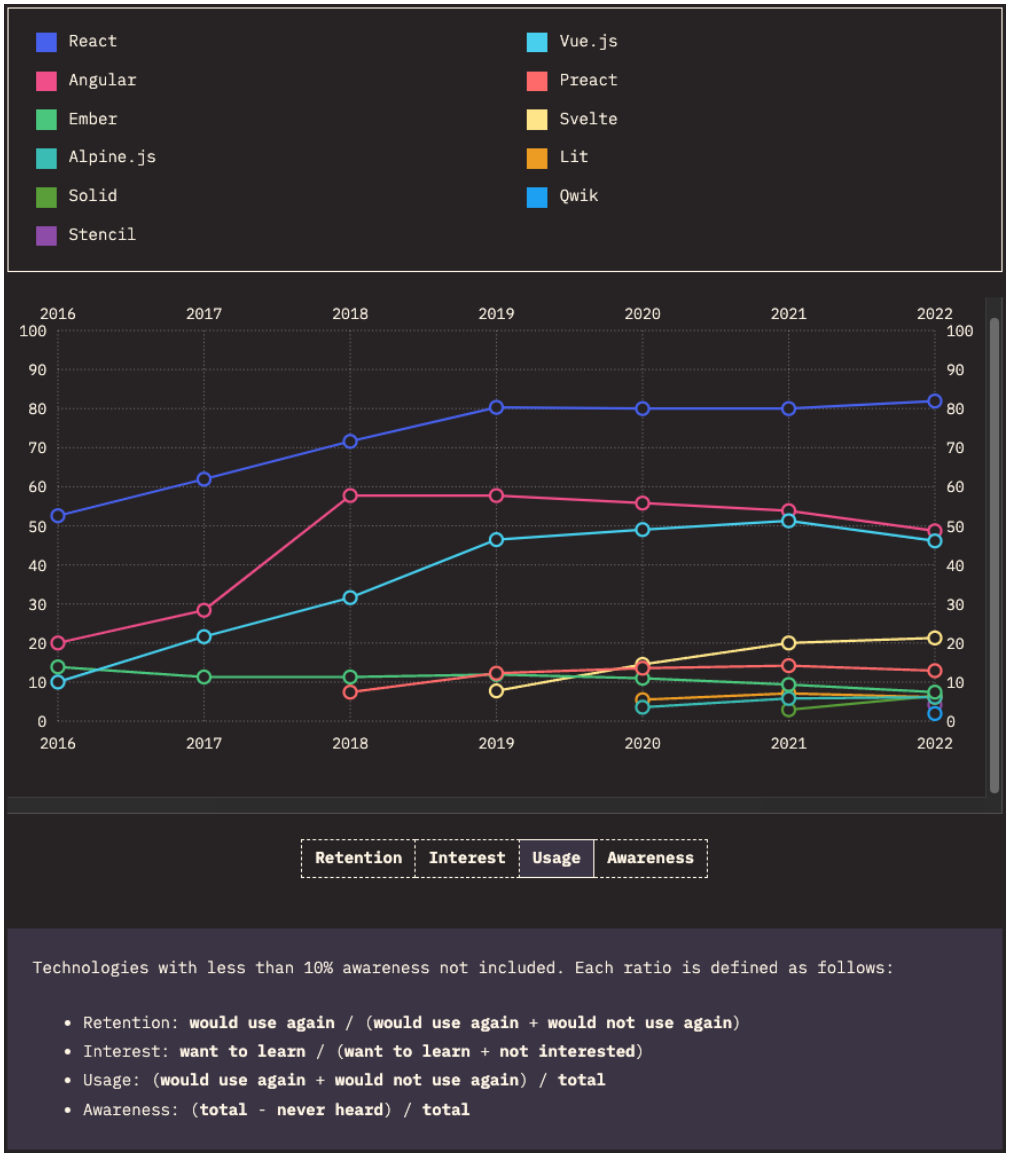
sillä se tarjoaa Youtubea laajemmat käyttökokemuksen säätömahdollisuudet. Esimerkiksi videosoitinta voi muokata yrityksen brändin mukaiseksi, videoiden ja livelähetysten näkyvyyttä voi säätää monella eri tavalla ja lähes kaiken Vimeoon liittyvän hallinnan voi hoitaa ulkoisesti Vimeon oman rajapinnan kautta. Lisäksi Vimeossa olevat videot eivät sisällä mainoksia, mikä on ammattimaisessa käytössä tärkeää. Käyttäjä voi perustaa oman maksullisen VOD-palvelunsa Vimeon sivuille käyttämällä sen on demand -ominaisuutta, jolloin muiden käyttäjien täytyy maksaa nähdäkseen käyttäjän lataamat videot. Toisessa ääripäässä Vimeota voisi käyttää taustalla toimivana videoiden säilytys- ja striimausalustana käyttäjän omalle VOD-sovellukselle. Tällöin Vimeon soitin olisi upotettu käyttäjän sovellukseen ja videoiden hallinta tapahtuisi Vimeon rajapinnan kautta. Videot näkyisivät vain käyttäjän sovelluksessa, eikä niitä pystyisi katsomaan Vimeon sivuilta. Käyttäjän oma sovellus hoitaisi tällöin pääsynhallinnan ja videoiden ostamiseen liittyvän toiminnallisuuden.

3 Moderni web-kehitys

Web-kehityksellä tarkoitetaan selaimella näytettävien verkkosivujen ja sovellusten rakentamista ja ylläpitoa. Nykyaikainen web-kehitys perustuu yksinkertaisimmillaan kolmeen ohjelmointikieleen, jotka ovat HTML, CSS ja JavaScript. Yksinkertaistettuna HTML:n avulla luodaan verkkosivun rakenne, CSS huolehtii sivun ulkoasusta ja JavaScriptillä saadaan sivulle toiminnallisuutta. Näiden lisäksi web-kehityksessä käytetään nykyään usein myös erilaisia sovelluskehyksiä ja kirjastoja. Esimerkkejä tällaisista ovat muun muassa React, Angular, Express ja Bootstrap.

Web-kehitys voidaan jakaa selainpuolen ja palvelinpuolen kehitykseen. Näiden lisäksi web-sovelluksella on yleensä tietokanta. Selainpuolella tarkoitetaan selaimen renderöimää sovelluksen käyttäjälle näkyvää osaa. HTML ja CSS ovat yksinomaan selainpuolen kieliä. JavaScript kehitettiin myös aluksi yksinomaan selainpuolen kieleksi, mutta nykyään sitä käytetään myös palvelinpuolella. Selainpuolen sovelluskehyksistä ja kirjastoista ylivoimaisesti suosituin on React.

Kuten kuvasta 4 voidaan todeta, State of Javascript -kyselyyn osallistuneista yli 80 prosenttia oli käyttänyt Reactia. Seuraavaksi suosituimpia Angular- ja Vue-sovelluskehyksiä oli käyttänyt noin puolet vastaajista. Muiden selainpuolen sovelluskehysten ja kirjastojen käyttö on melko vähäistä verrattuna kolmeen suosituimpaan.



Kuva 4. Käytetyimmät selainpuolen sovelluskehykset ja kirjastot vuosina 2016−2022 State of JavaScript 2022 -kyselyn mukaan. Kyselyyn osallistui vuonna 2022 lähes 40 000 sovelluskehittäjää. (15.)

Web-sovelluksen palvelinpuolella tarkoitetaan sovelluksen käyttäjälle näkymätöntä osaa. Sen tarkoituksena on yleensä sovelluksen tietosisällön

hallinnointi. Se voi esimerkiksi hakea sovellukseen liittyvästä tietokannasta dataa ja lähettää sen selainpuolelle käyttäjälle näytettäväksi. Palvelinpuoli voi myös muun muassa tallentaa, muokata ja poistaa dataa selainpuolelta tulevien pyyntöjen mukaisesti. Sen tehtävänä on usein toimia linkkinä selainpuolen ja tietokannan välissä. Koska sovelluksen käyttäjillä ei ole pääsyä näkemään palvelinpuolen koodia, tulisi sovelluksen arkaluontoiset tiedot käsitellä siellä ja jakaa selainpuolelle vain ne tiedot, joita senhetkisellä käyttäjällä on oikeus nähdä.

3.1 TypeScript-ohjelmointikieli

TypeScript on Microsoftin työntekijöiden 2010-luvun alussa luoma avoimen lähdekoodin ohjelmointikieli. Se ei varsinaisesti ole kokonaan oma ohjelmointikielensä, vaan käytännössä kyse on JavaScript-kielestä, johon on lisätty uusia ominaisuuksia. Kielenä toimimisen lisäksi TypeScript myös tarkistaa kirjoitetun koodin, ilmoittaa mahdollisista virheistä ja muuttaa koodin vastaavaksi JavaScript-koodiksi. (16.)

JavaScript on nykyään ylivoimaisesti suosituin toiminnallinen ohjelmointikieli web-kehityksessä, ja sen suurimpia vahvuuksia on sen joustavuus. JavaScript on heikosti tyypitetty kieli, ja se antaa kehittäjälle paljon vapauksia eikä puutu juurikaan siihen, millaista koodia kirjoitetaan. Pienessä sovelluksessa tästä ei tule suurtakaan ongelmaa, mutta sovelluksessa, jossa koodia on paljon, JavaScriptin antamista vapauksista tulee nopeasti rasite. (16.) Esimerkiksi, jos sovelluksessa monessa paikassa käytettävän muuttujan arvo muutettaisiin jälkikäteen erityyppiseksi, sovellus saattaisi kaatua, sillä kaikki muuttujalle koodissa tehdyt toimenpiteet eivät välttämättä tukisi uudentyyppistä arvoa. JavaScript ei varoittaisi tästä ongelmasta, eikä käyttäjä voisi olla varma, tuleeko ohjelma kaatumaan ilman, että hän ajaisi sen. TypeScript taas antaa käyttäjän määrittää muuttujan luomisvaiheessa sille tyypin, ja se huomaa välittömästi, jos muuttujaan yritetään laittaa jonkin muun tyyppistä arvoa. Se huomaa tämän myös silloin, kun tyyppiä ei ole muuttujalle erikseen määritetty. Tällöin TypeScript varoittaa käyttäjää tästä eikä käyttäjän tarvitse ajaa ohjelmaa

todetakseen siinä olevan virheen. Se osaa myös varoittaa etukäteen, jos muuttuja ei tue sille tehtävää toimenpidettä.

Esimerkkikoodi 1 havainnollistaa edellä kuvattua tilannetta. Kun käyttäjä yrittää laittaa numeroTyppinenMuuttuja-muuttujaan string-tyyppistä arvoa, TypeScript huomaa tämän heti, alleviivaa virhekohdan punaisella ja varoittaa käyttäjää esimerkkikoodissa 1 olevan kommentin tekstillä. Jos vastaavan koodin kirjoittaisi JavaScript-tiedostoon, käyttäjää ei varoitettaisi millään tavalla. Kun ohjelma suoritettaisiin, se kuitenkin kaatuisi, sillä string-tyyppisestä muuttujasta puuttuu numerotyppisistä muuttujista löytyvä toFixed()-toiminto.

```
let numeroTyyppinenMuuttuja: number = 10.00;

numeroTyyppinenMuuttuja = ’10.00’;

// Type 'string' is not assignable to type 'number'.

numeroTyyppinenMuuttuja.toFixed();
```

Esimerkkikoodi 1. TypeScript-koodin osa, joka sisältää virheen. Viidennellä rivillä oleva kommentti kertoo, millaisen varoituksen TypeScript antaa koodissa olevasta virheestä.

Esimerkkikoodi 2 havainnollistaa, että TypeScript osaa havaita virheen silloinkin, kun muuttujan alkuperäisen arvon tyyppi ei tue muuttujalle tehtävää toimenpidettä. Kuten edellä mainittiin, toFixed()-toimintoa ei löydy string-tyyppisestä muuttujasta ja jos kyseinen ohjelma suoritettaisiin, se kaatuisi. TypeScript kuitenkin huomaa tämän virheen jo etukäteen ja alleviivaa toFixed()-tekstin punaisella. Lisäksi TypeScript antaa esimerkkikoodissa 2 olevan kommentin varoituksen. JavaScript ei huomaisi tätäkään virhettä, ennen kuin ohjelma suoritettaisiin.

```
let stringTyyppinenMuuttuja = ’10.00’;

stringTyyppinenMuuttuja.toFixed();

// Property 'toFixed' does not exist on type 'string'.
```

Esimerkkikoodi 2. Hieman muunneltu versio esimerkkikoodin 1 TypeScript-koodista. Tämä voisi olla myös JavaScript-tiedostoon kirjoitettua koodia, mutta

viimeisen rivin kommentti kertoo, millaisen varoituksen TypeScript antaisi koodissa olevasta virheestä.

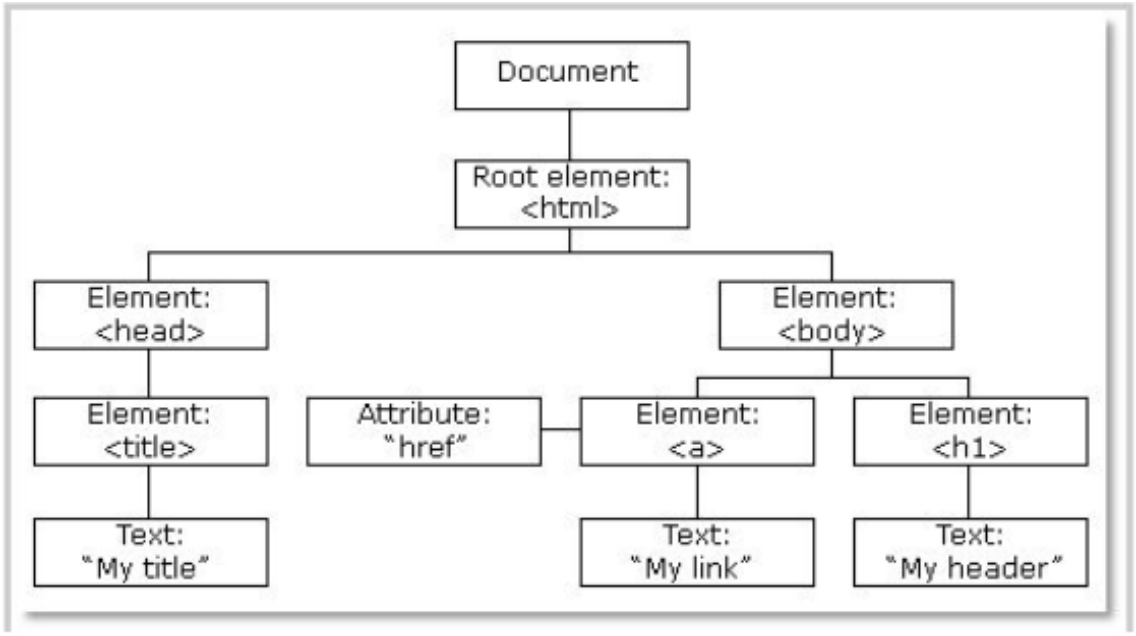
TypeScript on olio-ohjelmointikieli, eli toisin kuin JavaScript, se tukee luokkia ja niihin liittyviä konsepteja, kuten interface ja inheritance. Tämä tekee koodista helpommin ylläpidettävää ja järjestettävää. Kuten aiemmin mainittiin, TypeScript antaa käyttäjän määrittää muuttujille tyypit. Tämä tekee koodista helppolukuisempaa ja luotettavampaa. TypeScript Language Service (TLS) on TypeScriptiin kuuluva sovelluskerros, joka auttaa koodieditoreja muun muassa koodin muotoilussa, värittämisessä ja automaattisessa täydennyksessä. TLS:n avulla TypeScript pystyy myös tulkitsemaan muuttujan tyypin, vaikka käyttäjä ei olisi sitä erikseen määritellyt. Näin ollen tyypin määrittäminen muuttujalle ei TypeScriptissä ole pakollista. (17.)

On olemassa myös muita ohjelmointikieliä, jotka muuttavat koodinsa JavaScriptiksi. Näistä suosituimmat ovat CoffeeScript ja Dart. TypeScriptiin verrattuna niiden käyttö on kuitenkin melko marginaalista. Listalla, jolla käytetään mittarina GitHubiin tehtyjä koodin puskemisia, 50 suosituimman ohjelmointikielen joukosta vuoden 2022 viimeisellä vuosineljänneksellä TypeScript löytyy sijalta viisi. 8,176 prosenttia kaikista puskemisista oli tehty TypeScript-projekteihin. Dart löytyy samalta listalta sijalta 21 0,410 prosentin osuudellaan ja CoffeeScript sijalta 42 0,081 prosentilla. JavaScript on listalla kolmantena. Sitä oli käytetty kaikkiaan 11,016 prosentissa kaikista tilaston piiriin kuuluvista projekteista. (18.)

3.2 React-kirjasto

React on JavaScript-pohjainen selainpuolen käyttöliittymien kehitykseen tarkoitettu kirjasto. Se perustuu käyttöliittymäkomponentteihin ja, kuten luvun 3 alussa mainittiin, se on nykyään suosituin web-kehityksessä selainpuolella käytettävä kirjasto. Se kehitettiin alun perin vuonna 2011 käytettäväksi Facebookin uutissyötteen kehityksessä, ja vuonna 2013 se julkaistiin avoimen lähdekoodin kirjastona yleiseen käyttöön. (19; 20.)

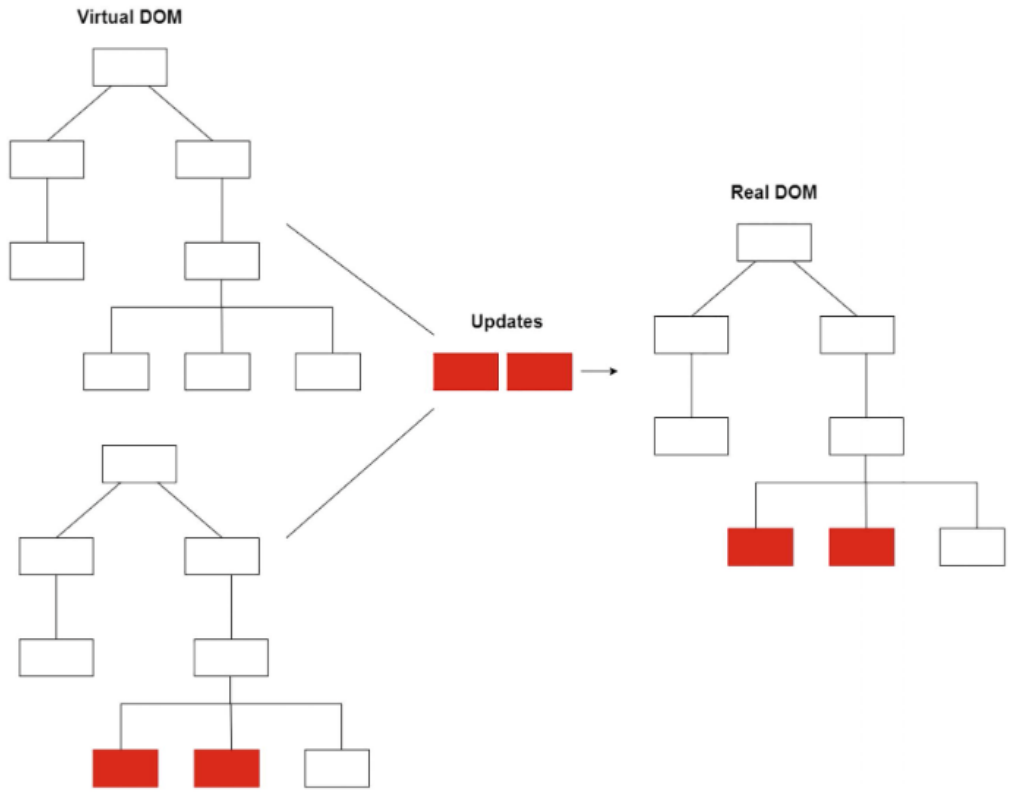
Jokaisen selaimen selainmoottori muodostaa sille annetusta HTML-koodista Document Object Model (DOM) -puun. Kuvassa 5 on esimerkki tällaisesta DOM-puusta. Puu koostuu elementeistä ja niihin liittyvistä attribuuteista ja sisällöistä. Ne yhdessä muodostavat HTML-dokumentin. Selain muodostaa myös niin sanotun CSS Object Model (CSSOM) -puun, joka määrittää DOM-puun elementeille tyylit. DOM- ja CSSOM-puut muodostavat yhdessä render-puun, joka nimensä mukaisesti renderöi elementit tyyleineen selaimeen ja näin näyttää sisällön näytöllä. (19.)



Kuva 5. Esimerkki DOM-puusta (20).

Sovelluksessa, joka koostuu vain HTML-tiedostosta ja siihen liitetyistä CSS- ja JavaScript-tiedostoista, sivun sisällön muuttaminen renderöi jokaisen muutettavan kohdan erikseen. Esimerkkinä voidaan käyttää edellä mainitun kaltaista sovellusta, joka sisältää napin, jota painamalla kahden p-elementin tekstisisältö muuttuu. Kun yhden tällaisen p-elementin sisältöä muutetaan, selain tulkitsee CSS-koodin uudestaan, validoi muutettavan elementin render-puussa, muodostaa sen layoutin uudelleen ja lopuksi uudelleen renderöi sen ruudulle. Esimerkin tapauksessa nämä samat toimenpiteet tehdään erikseen kahdelle p-elementille ja näin selain joutuu päivittämään DOM-puun kahdesti.

Varsinkin suurissa sovelluksissa tällainen DOM-puun päivittäminen jokaisen muutettavan elementin kohdalla hidastaa sovelluksen toimintaa huomattavasti. React ratkaisee tämän ongelman käyttämällä niin sanottua virtuaalista DOM (VDOM) -puuta, joka kuvataan kuvassa 6. (19.)



Kuva 6. Kaavio kuvaa, kuinka React päivittää DOM-puun elementtejä. Punaiset laatikot kuvaavat elementtejä, joita on muutettu. (19.)

React luo kaksi kopiota DOM-puusta muistiinsa. Kun elementtejä muutetaan, React muuttaa toista DOM-kopioistaan ja vertaa sitä toiseen kopioon selvittääkseen, mitä on muutettu. Tämän jälkeen se kerää muutokset yhteen, ryhmittelee ne ja liittää ne oikeaan DOM-puuhun yhdellä kerralla. Koska sovelluksen DOM-puuta päivitetään vain kerran riippumatta siitä, kuinka montaa elementtiä muutetaan, parantaa Reactin käyttö sovelluksen suorituskykyä huomattavasti. (19; 20.)

Reactissa käyttöliittymä muodostuu komponenteista, jotka ovat oikeastaan JavaScript-funktioita. Nämä komponentit palauttavat JavaScript XML (JSX) -

koodia tai, jos kielenä on käytetty TypeScriptiä, TypeScript XML (TSX) -koodia. JSX muistuttaa HTML:n syntaksia, mutta kyse on kuitenkin eri koodista. Sen avulla kuvataan, millaiselta käyttöliittymän tulisi näyttää. JSX-tiedostossa määritellään siis kyseisen komponentin HTML- ja JavaScript-koodi. (19.)

Esimerkkikoodissa 3 on melko yksinkertainen App-niminen React-komponentti. Komponentin alussa luodaan kaksi state-objektia, aika ja numero. State-objektin avulla voidaan Reactissa muuttaa komponentin sisältöä dynaamisesti. Tässä tapauksessa molempien state-objektien alkuperäisiksi vakioarvoiksi on määritelty 0. State-objektien jälkeen komponenttiin on lisätty Reactin oma useEffect-funktio, jonka sisältö ajetaan, kun komponentti, johon kyseinen useEffect on lisätty, renderöidään. Jos useEffect-funktion lopussa olevien hakasulkeiden sisään on lisätty yksi tai useampi state, useEffect-funktion sisältö ajetaan myös silloin, kun jokin näistä state-objekteista päivittyy. Esimerkkikoodissa 3 useEffect-funktion sisältö ajetaan alussa komponentin renderöimisen yhteydessä ja aina, kun aika-staten arvo muuttuu.

Kun esimerkkikoodissa 3 oleva App-komponentti renderöidään, p-elementeistä ylemmän sisältö on ”Keskiyöstä 1.1.1970 on aikaa 0 millisekuntia”, sillä aika-staten vakioarvoksi on määritelty 0 eikä kyseistä state-objektia ole vielä päivitetty. Sen sijaan alemman p-elementin sisältönä on jokin numero 0 ja 100 välillä, sillä numero-staten uudeksi arvoksi päivitetään jokin numero tältä väliltä useEffect-funktiossa. Aika-staten tapaan myös numero-staten alkuperäinen arvo on 0, mutta tämä state päivitetään komponentin renderöimisen yhteydessä niin nopeasti, että alkuperäistä arvoa ei ehditä näyttää. Kun komponentissa olevaa nappia painetaan, päivitetään myös aika-staten arvo osoittamaan, kuinka monta millisekuntia on kulunut keskiyöstä 1.1.1970. Samalla päivittyy myös alemmassa p-elementissä oleva numero, sillä aika-staten arvon päivitys laukaisee useEffect-funktion.

```
import { useEffect, useState } from 'react';

const App = () => {
    const [aika, setAika] = useState(0);
    const [numero, setNumero] = useState(0);

    useEffect(() => {
        setNumero(Math.random() * 100);
    }, [aika]);

    const paivitaAika = () => {
        setAika(Date.now());
    }

    return (
        <div>
            <p>
                Keskiyöstä 1.1.1970 on aikaa {aika} millisekuntia.
            </p>
            <p>
               {numero}
            </p>
            <button onClick={paivitaAika}>Päivitä aika</button>
        </div>
    );
};

export default App;
```

Esimerkkikoodi 3. App-niminen React-komponentti. Se palauttaa div-elementin, joka sisältää kaksi p-elementtiä ja nappielementin. Nappia painamalla p-elementtien sisältö muuttuu.

Kaiken kaikkiaan Reactin vahvuuksiin kuuluu se, että sillä pystyy luomaan hyvin dynaamisia sovelluksia helposti ja melko vähällä koodilla. Sen komponentit selkeyttävät koodia ja mahdollistavat sen uudelleenkäytön helposti useammassa paikassa. Reactin virheenjäljitykseen löytyy myös työkaluja, jotka tekevät siitä vaivatonta. Esimerkiksi Chrome-selaimeen löytyy laajennuksia, joiden avulla pääsee seuraamaan muun muassa Reactin state-objektien päivittymistä. (20.)

3.3 Node.js-suoritusympäristö

Node.js on vuonna 2009 julkaistu avoimen lähdekoodin alustariippumaton JavaScriptin suoritusympäristö. Alustariippumattomuudella tarkoitetaan sitä, että se toimii eri käyttöjärjestelmillä, kuten Windowsilla, Linuxilla ja macOS-käyttöjärjestelmällä. Ennen Node.js:n kehitystä JavaScript-koodia pystyttiin

suorittamaan vain selaimilla, sillä ainoastaan niissä oli suoritusympäristö JavaScriptille. Tämä tarkoitti sitä, että JavaScriptiä pystyttiin käyttämään vain selainpuolen sovellusten kehittämiseen. Node.js on kehitetty Google Chrome -selaimen käyttämän V8-JavaScript-moottorin päälle, mikä mahdollistaa myös palvelinpuolen sovellusten kehittämisen JavaScriptillä. (21.)

Node.js-sovellus toimii yhdessä säikeessä toisin kuin monet perinteiset palvelinpuolen ohjelmat, jotka luovat uuden säikeen jokaisen niihin tulleen pyynnön suoritusta varten. Se luo asynkronisesti toimivan event loopin. Tämä tarkoittaa sitä, että Node.js ei pysäytä koodin suoritusta, kun se esimerkiksi odottaa tietokannasta dataa, vaan ajaa useita pyyntöjä samanaikaisesti taustaprosesseina. Näin ollen yhden pyynnön suorittaminen ei estä muiden samanaikaisten pyyntöjen suoritusta, kuten monissa perinteisissä palvelinpuolen ohjelmissa. Node.js-sovellukset ovatkin hyvin skaalautuvia, sillä ne pystyvät käsittelemään tuhansia samanaikaisia pyyntöjä yhdessä säikeessä, toisin kuin perinteiset palvelinpään ohjelmat, jotka joutuvat luomaan jokaiselle pyynnölle oman säikeensä ja näin ollen kuluttavat huomattavasti enemmän palvelimen resursseja. Yhden säikeen arkkitehtuurinsa takia Node.js ei kuitenkaan sovellu käytettäväksi sovelluksissa, jotka vaativat palvelimen prosessorilta hyvin paljon laskentatehoa. Tällöin säikeen raskas kuormitus saattaisi heikentää palvelimen suorituskykyä ja jopa estää uusien pyyntöjen käsittelyn. (22; 23.)

Yksi suosituimpia Node.js:n ominaisuuksia on sen mukana tuleva pakettimanageri Node Package Manager (NPM). NPM sisältää miljoonia paketteja, ja se onkin maailman suurin ohjelmistorekisteri. (22.) Tällainen paketti sisältää jonkin avoimen lähdekoodin moduulin tarvitsemat tiedostot. Moduulilla tarkoitetaan JavaScript-kirjastoa, joka sisältää Node.js-projektin kehitystä helpottavia työkaluja ja funktioita. (24.) Node.js-paketin käyttöönotto on yksinkertaista. Ensin kirjoitetaan Node.js-projektin kansiossa terminaaliin npm install -komento ja sen jälkeen halutun paketin nimi. Kun tämän komennon suorittaa, NPM asentaa halutun paketin projektiin. Tämän jälkeen paketin sisältämä moduuli tuodaan tiedostoon, jossa sitä tullaan käyttämään import-

avainsanalla. Esimerkkikoodissa 4 tuodaan import-avainsanalla http-objekti http-moduulista ja luodaan tämän avulla yksinkertainen Node.js-palvelinohjelma. Tämä ohjelma palauttaa statuskoodin 200 ja Hello world -tekstin, kun käyttäjä lähettää pyynnön osoitteeseen 127.0.0.1:3000. Esimerkiksi, jos kyseiseen osoitteeseen menee selaimen avulla, selain näyttää kyseisen Hello world -tekstin. Http-moduuli tulee Node.js:n asennuksen mukana, joten sitä ei tarvitse erikseen asentaa npm-komennolla.

```
import { http } from 'http';

const host = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, host, () => {
  console.log(`Server running at http://${host}:${port}/`);
});
```

Esimerkkikoodi 4. Yksinkertainen Node.js-palvelinohjelma, joka kuuntelee porttia 3000 käyttäjän oman laitteen localhost-osoitteessa.

Statistan kyselyn mukaan Node.js on maailman eniten käytetty sovelluskehys web-kehityksessä vuonna 2022 (25). Sitä käytetään yli 30 miljoonalla verkkosivulla maailmanlaajuisesti, ja sitä on ladattu arviolta noin 1,5 miljardia kertaa (26). Maailmanlaajuisesti merkittävistä yrityksistä muun muassa Netflix, PayPal, LinkedIn, Twitter ja eBay käyttävät sitä (22).

3.4 NestJS-sovelluskehys

NestJS on avoimen lähdekoodin Node.js-sovelluskehys selainpuolen sovellusten kehittämiseen. Se on rakennettu TypeScriptin avulla, joten se tukee kaikkia TypeScriptin ominaisuuksia, kuten virheen havaitsemisen ennen koodin ajoa. Ellei toisin määritellä, NestJs käyttää erittäin suosittua Express.js-sovelluskehystä pyyntöjen prosessointiin, ja se tukeekin kaikkia Express.js:n ominaisuuksia. Myös Fastifyn, toisen suositun Node.js-sovelluskehyksen

ominaisuuksia voi hyödyntää NestJS:n avulla. Näiden lisäksi siihen on sisäänrakennettu monia moduuleja, jotka esimerkiksi pelkkää Express.js-sovelluskehystä käyttäessä pitäisi asentaa erikseen. Tällaisia ovat esimerkiksi tiedostojen lataukseen tarkoitettu Multer ja autentikointeihin keskittyvä Passport.js. NestJS sisältää myös valmiin tuen muun muassa Websocket-yhteyksille, GraphQL-kyselykielelle ja pyyntöjen validoinnille. Sen arkkitehtuuriin on otettu hyvin paljon vaikutteita Angular-sovelluskehyksestä. (27; 28.)

NestJS:n ydinkomponentteihin kuuluvat kontrollerit, provider-luokat ja moduulit. Kontrollerit huolehtivat sovellukseen tulevien pyyntöjen käsittelystä ja oikeanlaisen vastauksen lähettämisestä. (27.) Esimerkkikoodissa 5 on yksinkertainen kontrolleri, joka huolehtii sovellukseen tulevien GET-pyyntöjen käsittelystä ajamalla getHello()-metodin. Tämä palauttaa EsimerkkiService-nimisen service-luokan getHello()-metodin palauttaman vastauksen pyynnön lähettäneelle käyttäjälle.

```
import { Controller, Get } from '@nestjs/common';
import { EsimerkkiService } from './esimerkki.service';

@Controller()
export class EsimerkkiController {
  constructor(private readonly esimerkkiService: EsimerkkiService) {}

  @Get()
  getHello(): string {
    return this.esimerkkiService.getHello();
  }
}
```

Esimerkkikoodi 5. Yksinkertainen NestJS-kontrolleri.

Provider-luokkien tehtävänä on yleensä sovelluksen monimutkaisempien tehtävien suoritus. Provider-luokkiin kuuluvat muun muassa service- ja repository-luokat. Esimerkiksi tiedon tallennus tietokantaan tapahtuu tyypillisesti NestJS-sovelluksessa niin, että kontrolleri ottaa tiedon vastaan, käsittelee sen ja lähettää sen service-luokalle. Service-luokassa tieto tallennetaan repository-luokan avulla tietokantaan. Luokan voi määrittää NestJS:ssä provider-luokaksi lisäämällä siihen @Injectable()-dekoraattorin. Provider-luokan voi tuoda toiseen luokkaan kyseisen luokan constructor-metodin avulla, jolloin NestJS huolehtii

automaattisesti luokkien välisistä riippuvuuksista. (26.) Esimerkkikoodi 6 sisältää esimerkkikoodin 5 kontrolleriin tuodun EsimerkkiService-luokan, jonka getHello()-metodi palauttaa Hello World -tekstin. Käyttäjä, joka lähettää GET-pyynnön esimerkkikoodien 5 ja 6 kuvaamaan sovellukseen, saa siis vastauksena tuon kyseisen tekstin.

```
import { Injectable } from '@nestjs/common';

@Injectable()
export class EsimerkkiService {
  getHello(): string {
    return 'Hello World';
  }
}
```

Esimerkkikoodi 6. Yksinkertainen NestJs:n service-luokka.

NestJS:n moduulit ovat luokkia, jotka luodaan @Module()-dekoraattorilla. Niiden tarkoitus on jäsennellä sovelluksen rakennetta ja liittää yhteen toisiinsa läheisesti liittyvät toiminnot. Jokaisella NestJS-sovelluksella on vähintään root-moduuli. Se sisältää kaikki sovelluksen muut moduulit tai, jos muita moduuleita ei ole, kaikki sovelluksen provider-luokat. NestJS käyttää sitä aloituspisteenä, kun se selvittää sovelluksen rakennetta ja riippuvaisuuksia. On hyvin suositeltavaa käyttää root-moduulin lisäksi sovelluksessa myös muita moduuleita, jotta sovelluksen rakenne pysyy selkeänä. (27.) Esimerkkikoodin 7 moduulissa toiminnallisesti toisiinsa liittyvät esimerkkikoodin 5 kontrolleri ja esimerkkikoodin 6 service on määritelty kuuluvaksi kyseiseen moduuliin. Tämä moduuli muodostaa yhden toiminnallisen kokonaisuuden ja täten selkeyttää sovelluksen rakennetta.

```
import { Module } from '@nestjs/common';
import { EsimerkkiController } from './esimerkki.controller';
import { EsimerkkiService } from './esimerkki.service';

@Module({
  controllers: [EsimerkkiController],
  providers: [EsimerkkiService],
})
export class EsimerkkiModule {}
```

Esimerkkikoodi 7. Yksinkertainen NestJS-moduuli.

4 Työn tavoitteet ja lähtökohdat

4.1 Tavoitteet

Insinöörityön tavoitteena oli luoda Pharmac Finland Oy -nimiselle yritykselle alusta, jonka kautta yritys voisi myydä katseluoikeuksia tuottamilleen videomuotoisille koulutuksille ja webinaareille. Pharmac Finland Oy on Vantaalla toimiva pääosin lääkkeiden koneellista annosjakelua tuottava yritys, joka on viime vuosina laajentanut tarjoamansa palvelut käsittämään myös muun muassa pääosin apteekeille tarjottavat videomuotoiset koulutukset. Tähän mennessä Pharmac on myynyt koulutuksiaan lataamalla ne ensin DreamBroker-videopalveluun ja lähettämällä sitten tietyn ajan voimassa olevia linkkejä videoihin niitä ostaneille asiakkaille sähköpostin välityksellä. Tämä vaatii kuitenkin Pharmacin työntekijöiltä turhan paljon työtä, sillä heidän on jokaisen tilauksen yhteydessä luotava tilaajalle oma linkki videoon, lähetettävä se tilaajalle sähköpostitse ja muistettava käydä mitätöimässä linkki, kun tilauksen voimassaoloaika päättyy. Tässä uudessa palvelussa tilauksen ja sen voimassaolon hallinnan tuli tapahtua automaattisesti.

Uusi VOD-palvelu tuli liittää osaksi yrityksen kehitteillä olevaa ekstranetportaalia. Ekstranetin tarkoitus on toimia alustana, josta Pharmacin asiakkaat voivat löytää keskitetysti kaikki yrityksen heille tarjoamat palvelut. Ekstranetportaaliin kuuluu myös hallintaosio, jonka kautta yrityksen työntekijät, joilla on siihen oikeus, voivat hallita ekstranetissä näytettävää sisältöä. Tavoitteena oli, että työn tuotoksena syntyvä alusta jakautuisi kolmeen eri paikkaan ekstranetissä. Katseluoikeudet videomuotoisiin koulutuksiin ja webinaareihin ostettaisiin verkkokauppaosiossa. Ostettuja koulutuksia katseltaisiin taas niin sanotussa materiaalipankkiosiossa, jossa videomateriaalien lisäksi voisi olla myös muun tyyppisiä materiaaleja, kuten lomakkeita ja PDF-dokumentteja. Tähän projektiin kuului kuitenkin vain videomuotoisten materiaalien näyttäminen materiaalipankissa. Kolmas paikka,

johon tämän työn toiminnallisuuden suunniteltiin jakautuvan, on ekstranetin hallintaosio. Tavoitteena oli, että videoiden hallinta sisällönhallitsijoiden toimesta tapahtuisi tässä osiossa.

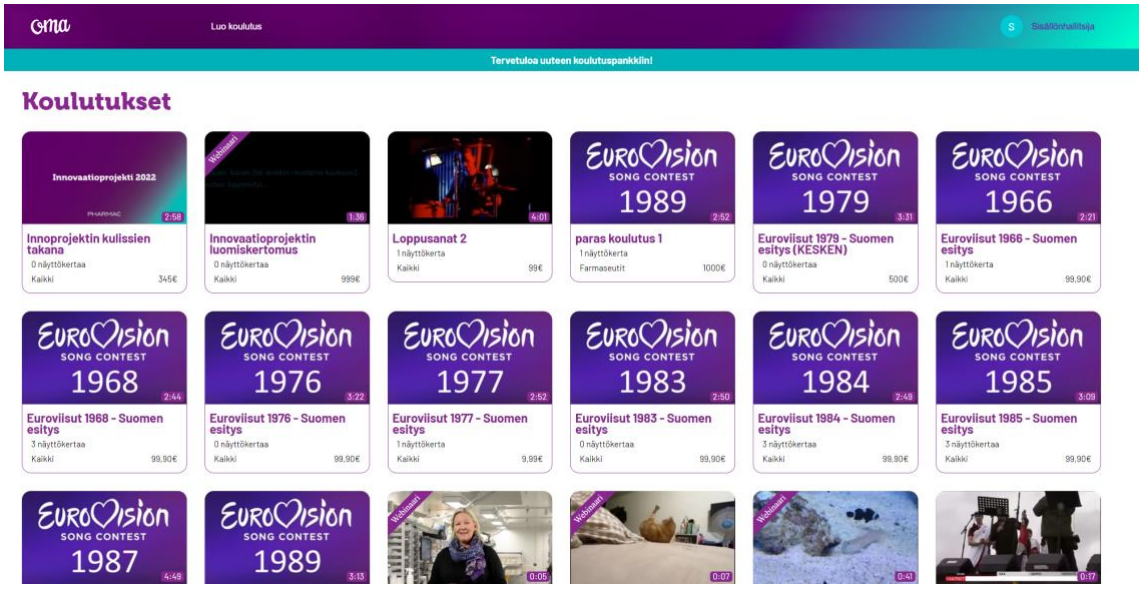
Tämän työn tuotoksena syntyvässä VOD-palvelussa tuli pystyä selaamaan kirjautuneen käyttäjän käyttäjäryhmälle tarjolla olevia koulutuksia, tarkastella koulutuksiin liittyviä tarkempia tietoja, katsoa niistä tehtyjä teaser-videoita, ostamaan katseluoikeuksia koulutuksiin kuukaudeksi, katsomaan koulutuksia, joihin käyttäjällä on voimassa oleva tilaus ja tykkäämään niistä tykkäysnapin avulla. Lisäksi Pharmacin työntekijöiden, joilla on sisällönhallitsijan oikeudet tuli pystyä luomaan uusia koulutustuotteita sekä muokkaamaan ja poistamaan niitä. Koulutustuotteiden luomisen yhteydessä niihin tuli pystyä lisäämään muun muassa nimi, kuvaus, koulutusvideo, teaser-video, vapaaehtoinen esikatselukuva, hinta ja käyttäjäryhmät, joille kyseinen koulutustuote näkyy. Koulutustuotteiden piti myös tukea ekstranetin kolmea kielivaihtoehtoa, jotka ovat suomi, ruotsi ja englanti. Palvelun tuli olla myös helppokäyttöinen, hyvännäköinen ja responsiivinen.

4.2 Prototyyppi

Palvelun suunnitteleminen aloitettiin alun perin jo elokuussa 2022, sillä siitä tehtiin syksyn aikana prototyyppi Metropolian innovaatioprojektissa viiden yrityksessä työskentelevän opiskelijan yhteistyönä. Lähtökohtana oli, että videot striimattaisiin ulkoisen videonjakelualustan kautta, sillä yrityksellä ei ole kapasiteettia striimata videoita omilta palvelimiltaan. Kun eri vaihtoehtoja oli kartoitettu, päädyttiin valitsemaan striimausalustaksi Vimeo ja sen tarjoamista paketeista Advanced. Valinta oli lopulta helppo, sillä Vimeo tarjosi selkeästi muita vaihtoehtoja enemmän tallennuskapasiteettia ja kaistanleveyttä muita edullisempaan hintaan. Lisäksi yksi tärkeimmistä kriteereistä striimausalustaa valittaessa oli mahdollisuus rajoittaa videoiden näkyvyyttä niin, että ne näkyisivät vain Pharmacin tulevassa VOD-palvelussa. Vimeossa tämä onnistuu verkkotunnus-rajoituksen avulla. Videon voi siis määrittää näkyväksi vain tietyssä verkkotunnuksessa olevalle sivulle upotettuna. Kaikki videoihin liittyvät

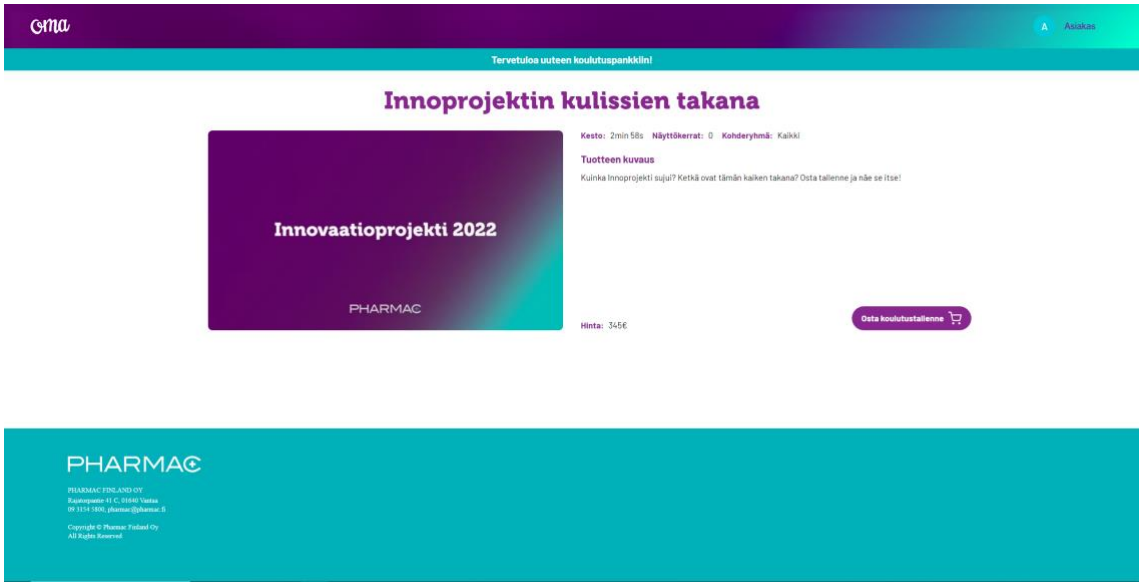
hallintatoimenpiteet haluttiin myös saada integroitua tulevaan VOD-palveluun, mikä onnistui Vimeon kattavan avoimen rajapinnan avulla. Prototyypin selainpuoli toteutettiin Reactin avulla, ja ohjelmointikielenä käytettiin TypeScriptiä. Prototyypin palvelinpuoli rakennettiin Node.js:n, Express.js-sovelluskehyksen ja TypeScriptin avulla. Tämän insinöörityön tuotoksesta poiketen prototyyppiä ei liitetty osaksi ekstranettiä, vaan siitä tuli erillinen sovellus.

Prototyyppi saatiin onnistuneesti valmiiksi, ja se esiteltiin osalle yrityksen työntekijöistä, jotka myös pääsivät kokeilemaan sitä. Siitä saatua palautetta ja jatkokehitysideoita hyödynnettiin tämän työn kehityksessä. Kuvassa 7 näkyy prototyypin pääsivulla oleva kaikkien saatavilla olevien koulutustallenteiden korttimuotoinen listaus. Korttien kuvat haettiin Vimeosta, josta ne saatiin kyseisen koulutuksen video-objektin mukana. Jos koulutuksen luonut käyttäjä ei ollut tallentanut erikseen kuvaa koulutukselle, Vimeo lähetti kyseisestä videosta otetun kuvan. Myös kuvien päällä näkyvät videon kestot saatiin Vimeon avoimesta rajapinnasta tulevien video-objektien mukana. Koulutusten lataamisesta etusivulle tuli kuitenkin melko hidasta varsinkin, jos koulutuksia oli paljon, sillä edellä mainitut kuva ja kesto jouduttiin hakemaan jokaiselle koulutukselle erikseen Vimeon rajapinnasta. Tuon ongelman ratkaiseminen oli yksi tämän työn keskeisimmistä lähtökohdista.



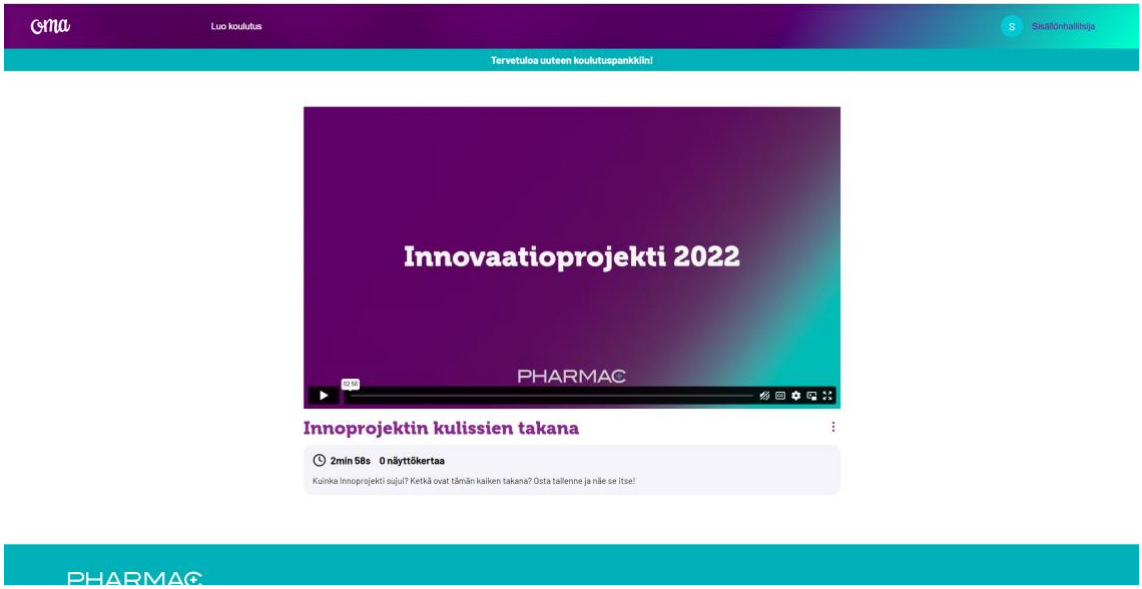
Kuva 7. VOD-palvelusta tehdyn prototyypin etusivu.

Prototyyppiin luotiin kaksi demokäyttäjää: sisällönhallitsija ja asiakas. Nämä eivät olleet oikeita käyttäjiä, vaan kyse oli vain Reactin state-objektiin tallennetusta arvosta. Kuvassa 7 näkyvää sovelluksen yläpalkin oikeassa laidassa olevaa tekstiä klikkaamalla pystyi muuttamaan aktiivista käyttäjää. Demokäyttäjät luotiin havainnollistamaan, miten sovelluksessa näkyvä sisältö riippuu siitä, käyttääkö sovellusta yrityksen sisällönhallitsija vai ulkoinen asiakas. Kuvassa 8 on prototyypin yksittäisen koulutuksen ostonäkymä. Jos kyseiseen koulutukseen olisi liitetty teaser-video, se näkyisi otsikon alla vasemmalla olevan kuvan tilalla.



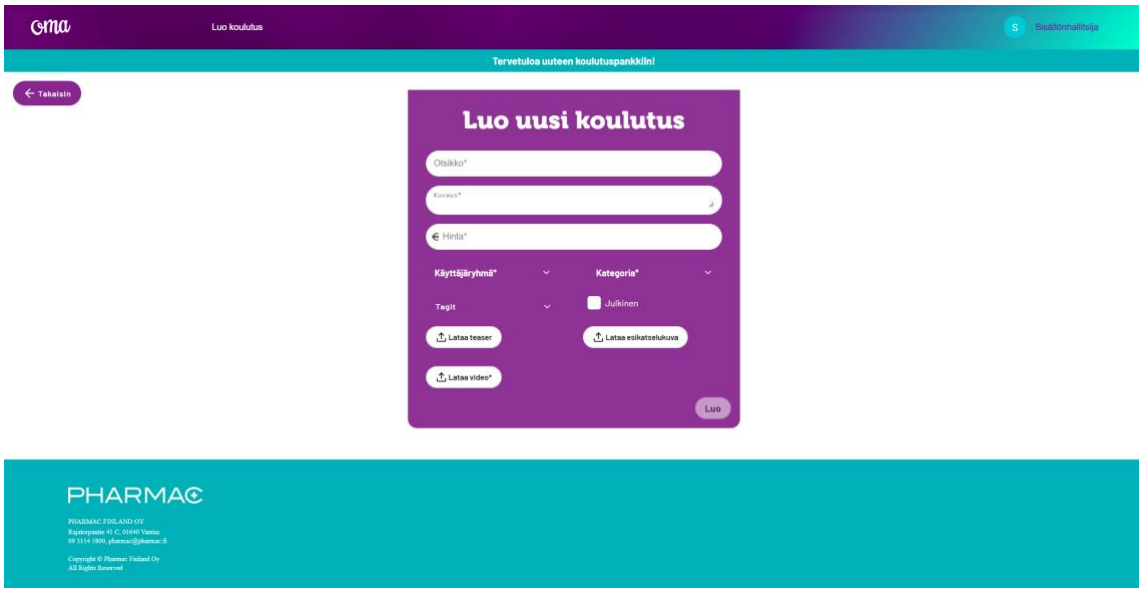
Kuva 8. Koulutuksen ostonäkymä VOD-palvelun prototyypissä.

Kun koulutuksen osti prototyypissä painamalla kuvassa 8 näkyvää ”Osta koulutustallenne” -nappia, sovellus tallensi koulutuksen ID-numeron selaimen välimuistiin. Tämän jälkeen käyttäjä pääsi koulutusta klikkaamalla suoraan kuvassa 9 näkyvään koulutustallenteen katselunäkymään.



Kuva 9. Koulutustallenteen katselunäkymä VOD-palvelun prototyypissä.

Kuvassa 10 olevaan koulutuksen luontinäkymään pääsi vain, jos oli valinnut käyttäjäksi sisällönhallitsijan. Jos koulutukseen lisäsi tiedostokooltaan suuren videon lisäksi vielä suuren teaser-videon, saattoi koulutuksen luominen epäonnistua. Tarkkaa syytä tähän ei löydetty, mutta ongelma liittyi sovelluksen palvelinpuolen ja Vimeon rajapinnan väliseen yhteyteen, joka katkesi tällaisissa tilanteissa. Koulutuksen muokkaus- ja poistamisominaisuudet löytyivät prototyypissä kuvan 9 näkymässä olevaa kolmea pistettä klikkaamalla, jos valittuna käyttäjänä oli sisällönhallitsija.



Kuva 10. Koulutuksen luontinäkymä VOD-palvelun prototyypissä.

5 Striimauspalvelun toteutus

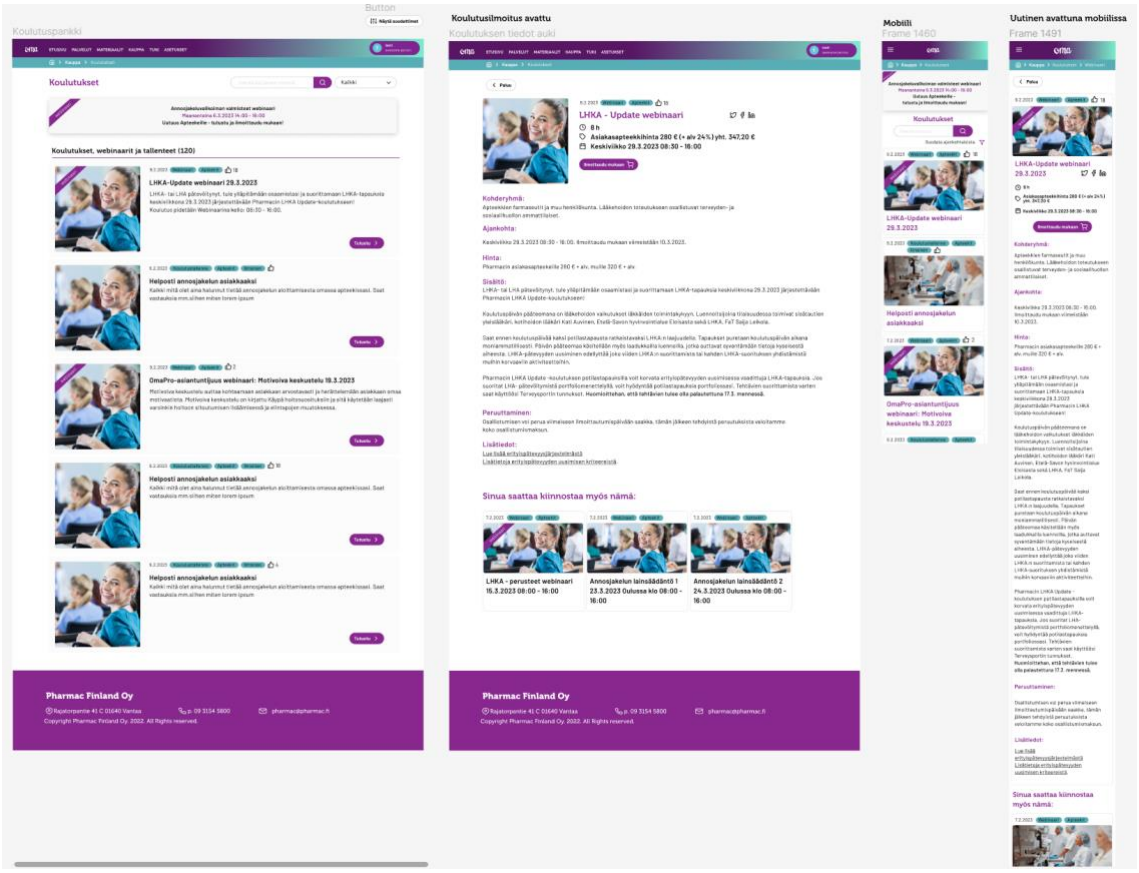
5.1 Suunnittelu

Ennen striimauspalvelun teknisen toteutuksen aloittamista käytiin yrityksen teknologiapäällikön johdolla noin kahden tunnin palaverissa läpi palveluun liittyviä vaatimuksia. Alun perin tavoitteena oli, että palvelu sisältäisi myös maksamistoiminallisuuden. Tämä olisi kuitenkin vaatinut tuotteiden maksutietojen hakemisen yrityksen kehitteillä olevasta sisäisille palveluille tarkoitetusta toiminnanohjausjärjestelmästä Tätä ominaisuutta ei kuitenkaan toiminnanohjausjärjestelmästä vielä löytynyt, joten palaverissa päätettiin jättää maksamistoiminnallisuus tämän työn laajuudesta pois. Tavoitteeksi tuli tehdä tuotteen ostotoiminnallisuus sellaiseksi, että siihen on helppo myöhemmin lisätä tämä maksamistoiminnallisuus.

Palvelun tietokantarakenteesta suunniteltiin huomattavasti laajempi kuin prototyypissä käytetty melko suppea tietokantarakenne. Kun prototyypissä kaikki koulutuksen tiedot säilytettiin yhdessä tietokantataulussa, nyt koulutus jakautui koulutustuotetauluun ja siihen yhdistettyyn kaikkien tuotteiden

yhteiseen tauluun. Jälkimmäinen näistä sisältää kaikenlaisille tuotteille yhteisiä kenttiä, kuten esimerkiksi kentät verokannalle, osto- ja myyntihinnalle sekä myyntitilille. Myös käyttäjärooleille ja tageille luotiin omat tietokantataulunsa, joista määriteltiin yhteydet koulutustuotetauluun. Koulutustuotteen tykkäyksille ja tilauksille taas luotiin välitaulut, jotka yhdistävät käyttäjänhallintajärjestelmästä saadun käyttäjän ID-numeron koulutustuotetaulun ID-numeroon.

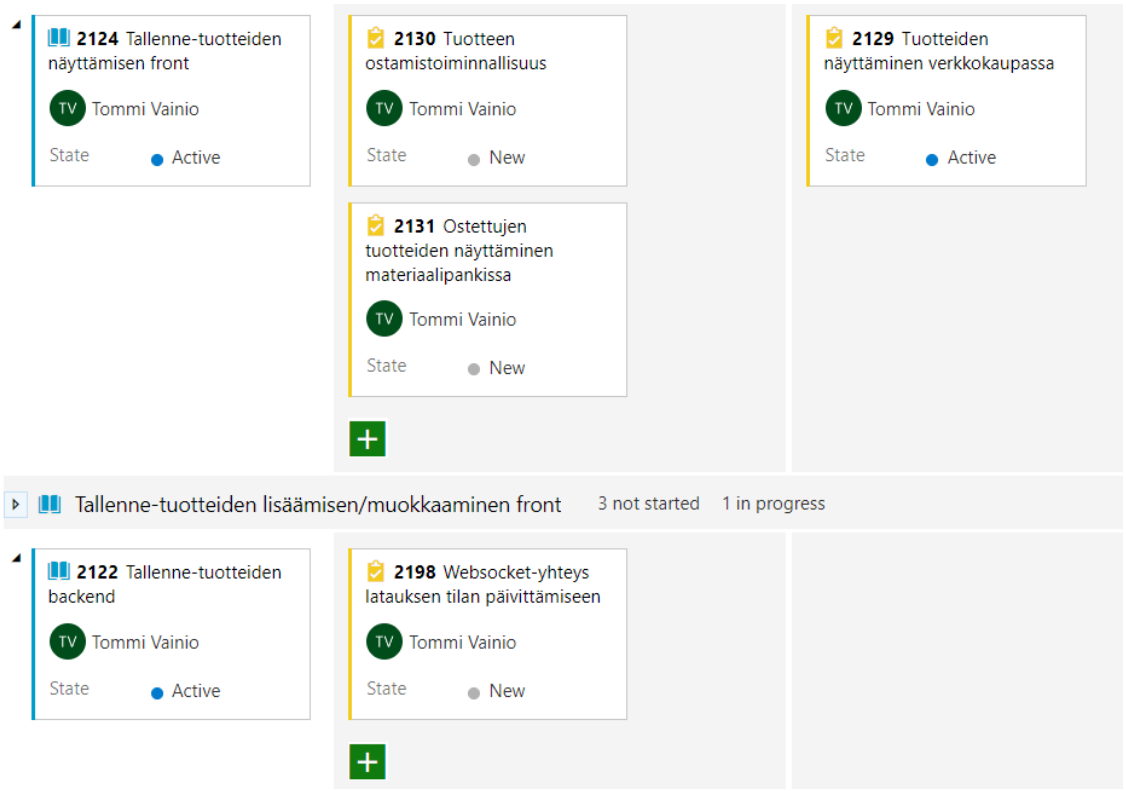
Prototyyppiä vaivannut koulutusten latautumisen hitaus päätettiin korjata säilyttämällä Vimeossa ainoastaan videotiedostot. Koulutustuotteiden esikatselukuvat päätettiin tallentaa Azuren Blob Storage -datapankkiin ja koulutuksen kesto sekä tagit yrityksen omaan tietokantaan. Näin kaikkia koulutuksia listattaessa ei tietoja tarvinnut hakea ollenkaan Vimeosta, jolloin koulutusten latautumisesta tuli huomattavasti nopeampaa. Palvelun ulkoasu oli määrä tehdä yrityksen käytössä olevan ulkoisen käyttöliittymäsuunnittelijan Figma-sovellukseen tekemien mallien pohjalta. Kuvassa 11 on palvelun verkkokauppaosion näkymien mallit sekä tietokoneen ruudulle että mobiililaitteille. Osa malleista oli tehty jo aiemmin, jolloin palvelun määritelmät eivät olleet vielä muotoutuneet nykyiseen muotoonsa. Tämän vuoksi näissä malleissa oli joitain puutteita, ja siksi mallien ulkonäköä ei voinut suoraan kopioida. Esimerkiksi kuvan 11 mallien yksittäisen koulutuksen näkymästä puuttuu teaser-video, joka määritelmän mukaan kuuluu kyseiseen näkymään.



Kuva 11. Figma-sovellukseen luodut striimauspalvelun verkkokauppaosion näkymien mallit.

Pharmacin ekstranetprojektissa noudatetaan ketterää kehitysmallia, joten projektia edistetään kahden viikon pituisissa sprinteissä. Jokaista projektiin tehtävää kokonaisuutta varten luodaan Azuren Devops-palveluun käyttäjätarinat ja niiden alle tehtävät eli taskit, jotka kuvaavat pienempiä suoritettavia kokonaisuuksia. Kun tiettyä kokonaisuutta aletaan tekemään, siihen liittyvät käyttäjätarina ja task laitetaan active-tilaan. Kun kokonaisuus on saatu valmiiksi, laitetaan siihen liittyvä task test-tilaan, jolloin joku ekstranetprojektiryhmään kuuluvista henkilöistä käy testaamassa kyseisen kokonaisuuden. Jos kokonaisuus todetaan määrittelyjä vastaavaksi, siihen liittyvä task laitetaan closed-tilaan, jolloin se poistuu sprintin vaihtuessa seuraavaan. Myös tälle työlle luotiin Devopsiin Extranet-taululle käyttäjätarinat ja taskit. Kuvassa 12 näkyy osa työlle luoduista käyttäjätarinoista ja taskeista. Käyttäjätarinoissa on vasemmassa reunassa sininen viiva, kun taas taskeissa viiva on keltainen. Työ

jaettiin kolmeen käyttäjätarinaan, joista yksi edusti työn palvelinpuolta kokonaisuudessaan, toinen selainpuolen käyttäjälle näkyviä näkymiä ja kolmas selainpuolen tuotteiden hallintanäkymiä. Kuvanottohetkellä projektin palvelinpuolen tämän työn puitteissa toteutetut ominaisuudet oli jo tehty, testattu ja suljettu, joten niihin liittyvät taskit olivat jo poistuneet taululta. Tuolloin kehityksessä olivat koulutustuotteiden verkkokauppanäkymät, sillä niitä kuvaava task on kuvassa active-tilassa.



Kuva 12. Striimausplaveluun liittyviä käyttäjätarinoita ja taskeja ekstranetprojektin taululla Azure Devopsissa.

5.2 Tekninen toteutus

Prototyypin tapaan striimauspalvelun selainpuoli tehtiin TypeScriptin ja Reactin avulla. Palvelun ulkoasuun taas käytettiin Pharmacin omaa Pharmac-UI-tyylikirjastoa. Palvelinpuolen kielenä toimi myös TypeScript ja ajoympäristönä Node.js. Prototyypistä poiketen palvelinpuolella ei käytetty pelkkää Express.js-

sovelluskehystä, vaan tämän tehtävän hoiti NestJS, josta kuitenkin löytyvät Express.js:n ominaisuudet sisäänrakennettuna.

5.2.1 Palvelinpuoli

Striimauspalvelun kehitys aloitettiin palvelinpuolen toiminnallisuuksien ja tietokantataulujen luomisella. Palvelinpuolen ja tietokannan väliseen yhteydenpitoon käytettiin TypeORM-kirjastoa. Yksinkertaistettuna TypeORM muuttaa objektiorientoituneiden ohjelmointikielien, kuten TypeScriptin, koodin relaatiotietokannoille tarkoitetuiksi SQL-kyselyiksi. Koodiin luodaan entiteettejä eli luokkia, jotka mallintavat tietokantataulua. Näiden entiteettien perusteella TypeORM luo automaattisesti tietokantaan taulut ja niiden välitaulut. Kuvassa 13 on tuotteen tageja varten tehty entiteetti, joka sisältää kentät id:lle, suomen-, ruotsin- ja englanninkieliselle nimelle sekä tagin tyypille. Tämän entiteetin instansseja voidaan liittää id-kentän arvon avulla koulutustuote-entiteettiin. Koulutustuote-entiteettiin määriteltiin kenttä, joka sisältää listan kyseiseen koulutukseen liitettyjä tageja.

```
@Entity({ name: 'Tag' })
export class TagEntity {
  @ApiProperty({
    readOnly: true,
  })
  @PrimaryGeneratedColumn()
  id: number;

  @ApiProperty()
  @Column({ name: 'nameFi', type: 'nvarchar', nullable: false })
  nameFi: string;

  @ApiProperty({
    nullable: true,
    required: false,
  })
  @Column({ name: 'nameSwe', type: 'nvarchar', nullable: true })
  nameSwe: string;

  @ApiProperty({
    nullable: true,
    required: false,
  })
  @Column({ name: 'nameEn', type: 'nvarchar', nullable: true })
  nameEn: string;

  @ApiProperty()
  @Column({ name: 'type', type: 'nvarchar', nullable: false })
  type: string;

  constructor(partial: Partial<TagEntity>) {
    Object.assign(this, partial);
  }
}
```

Kuva 13. TypeORM-entiteetti, joka luo tietokantaan tageille oman taulun.

Palvelun tietokantatoiminnallisuuksien luomisen jälkeen tehtiin palvelun palvelinpuolen ja Vimeon avoimen rajapinnan välistä vuorovaikutusta varten oma service-luokka. Tämä luokka sisältää metodit videon lataukseen Vimeoon sekä sen poistamiseen ja muokkaamiseen. Prototyypissä videotiedosto ladattiin Vimeoon POST-kutsulla form-datana. Videon latauksen Vimeoon voi suorittaa myös tus-protokollaa käyttäen. Kyseessä on HTTP:n päälle rakennettu

protokolla tiedostojen latausta varten. Sen avulla tiedostoa ladatessa latauksen keskeydyttyä sitä ei tarvitse aloittaa alusta, vaan lataus jatkuu siitä, mihin se jäi keskeytyksen sattuessa. Tus-protokollalla tiedostot on mahdollista ladata myös osissa, jolloin latauksen etenemistä voidaan seurata. Prototyyppiä kehittäessä päädyttiin yksinkertaisempaan lataustapaan, mutta tuolloin päätettiin, että lopullisessa versiossa käytettäisiin tus-protokollaa, jotta videon latauksen eteneminen voidaan näyttää käyttäjälle. Näin ollen, tässä työssä videon lataus tehtiin käyttäen tus-js-client-moduulia, jonka avulla video saadaan ladattua Vimeoon käyttäen tus-protokollaa ja latauksen edistymistä voidaan seurata helposti.

Kuvassa 14 on videon Vimeoon lataukseen tarvittava koodi. Ensin videolle luodaan Vimeoon instanssi, joka palauttaa muun muassa linkin videon lataussijaintiin. Tämän jälkeen videotiedostoa aletaan ladata Vimeoon tus-js-client-moduulin tarjoaman luokan avulla. Jos videotiedosto on kooltaan yli 128 megatavua, se ladataan osissa, jolloin latauksen etenemistä voidaan seurata onProgress-funktion avulla. Tätä funktiota kutsutaan aina, kun uutta osaa videosta aletaan lataamaan. Palvelun selainpuolen ja palvelinpuolen välille on tarkoitus rakentaa myöhemmin websocket-yhteys, jolloin palvelin voi päivittää selaimelle reaaliajassa videon latauksen edistymistä. Tällöin kuvassa 14 näkyvät onError-, onProgress- ja onSuccess-funktiot lähettävät selainpuolelle tiedon latauksen tilasta, kun näitä funktioita kutsutaan. Insinöörityön tiukan aikataulun takia websocket-yhteys päätettiin kuitenkin jättää tekemättä tämän työn puitteissa ja toteuttaa se myöhemmin, kun on saatu ensin perustoiminnallisuudet tehtyä. Tässä insinöörityössä tehty videon lataustoiminnallisuus toimii paremmin kuin prototyypin vastaava toiminnallisuus, sillä suurikokoisiakin videoita pystyy lataamaan useita kerralla ilman, että lataus epäonnistuu. Prototyypissä suurikokoisten videon ja teaser-videon lataukset peräkkäin epäonnistuivat huomattavan usein. Tämän insinöörityön ratkaisussa, tulevan websocket-yhteyden ansiosta, ohjelman ei tarvitse jäädä odottamaan videon latausta, vaan se voi jatkaa suoritustaan heti videon latauksen alettua. Näin ollen varsinaisen videon ja teaser-videon lataus tapahtuu samanaikaisesti, mikä nopeuttaa kokonaisprosessia.

```
const create = await axios({
  url: `${VIMEO_API_URL}/me/videos`,
  method: 'POST',
  headers: {
    Authorization: `bearer ${VIMEO_API_ACCESS_TOKEN}`,
    'Content-Type': 'application/json',
    Accept: 'application/vnd.vimeo.*+json;version=3.4',
  },
  data,
});

if (!create || !create.data || create.status !== 200) {
  throw new HttpException(`Error creating ${type}`, 400);
}

const upload = new tus.Upload(video.buffer, {
  uploadUrl: create.data.upload.upload_link,
  chunkSize: video.size / 10 > 134217728 ? video.size / 10 : 134217728,
  onError: (error) => {
    console.log('Failed because: ' + error);
    throw new HttpException('Video Upload failed', 500);
  },
  onProgress: (bytesUploaded, bytesTotal) => {
    const percentage = ((bytesUploaded / bytesTotal) * 100).toFixed(2);
    console.log(bytesUploaded, bytesTotal, percentage + '%' + ' ' + type);
  },
  onSuccess: () => {
    console.log(`${type} uploaded successfully`);
  },
});

upload.start();
```

Kuva 14. Videon lataukseen Vimeoon tarvittava koodi sovelluksen palvelinpuolella.

Kaiken kaikkiaan tätä projektia varten ekstranetin palvelinpuolelle tehtiin päätepisteet (endpoint) seuraaville toimenpiteille:

- kaikkien koulutustuotteiden haku
- yksittäisen koulutustuotteen haku
- uuden koulutustuotteen luominen
- koulutustuotteen muokkaaminen
- koulutustuotteen poistaminen

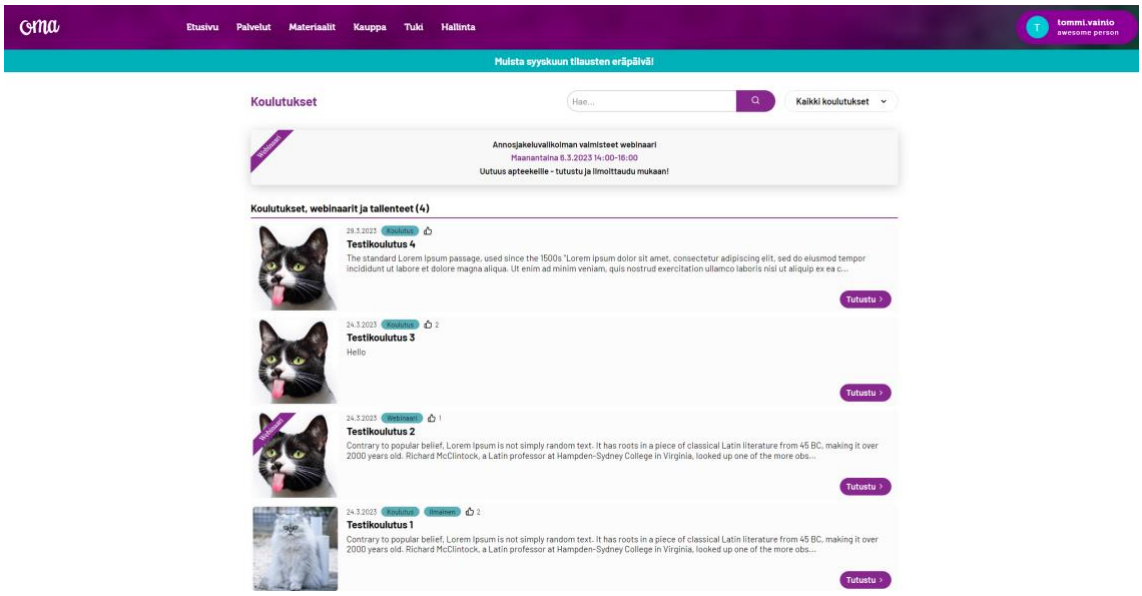
- tilauksen luominen
- tykkäyksen luominen
- tykkäyksen poistaminen
- uuden tagin luominen
- tagin poistaminen
- lisätyn kuvan poistaminen koulutuksesta
- teaser-videon poistaminen.

Koulutusten hakutoiminnallisuus tehtiin prototyyppiin nähden tiukemmaksi, sillä selainpuolelle lähetetään nyt vain ne koulutukset, jotka käyttäjällä on oikeus nähdä. Myös yksittäisen koulutuksen haun kohdalla tarkistetaan, onko käyttäjällä voimassa oleva tilaus, ennen kuin koulutusvideo liitetään koulutukseen. Prototyypissä selainpuolelle lähetettiin kaikki koulutukset ja niistä kaikki tiedot, mukaan lukien koulutusvideo, ja annettiin selainpuolen hoitaa tietojen suodattaminen. Tämä ei ollut hyvä ratkaisu, sillä selainpuolelle lähetettyihin tietoihin on kenellä tahansa mahdollisuus päästä käsiksi. Tämän insinöörityön tuloksena syntyneessä palvelussa riski tietojen päätymisestä vääriin käsiin on siis huomattavasti pienempi.

5.2.2 Selainpuoli

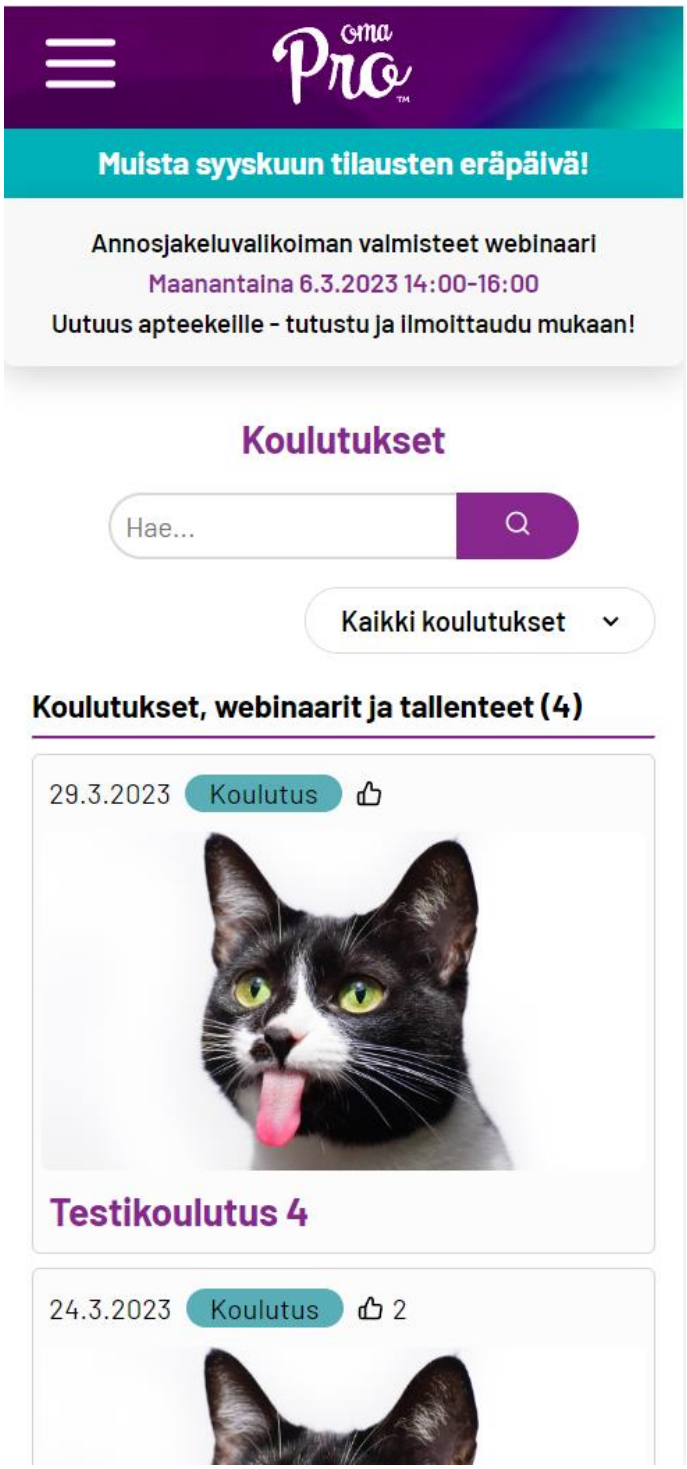
Striimauspalvelun palvelinpuolen toiminnallisuuksien valmistuttua aloitettiin ekstranetin selainpuolelle kehittämään kuvan 11 Figma-mallien mukaista verkkokauppaosiota koulutustuotteille. Työ jakautui tämän osion kohdalla kaikkien käyttäjälle saatavilla olevien koulutustuotteiden listanäkymään, yksittäisen koulutustuotteen näkymään ja näiden molempien mobiililaitteille tarkoitettuihin näkymiin. Kuvassa 16 on valmis tietokoneen näytölle tarkoitettu koulutustuotteiden listanäkymä. Koska koulutusten haku tehtiin tässä projektissa palvelinpuolella eri tavalla kuin prototyypissä, ne latautuvat tässä näkymässä erittäin nopeasti. Koulutuksia voi suodattaa hakusanan sekä kategorioiden ja tagien avulla. Suodatus onnistuu myös yhtäaikaisesti hakusanan ja alasvetovalikosta valitun kategorian tai tagin avulla. Tuotteille voi myös lisätä tykkäyksen tai poistaa oman tykkäyksen painamalla tuotekortissa

olevaa peukalokuvaketta. Kolmessa ensimmäisessä tuotteessa oleva kieltä näyttävän kissan kuva määritettiin väliaikaisesti tuotteen vakiokuvaksi, eli se näytetään kaikissa sellaisissa tuotteissa, joille ei ole erikseen määritelty kuvaa. Kaikki palvelun kuvat, mukaan lukien vakiokuvat, tallennetaan palvelinpuolen kautta Azure Blob Storage -palvelussa olevaan ekstranetin kuville tarkoitettuun säilöön.



Kuva 15. Koulutustuotteiden listanäkymä tietokoneen näytöllä.

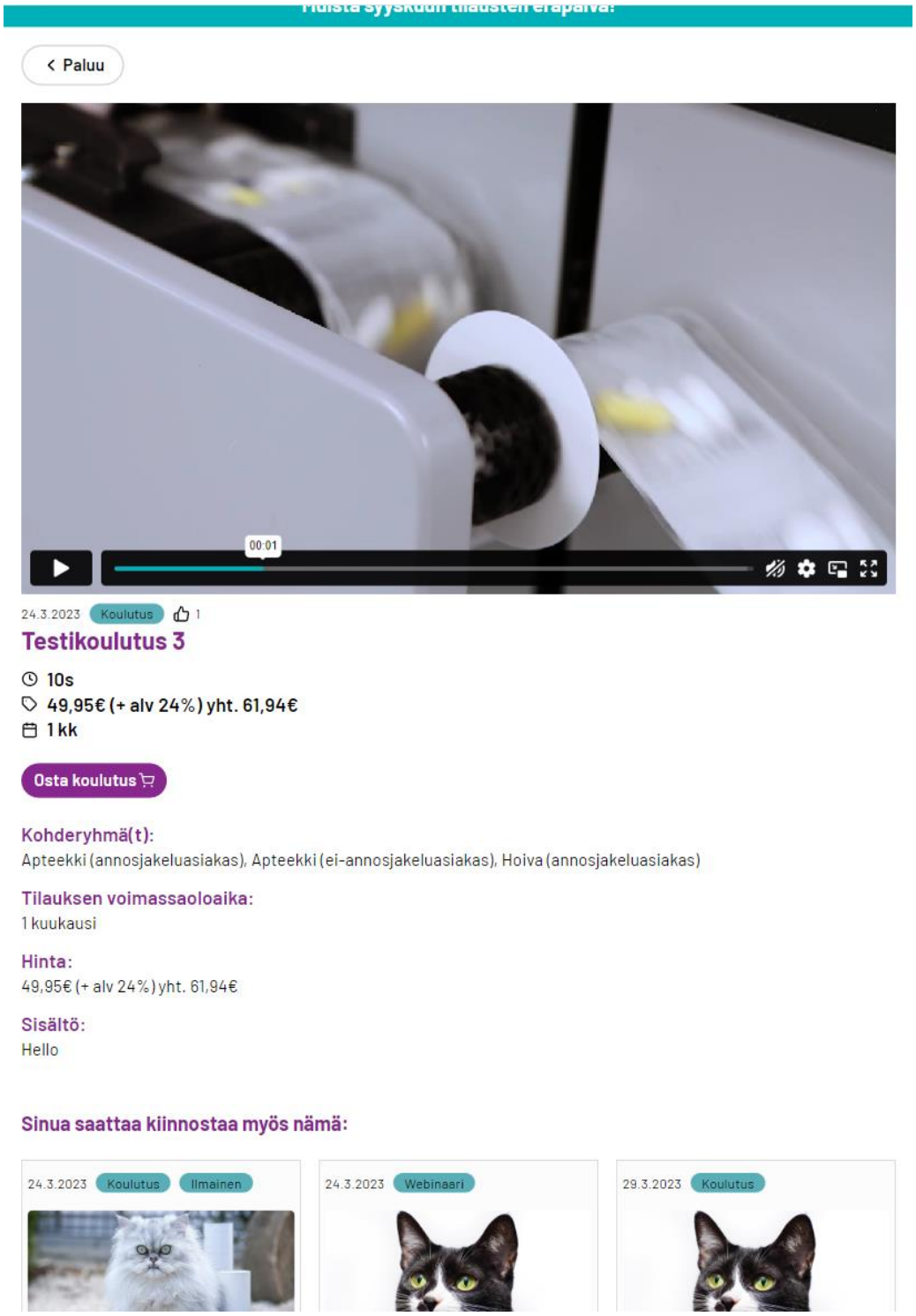
Kuvassa 16 on kuvan 15 näkymä älypuhelimen näytöllä katsottuna. Alkuperäisten tietokantamäärittelyjen mukaan tuotteella piti olla yksi kenttä kuvalle. Kun tuotteiden verkkokauppanäkymien kehitys aloitettiin, huomattiin kuitenkin, että Figmaan tehtyjen mallien mukaan tietokoneen ruudulla ja mobiililaitteella näkyvillä kuvilla on eri kuvasuhteet. Tämän vuoksi tietokantaan piti käydä jälkikäteen lisäämässä kenttä mobiililaitteille tarkoitetulle kuvalle. Raja, jossa tietokoneen näytöllä tarkoitettu näkymä muuttuu mobiililaitteelle tarkoitetuksi näkymäksi, on asetettu 800 pikselin näytön leveyden kohdalle. Näin ollen leveällä näytöllä varustetut mobiililaitteet, kuten osa tableteista, käyttää tietokoneen näytölle suunnattua näkymää. Näkymä on rakennettu kuitenkin hyvin skaalautuvista elementeistä, joten se näyttää hyvältä vielä 800 pikselin levyisellä näytölläkin.



Kuva 16. Koulutustuotteiden listanäkymän yläosa älypuhelimen näytöllä.

Kuvassa 17 näkyy suurin osa yksittäisen koulutustuotteen näkymästä tabletin näytöllä. Tähän näkymään päästään, kun klikataan kyseistä koulutustuotetta verkkokauppanäkymän koulutustuotelistassa. Kuvan koulutustuotteelle on

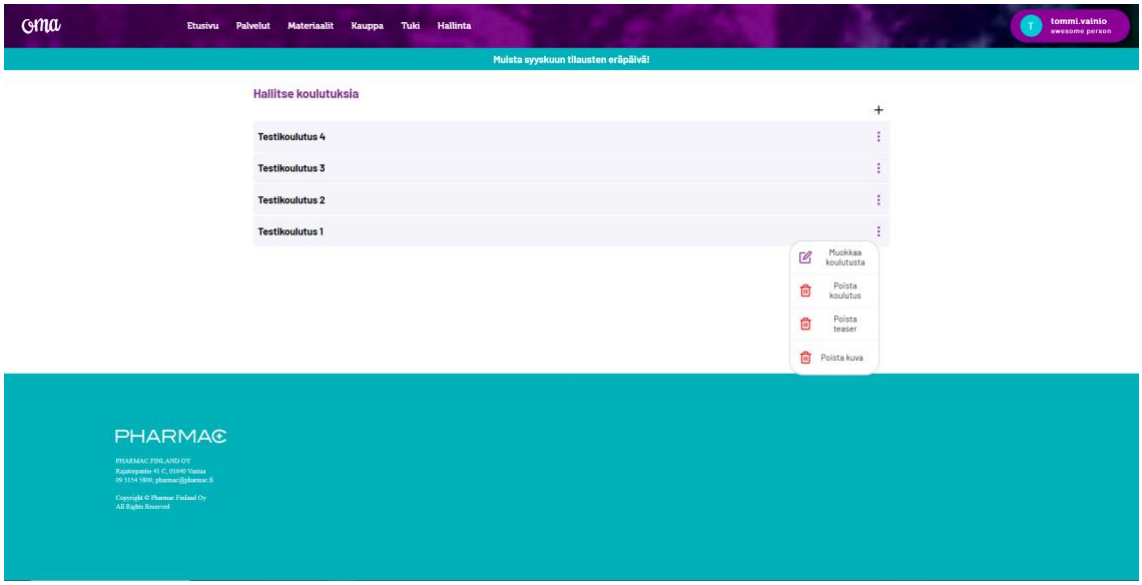
lisätty luomisvaiheessa teaser-video, joka toimii koulutuksen mainoksena. Videon luomisen yhteydessä on määritelty, että videosoitin on mahdollisimman yksinkertainen. Siinä ei esimerkiksi ole Vimeon logoa eikä mahdollisuutta jakaa tai ladata videota. Myös soittimen nappeihin ja aikajanaan on määritelty yrityksen käyttämä väri. Nämä samat määritelmät lisätään jokaiseen Vimeoon luotavaan videoinstanssiin, joten jokaisen videon soitin on samanlainen. Työn aloituksen yhteydessä määriteltiin, että ainakin aluksi jokaisen koulutuksen katseluoikeuden voi ostaa kuukaudeksi kerrallaan. Myöhemmin tätä ostettavan tilauksen kestoa voi mahdollisesti muuttaa. Kun ”Osta koulutus” -nappia painaa, kirjautuneelle käyttäjälle luodaan kuukauden kestävä tilaus kyseiseen koulutukseen. Jos käyttäjällä on voimassa oleva tilaus tähän koulutukseen, ”Osta koulutus” -nappi on harmaana eikä se reagoi klikkauksiin. Tuotteen maksamistoiminnallisuudet tulevat myöhemmin napin klikkauksen ja tilauksen luomisen väliin.



Kuva 17. Yksittäisen koulutustuotteen näkymä verkkokaupassa tabletin näytöllä.

Koulutustuotteiden hallintaosion tuotelistanäkymää varten ei ollut ehditty tehdä Figmaan malleja tämän näkymän toteuttamisvaiheessa, ja ohjeeksi tulikin

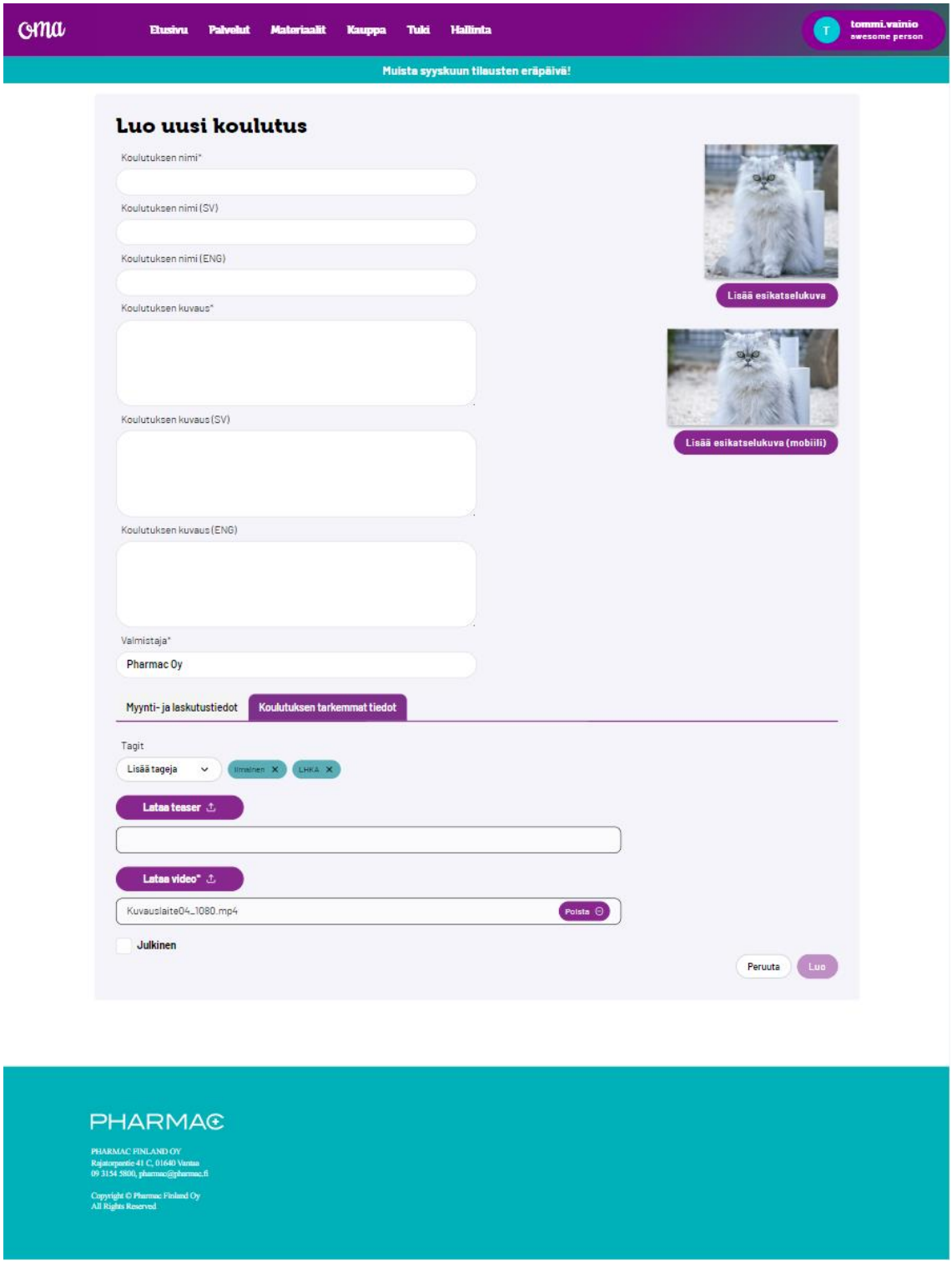
suunnitella itse ainakin toistaiseksi käytettävä yksinkertainen tyyli tähän näkymään. Kuvassa 18 näkyy suunnittelun lopputulos. Tähän näkymään riitti hyvin yksinkertainen koulutuslista, sillä näkymän pääsisivät näkemään vain Pharmacin omat työntekijät, joilla on sisällönhallitsijan oikeudet. Lisäksi tarkemmat tiedot koulutuksista löytyvät sekä verkkokauppaosiosta että materiaalipankkiosiosta, joten tähän näkymään niitä ei tarvita. Uuden koulutuksen luontinäkymään pääsee tästä näkymästä koulutuslistan yläpuolella oikeassa reunassa olevaa plus-kuvaketta painamalla. Olemassa olevan koulutuksen hallintaominaisuudet löytyvät koulutuksen kolmea pistettä painamalla avautuvasta alasvetovalikosta.



Kuva 18. Koulutustuotteiden listaus tuotteiden hallintaosiossa.

Kuvassa 19 näkyy osa koulutuksen luontinäkymästä. Koska tietokenttiä tarvittiin paljon, päätettiin niistä osa sijoittaa kahden välilehden taakse, jotta näkymä pysyisi suhteellisen kompaktina. Välilehtien toteutus päädyttiin lopulta tekemään alusta asti itse, vaikka Pharmac-UI-tyylikirjastosta olisi löytynyt siihen valmis komponentti. Syy tähän ratkaisuun oli se, että Pharmac-UI:n komponentti oli ulkonäöltään hieman erilainen kuin Figmaan tehdyssä mallissa ollut välilehtikomponentti ja sen ulkoasun muokkaaminen olisi vaatinut muutoksia itse tyylikirjaston puolelle. Tiukan aikataulun takia päädyttiin nopeampaan

ratkaisuun ja samalla saatiin näkymästä käyttöliittymäsuunnittelijan malliin nähden lähes identtinen.



Kuva 19. Koulutustuotteen luontinäkymä.

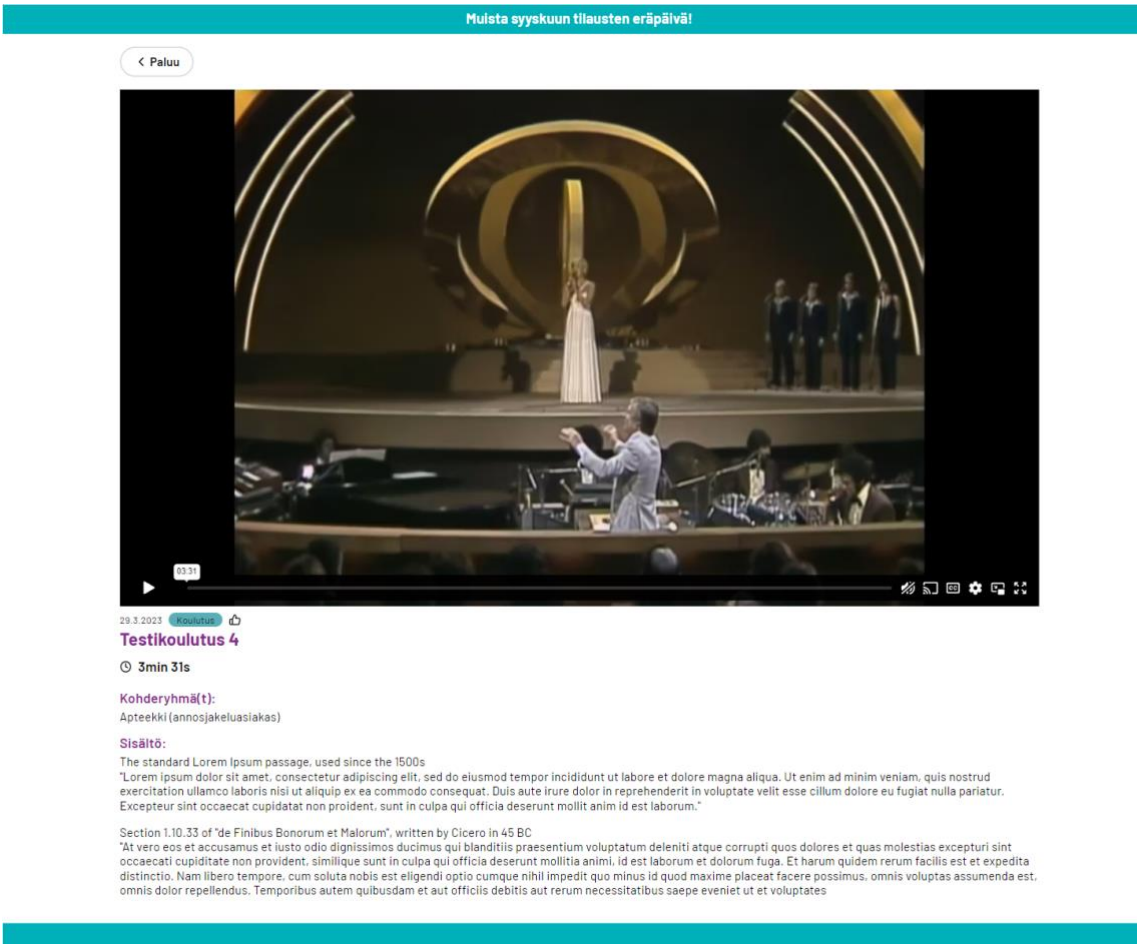
Kuvan 19 näkymä jaettiin useampaan komponenttiin, jotta koodi pysyisi helppolukuisena. Kuvassa 20 on välilehtien yläpuolella näkyvän komponentin koodi. Komponentti koostuu lähinnä @emotion/styled-kirjaston avulla tehdyistä tyylitellyistä React-komponenteista ja Pharmac-UI:n komponenteista. Tämä TrainingBasicInfoAndImages-niminen komponentti ottaa vastaan muun muassa listan, joka sisältää komponentissa olevien tekstikenttien tarvitsemat tiedot ja renderöi tämän listan perusteella kyseiset kentät map-funktiota käyttäen. Komponentti sisältää myös piilotetut tiedostojen lisäämiseen tarkoitetut input-elementit. Näiden elementtien tyyliä ei voi itsessään muokata, mutta tämä ongelma on kierretty laukaisemalla tällaisen elementin klikkaustapahtuma painamalla tätä varten luotua tyyliteltyä nappia ja piilottamalla itse input-elementti. Komponentissa näytetään myös käyttäjän input-elementin kautta valitsemat kuvat. Jos kuvaa ei ole valittu, sen tilalla näytetään kuvan haluttu kuvasuhde.

```
const TrainingBasicInfoAndImages = ({ inputData, trainingData, handleInputChange, handleFileChange, previewImage, t, hiddenImageInput, hiddenMobileImageInput, handleFileButtonClick } : Props) => {

    return (
        <BasicInfoAndImageContainer>
            <BasicInfoContainer>
                { inputData.map((a: {name: string, type: string, maxLength: number}, i: number) => (
                    <InputContainer key={ a.name }>
                        <PharmacText marginSmallBottom marginSmallLeft>{ t(`TRAININGS.MANAGE.OPTIONS.CREATE.${ a.name.toUpperCase() }`) }</PharmacText>
                        <PharmacInput value={ trainingData[a.name] } onChange={ handleInputChange } maxLength={ a.maxLength } name={ a.name } type={ a.type } />
                    </InputContainer>
                )) }
            </BasicInfoContainer>
            <ImagesContainer>
                <DefaultPreviewImageContainer type='desktop'>
                    { previewImage.image ? (
                        <PreviewImage type='desktop' src={ previewImage.image } />
                    ) : (
                        <PharmacText>1:1</PharmacText>
                    ) }
                </DefaultPreviewImageContainer>
                <PharmacButton onClick={ () => handleFileButtonClick('image') }>{ t('TRAININGS.MANAGE.OPTIONS.CREATE.ADD_IMAGE') }</PharmacButton>
                <HiddenFileInput
                    name="image"
                    type="file"
                    ref={ hiddenImageInput }
                    onChange={ handleFileChange }
                    accept="image/*"
                />
                <DefaultPreviewImageContainer type='mobile'>
                    { previewImage.mobileImage ? (
                        <PreviewImage type='mobile' src={ previewImage.mobileImage } />
                    ) : (
                        <PharmacText>16:9</PharmacText>
                    ) }
                </DefaultPreviewImageContainer>
                <PharmacButton onClick={ () => handleFileButtonClick('mobileImage') }>{ t('TRAININGS.MANAGE.OPTIONS.CREATE.ADD_MOBILE_IMAGE') }</PharmacButton>
                <HiddenFileInput
                    name="mobileImage"
                    type="file"
                    ref={ hiddenMobileImageInput }
                    onChange={ handleFileChange }
                    accept="image/*"
                />
            </ImagesContainer>
        </BasicInfoAndImageContainer>
    );
};

export default TrainingBasicInfoAndImages;
```

Kuva 20. TrainingBasicInfoAndImages-nimisen React-komponentin koodi.

Viimeisenä tämän työn puitteissa tehtiin materiaalipankkiosio koulutusten osalta. Se on ulkoasultaan ja toiminnallisuuksiltaan hyvin samankaltainen

sovelluksen verkkokauppaosion kanssa. Merkittävin ero on se, että materiaalipankkiosiossa listataan vain ne koulutukset, joihin kirjautuneella käyttäjällä on voimassa oleva tilaus, sekä ilmaiskoulutukset. Kuvassa 21 on materiaalipankkiosion yksittäisen koulutuksen näkymä, joka eroaa verkkokauppaosion vastaavasta näkymästä vähäisemmän tietomäärän ja näytettävän videon tyypin osalta. Kun verkkokauppaosiossa näytetään varsinaista koulutustallennetta mainostava teaser-video, tässä näytetään itse koulutustallenne. Osioiden samankaltaisuuden ansiosta materiaalipankin näkymien kehitys eteni nopeasti, kun voitiin käyttää paljon verkkokauppaosion komponentteja ja funktioita.



Kuva 21. Materiaalipankkiosion yksittäisen koulutuksen näkymä.

5.3 Lopputulokset ja jatkokehitys

Insinöörityön lopputuloksena saatiin sulavasti toimiva, eri kielivaihtoehtoja tukeva ja täysin responsiivinen tilausvideopalvelu. Palvelu integroitiin osaksi ekstranet-sovellusta, ja sen toiminnallisuudet jakautuvat ekstranetissä verkkokauppa-, materiaalipankki- ja hallintaosioihin. Verkkokauppaosiossa ekstranetin käyttäjä voi selailla hänelle saatavilla olevia koulutustuotteita, suodattaa niitä hakusanojen, tagien ja kategorioiden avulla, katsella yksittäisen koulutustuotteen tarkempia tietoja ja siihen mahdollisesti liitetyn teaser-videon, lisätä tuotteelle tykkäyksen ja ostaa siihen katseluoikeuden kuukaudeksi. Ostaminen ei kuitenkaan vaadi tässä vaiheessa vielä tuotteen maksamista, vaan pelkkä napin painallus riittää. Ekstranetin materiaalipankkiosiossa käyttäjä voi selailla koulutuksia, joihin hänellä on voimassa oleva katseluoikeus, mukaan lukien ilmaiset koulutustuotteet, suodattaa näitä koulutuksia samoilla parametreillä kuin verkkokauppaosiossa, lisätä tykkäyksiä ja katsoa niihin liitetyn koulutustallenteen. Koulutustallenne on määritelty näkyväksi vain ekstranetin verkkotunnuksessa, joten käyttäjä ei pääse katsomaan tallennetta vain kopioimalla sen linkin. Ekstranetin hallintaosiossa sisällönhallitsijan oikeudet omaava Pharmacin työntekijä voi luoda uusia koulutustuotteita, sekä muokata ja poistaa olemassa olevia tuotteita. Koulutuksista voi myös poistaa niihin mahdollisesti lisätyn kuvan ja teaser-videon. Jos koulutustuotteeseen liitetyn kuvan poistaa, tuote käyttää kaikille koulutustuotteille määriteltyä oletuskuvaa.

Suurimmiksi jatkokehityskohteiksi jäivät maksutoiminnallisuus, webinaarien katseluoikeuksien myyminen verkkokaupan kautta, niiden mahdollinen näyttäminen ekstranetissä ja koulutustallenteen latauksen edistymisen näyttäminen websocket-yhteyden avulla. Maksutoiminnallisuutta aletaan toteuttamaan, kunhan siihen vaaditut toiminnallisuudet on saatu tehtyä Pharmacin toiminnanohjausjärjestelmään muiden kehittäjien toimesta. Tulossa olevan webinaarin katseluoikeuden sisältävä tuote vaatii koulutustallenteeseen verrattuna osittain hieman erilaisia tietoja. kuten esimerkiksi webinaarin järjestämisajankohdan. Webinaarituotteiden kehitykseen ei tässä vaiheessa

kuitenkaan vielä panostettu, sillä webinaarien tarkka näyttämistapa on edelleen päättämättä. Palvelun prototyypin kehityksen alkaessa tavoitteena oli, että webinaarien järjestäminen ja näyttäminen tapahtuisi tämän VOD-palvelun kautta ekstranetissä. Prototyypin kehityksen yhteydessä selvisi kuitenkin, että Advanced-paketin tilauksella webinaareja ei voi hallita Vimeon rajapinnan kautta. Tuolloin suunniteltiin alustavasti, että webinaarien luomisen voisi tehdä Vimeon verkkosivujen kautta ja niiden näyttämisen taas tässä ekstranetin VOD-palvelussa. Tämän vaihtoehdon tarkempi tutkiminen päätettiin jättää myöhemmäksi, eikä sitä ehditty tutkia tämänkään työn puitteissa, joten se jää tehtäväksi myöhemmin.

Tämän projektin selainpuolen ja palvelinpuolen välille ei oltu alun perin suunniteltu websocket-yhteyttä, mutta videon lataustoiminnallisuutta kehitettäessä todettiin, että latauksen seuranta olisi järkevintä tehdä websocket-yhteyden avulla. Tämä päätettiin toteuttaa vasta viimeisenä, sillä sen toteuttaminen vaatii jonkin verran aikaa ja haluttiin varmistaa, että tärkeimmät ominaisuudet ehditään varmasti tehdä työn puitteissa. Lopulta työn toteutuksen aikataulu meni sen verran tiukaksi, että websocket-yhteyttä ei ehditty toteuttaa tämän työn puitteissa ja se jäi jatkokehityskohteeksi. Edellä mainittujen kohteiden lisäksi palvelun ulkoasuun, käytettävyyteen ja toiminnallisuuksiin liittyen saattaa tulla tulevaisuudessa uusia jatkokehityskohteita, kunhan yhä useampi henkilö pääsee testaamaan palvelua.

6 Yhteenveto

Insinöörityössä päästiin kaikkiin tavoitteisiin, jotka oli asetettu työn alkaessa. Pharmac sai ekstranetportaaliinsa annetut vaatimukset täyttävän video on demand -palvelun, jonka kautta se voi myydä katseluoikeuksia koulutuksiinsa ja webinaaritallenteisiinsa. Alun perin tavoitteena oli, että työn tuloksena olisi saatu täysin valmis palvelu, mutta jo ennen työn aloitusta oli selvää, että ulkoisten tekijöiden ja tiukan aikataulun vuoksi tämä ei olisi mahdollista. Näin ollen maksamiseen ja webinaareihin liittyvät toiminnallisuudet jätettiin jo tuolloin tavoitteista pois. Tällä ei kuitenkaan ole suurta merkitystä, sillä ekstranetin,

johon palvelu on integroitu, kehitys on edelleen kesken. Palvelua ei siis olisi päästy kokeilemaan oikeiden asiakkaiden kanssa tämän työn puitteissa, vaikka se olisikin saatu täysin valmiiksi.

Työn myötä opittiin paljon uusia asioita, joita ei prototyyppiä kehittäessä ollut tehty. Tällaisia olivat esimerkiksi videon lataaminen tus-protokollan avulla, oikeiden käyttäjien pääsynhallinta ja tietyn ajan voimassa olevien tilauksien toteuttaminen käyttäjädataa hyödyntäen. Myös moni jo prototyypissä ollut ominaisuus, kuten koulutusten haku ja käyttäjälle lähetetyn tiedon rajoittaminen, tehtiin tämän työn puitteissa paremmin. Insinöörityön lopputuloksesta tuli prototyyppiin verrattuna selkeästi parempi. Lopputuloksena syntynyttä palvelua testasivat yrityksessä ekstranetprojektin kuuluvat työntekijät ja palveluun oltiin tyytyväisiä. Työn verkkokauppaosiota ehdittiin esitellä myös laajemmalle joukolle yrityksen työntekijöitä, ja se sai heiltäkin osakseen positiivista palautetta.

Lähteet

1 Oluwademilade, Afolabi. 2022. What is Video on Demand (VOD) and How Does It Work? Verkkoaineisto. MUD. <https://www.makeuseof.com/what-is-video-on-demand-how-it-works>. Luettu 15.2.2023.

2 Pereira, Rita & Tam, Carlos. 2021. Impact of enjoyment on the usage continuance intention of video-on-demand services. Information & management 2021. Vol. 58.

3 Revenue of the video-on-demand market worldwide from 2017 to 2027, by segment. 2023. Verkkoaineisto. Statista. <https://www.statista.com/forecasts/460399/video-on-demand-revenue-in-the-world-forecast>. Luettu 16.2.2023.

4 Streaming claims largest piece of TV viewing pie in July. 2022. Verkkoaineisto. Nielsen. <https://www.nielsen.com/insights/2022/streaming-claims-largest-piece-of-tv-viewing-pie-in-july>. Luettu 22.2.2023.

5 Curry, David. 2023. Video Streaming App Revenue and Usage Statistics (2023). Verkkoaineisto. Business of Apps. <https://www.businessofapps.com/data/video-streaming-app-market>. Luettu 15.2.2023.

6 Kariuki, Patrick. 2023. How and When Did Netflix Start? A Brief History of the Company. Verkkoaineisto. MUD. <https://www.makeuseof.com/how-when-netflix-start-brief-company-history>. Luettu 15.2.2023.

7 Pattison, Sandra. 2022. 35 Streaming Services Statistics for 2023: Deep Dive Into Video & Music Streaming. Verkkoaineisto. Cloudwards. <https://www.cloudwards.net/streaming-services-statistics/#Sources>. Luettu 16.2.2023.

8 Number of Netflix paid subscribers worldwide from 1st quarter 2013 to 4th quarter 2022. 2023. Verkkoaineisto. Statista. <https://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide>. Luettu 16.2.2023.

9 Handler Miller, Carolyn & Kane, Frank. 2022. Streaming Video Strategies. O'Reilly Media.

10 DeJonghe, Derek. 2021. Optimize Video Streaming Delivery. O'Reilly Media.

11 Beshkov, Mukhaddin. 2023. What is Content Delivery Network? Verkkoaineisto. <https://www.wallarm.com/what/what-is-content-delivery-network>. Päivitetty 20.2.2023. Luettu 15.3.2023.

12 What is Vimeo? How does it work and what does it Do? Verkkoaineisto. Radaar. <https://www.radaar.io/resources-121/blog-388/what-is-vimeo-how-does-it-work-and-what-does-it-do-1185>. Luettu 28.2.2023.

13 Wise, Jason. 2022. How many people use Vimeo in 2023? (US & worldwide stats). Verkkoaineisto. Earthweb. <https://earthweb.com/vimeo-users>. Päivitetty 4.12.2022. Luettu 1.3.2023.

14 Choose your plan. Verkkoaineisto. Vimeo. <https://vimeo.com/upgrade-plan>. Luettu 1.3.2023.

15 Burel, Eric & Greif, Sacha. 2022. Front-end Frameworks. Verkkoaineisto. 2022 State of JS. <https://2022.stateofjs.com/en-US/libraries/front-end-frameworks>. Luettu 1.3.2023.

16 Goldberg, John. 2022. Learning TypeScript. E-kirja. O'Reilly Media.

17 Shubel, Meredith. 2022. What is TypeScript? Verkkoaineisto. The New Stack. <https://thenewstack.io/what-is-typescript>. Luettu 6.3.2023.

18 GitHut 2.0. Verkkoaineisto. <https://madnight.github.io/githut/#/pushes/2022/4>. Luettu 6.3.2023.

19 Narayn, Hari. 2022. Just React!: Learn React the React Way. E-kirja. Apress.

20 Deshpande, Chinmayee. 2023. The Best Guide to Know What is React. Verkkoaineisto. Simplilearn. <https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs>. Päivitetty 7.2.2023. Luettu 8.3.2023.

21 Semah, Benjamin. 2022. What Exactly is Node.js? Explained for Beginners. Verkkoaineisto. freeCodeCamp. <https://www.freecodecamp.org/news/what-is-node-js>. Luettu 9.3.2023.

22 Sufiyan, Taha. 2023. What is Node.js: A Comprehensive Guide. Verkkoaineisto. Simplilearn. <https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs>. Päivitetty 3.2.2023. Luettu 9.3.2023.

23 Sheldon, Robert & Denman, James. 2022. Node.js (Node). Verkkoaineisto. TechTarget. <https://www.techtarget.com/whatis/definition/Nodejs>. Luettu 9.3.2023.

24 Hooda, Parikshit. 2022. Node.js | NPM (Node Package Manager). Verkkoaineisto. GeeksforGeeks. <https://www.geeksforgeeks.org/node-js-npm-node-package-manager>. Päivitetty 8.3.2022. Luettu 12.3.2023.

25 Vailshery, Lionel Sujay. 2022. Most used web frameworks among developers worldwide, as of 2022. Verkkoaineisto. Statista. <https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web>. Luettu 12.3.2023.

26 Krstic, Branko. 2023. 64 Node JS Stats that Prove Its Awesomeness in 2023. Verkkoaineisto. Web tribunal. <https://webtribunal.net/blog/node-js-stats/#gref>. Luettu 12.3.2023.

27 McArthur, Ann. 2021. Why use Nest.js. Verkkoaineisto. DevCycle. <https://devcycle.com/blog/why-use-nest-js>. Luettu 14.3.2023.

28 Oyedele, Temitope. 2022. NestJs vs Express.js. Verkkoaineisto LogRocket. <https://blog.logrocket.com/nestjs-vs-express-js>. Luettu 14.3.2023.