Tampereen yliopisto

Tuomas Aho

TOIMENPIDEKORTIN DIGITAALINEN LEIMAUS LÄÄKETIETEEN OPETUKSESSA

Informaatioteknologian ja viestinnän tiedekunta
Diplomityö
Toukokuu 2019

TIIVISTELMÄ

Tuomas Aho: Toimenpidekortin digitaalinen leimaus lääketieteen opetuksessa
Diplomityö
Tampereen yliopisto
Diplomi-insinöörin tutkinto, Tietotekniikka, DI
Pääaine: Ohjelmistotuotanto
Tarkastajat: Professori Hannu-Matti Järvinen ja professori Niku Oksala
Toukokuu 2019

Tässä diplomityössä toteutettiin tietojärjestelmä, jolla digitalisoitiin lääketieteen opinnoissa käytetyt toimenpidekortit. Tähän saakka opiskelijat ovat keränneet suorituskuittaukset paperisille toimenpidekortille, joista opetuskoordinaattorit ovat siirtäneet tiedot opintosuoritusrekisteriin. Tämä on aiheuttanut runsaasti kaksoiskirjaamista. Lisäksi usein toimenpidekortit häviävät ja rikkoutuvat, sillä opetus tapahtuu useissa paikoissa ja useiden toimijoiden toimesta. Opiskelijoille on ollut vaikea antaa palautetta ja toisaalta opiskelijat ovat antaneet palautteen useita kuukausia opetuksen jälkeen, jolloin palaute on todennäköisimmin vääristynyt. Jotta opetusta voitaisiin kehittää, huolehtia oppimistapahtumien kertymisestä, valita opetusmenetelmiä ja tehdä pedagogista tutkimusta, tarvitaan rakenteinen alusta, jolla voidaan toteuttaa nämä tehtävät.

Työ tehtiin Tampereen yliopiston lääketieteen ja terveysteknologian tiedekunnalle. Toteutettu tietojärjestelmä on WWW-pohjainen MVP-toteutus (Minimum Viable Product) ja se osoittaa, että toimenpidekorttien digitalisointi on mahdollista. Projektin yhteydessä tehty järjestelmäpilotti osoittaa myös sen, että toteutetussa tietojärjestelmässä on jatkokehitettynä potentiaalia toimenpidekorttien korvaajaksi. Käyttöönotettuna järjestelmä muovaa lääketieteen opetuksen prosesseja läsnäolokirjausten osalta.

Tietojärjestelmässä käytetään älypuhelimen kameraa opintosuoritteiden leimaamisessa HTML5-rajapintojen kautta. Opettajat voivat luoda tietotojärjestelmän avulla suoraistuntoja, joihin opiskelijat voivat liittyä käyttämällä sovellukseen rakennettua QR-koodin lukijaa. Suoraistunnoissa käytetään tapauskohtaisia ajallisesti vaihtuvia QR-koodeja, joihin on rakennettu myös vilppiä estäviä mekanismeja. QR-koodit vaihtuvat ja uusiutuvat niin tiuhaan, että niitä on vaikea jakaa valokuvana käytettäväksi eteenpäin. Erikoistilanteita varten järjestelmään rakennettiin manuaalinen tapa lisätä opiskelijoita oppimistapahtumiin ja staattisia tulostettavia QR-koodeja, joilla voidaan antaa opintosuoritteita, vaikka paikalla ei ole tietojärjestelmää käyttävää opettajaa. Opettajat voivat edellä mainittujen menetelmien lisäksi lisätä opiskelijan suoraistuntoon lukemalla opiskelijan yksilöivää QR-koodia tai lukemalla opiskelijakortin takana olevan viivakoodin.

Järjestelmän kautta voidaan kerätä myös välitöntä kaksisuuntaista palautetta oppimistapahtumista, mikä auttaa opetuksen jatkuvassa kehittämisessä ja opetusmenetelmien valitsemisessa. Kerätty palaute mahdollistaa ensi kertaa eteenpäinsuuntautuvan pedagogisen tutkimuksen, jossa sitä voidaan käyttää tehokkaana tutkimustyökaluna. Kehitetyn järjestelmän avulla palaute saadaan välittömästi, jolloin se on luotettavaa ja sitä voidaan käyttää jatkuvaan laadunvalvontaan. Järjestelmä on rakennettu siten, että sitä voidaan käyttää myös esim. erikoislääkärikoulutuksen rakenteista arviointia arviointia ja myös muiden alojen kuten sairaanhoidon opinnoissa.

Avainsanat: SPA, Node.js, QR-koodi, REST, RESTful, GraphQL, React, WebSocket

ABSTRACT

Tuomas Aho: Digital confirmation system for medical student's learning tasks
Master's Thesis
Tampere University
Information technology, MSc
Major: Software Engineering
Examiners: Professor Hannu-Matti Järvinen and Professor Niku Oksala
May 2019

This Master's Thesis is based on information system implemented during a project where medical students attendance lists were digitalized. The project was established in Faculty of Medicine and Health Technology in Tampere University. Implemented information system is in a MVP phase (Minimum Viable Product) and it is Web based software which proves that attendance lists can be digitalized. The software was tested in a pilot project and pilot's results proves that paper lists can be replaced with the system after further development. The system will be changing processes in Medical Studies in case it is put into service.

This Thesis uses constructive research approach and it answers to manual paper work issues. Papers are often lost or destroyed or students don't remember to take those with them. Teaching can be arranged in multiple places and by many parties which aggravates the problem. Current attendance lists and other sheets causes a lot of paper work and double recording. Feedback is collected after long period of time which distorts results. Software based platform is needed to answers to these problems and collected data can be used to improve teaching and in education research purposes. The system developed answers to these needs and can be used as a reference system or base line for further development.

The system is an SPA based Web application which takes advantage of mobile phone's cameras via HTML5 APIs. Camera is used in attendance registration. Teachers can use the system to make live sessions and collect attendances with QR-codes that students can scan with a scanner implemented in the application. QR-code changes time periodically, so it can also prevent scams and fake attendances. Teachers can download PDF files for special occasions where system can not be used in online situation. Students can get attendance marks from reading QR-code from the file that can be printed. Teachers can also add students manually to evaluation sessions or read student's unique QR-codes or student card's bar codes with a scanner.

The system can be used to collect 2-way feedback from studying sessions where attendances are collected. For the first time, it can be used for prospective pedagogic studies aiming to develop teaching methodology. Feedback can be used to improve teaching and to improve personal expertise. Feedback can be collected instantly after studying sessions, collected feedback is accurate and it can be used in teaching quality management. In a old way 1-way feedback were collected after each study period. In the worst case scenario the feedback is collected after 3 months and because of that the feedback may not have been precise anymore.

Keywords: SPA, Node.js, QR-code, REST, RESTful, GraphQL, React, WebSocket

ALKUSANAT

Haluan kiittää työnantajaani Tampereen yliopistoa tämän diplomityöaiheen tarjoamisesta. Erityisesti haluan kiittää diplomityöni tarkastajia Hannu-Matti Järvistä ja Niku Oksalaa tuesta ja ohjauksesta, sekä aktiivisesta osallistumisesta projektiin. Haluan kiittää myös muita projektissa mukana olleita heidän osallistumisestaan ja panostuksestaan järjestelmän kehittämiseen.

Haluan kiittää vanhempiani ja ystäviäni, sekä Rölli-koiraa tuesta tämän työn aikana. Teillä on ollut suuri merkitys jaksamiseeni tämän työn aikana.

Tampereella, 2. toukokuuta 2019

Tuomas Aho

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

2FA	Kaksivaiheinen tunnistautuminen (engl. Two-factor authentication)
Apache HTTP Server	Avoimeen lähdekoodiin perustuva HTTP-välityspalvelinohjelma
API	Ohjelmistorajapinta (engl. Application programming interface)
CORS	Webin cross-origin on tietoturvaominaisuus eri originien välillä (engl. Cross-origin resource sharing)
CSS	Tyylikieli. WWW-dokumenteille kehitetty tyyliohjeiden laji. (engl. Cascading Style Sheets)
DOM	Dokumenttioliomalli (engl. Document Object Model)
GraphQL	Avoimen lähdekoodin API kyselykieli. Vaihtoehto RESTille
HAKA	Suomen korkeakoulujen luottamusvekosto. Työn yhteydessä tarkoitetaan Haka-käyttäjätunnistusjärjestelmää
HTML	Hypertekstin merkintäkieli (engl. Hypertext Markup Language)
HTTP	Protokolla, jota selaimet ja WWW-palvelimet käyttävät tiedonsiirtoon (engl. Hypertext Transfer Protocol)
I/O	Siirräntä. Työn yhteydessä tällä tarkoitetaan tiedonsiirtoa väylän kautta (engl. Input/Output)
idP	Identiteetinhallinjärjestelmä (engl. An identity provider)
JavaScript	Ohjelmointikieli
JWT	JSON-pohjainen avoimen standardin (RCF 7519) menetelmä käyttöoikeustietueiden (Access Token) hallinnoimiseen eri ohjelmistojen välillä. (engl. JSON Web Tokens)
LTS	Tuotteelle annettu pitkäaikaisen tuen lupaus (engl. Long-Term Support)
MPA	Serverin puolella templateilla ja sivupohjamoottoreilla generoitava monisivuinen sovellus (engl. Multiple-Page Application)
MVP	Pienin toimiva tuote (engl. Minimum Viable Product)
NFC	RFID tekniikkaa höydyntävä lähiviestintapa (engl. Near Field Communication)
Node.js	Avoimen lähdekoodin alustariippumaton JavaScript runtime-ympäristö JavaScript-koodin suorittamiseen palvelimella
ORM	Olio-relaatio mallinnus (engl. Object-relational mapping)

PostgreSQL	Relaatiotietokantajärjestelmä
Proxy	Välityspalvelin
PWA	Verkkoselainpohjainen sovellus, jossa on natiivisovelluksen tyyppisiä toiminnallisuuksia kuten Offline-ominaisuuksia, Push notificationeita ja laitteiston hyödyntämiseen liittyviä ominaisuuksia (engl. Progressive Web App)
QR-koodi	Ruutukoodi eli kaksiulotteinen kuviokoodi, johon on koodattu informaatiot (engl. Quick Response Code)
ReactJS	Suosittu JavaScript-kirjasto SPA-sovelluksien toteuttamisessa
REST	HTTP-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamisee (engl. Representational State Transfer)
RFID	Radiotaajuinen etätunnistus (engl. radio frequency identification)
Shibboleth	Kertakirjautumispalvelu tietoverkoille ja internetiin
SP	Palveluntarjoaja (engl. service provider)
SPA	Yhden sivun sovellus (engl. Single-Page Application)
SQL	IBM:n kehittämä standardoitu kyselykieli (engl. Structured Query Language)
SSR	SPA-sovellusnäkymien renderöinti palvelimella verkkoselaimen sijasta (engl. Server-Side Rendering)
TCP	Tietoliikenneprotokolla, jolla luodaan yhteyksiä tietokoneiden välille (engl. Transmission Control Protocol)
TUNI	Tampereen korkeakouluyhteisö (engl. Tampere Universities)
URL	verkkosivun osoite (engl. Uniform Resource Locator)
Web tai WWW	Internet-verkossa toimiva hajautettu hypertekstijärjestelmä (engl. World Wide Web)
WS	WebSocket

1 JOHDANTO

Syksyllä 2018 käynnistettiin Tampereen yliopistossa hanke, jossa pyrittiin digitalisoimaan lääketieteen opetuksessa käytettävät toimenpidekortit. Tämä diplomityö on toteutettu kyseisen hankkeen yhteydessä, missä toteutettiin MVP-toteutus toimenpidekorttien digitalisoimisesta. Tässä opinnäytetyössä käytetään konstruktiivista tutkimusasetelmaa. Työssä suunniteltiin toteutus käytännön ongelman ratkaisemiseksi. Yhtenä ongelmana työssä on toimenpidekorttien digitalisointi eli paperityön vähentäminen käyttökohteen ollessa lääketieteen opinnoissa olevissa suoritteiden kirjaamisessa. Ongelmina paperityön lisäksi olivat palautteen kerääminen useita kuukausia oppimistapahtuman jälkeen, jolloin saatu palaute vääristyy ja palautetta ei aiemmin ole saatu kaksisuuntaisesti. Lisäksi paperit häviävät ja hajoavat usein tai niitä ei muisteta ottaa mukaan. Ratkaisuna näihin ongelmiin on verkkoselainpohjainen tietojärjestelmä. Projektin aikana toteutettiin palautekysely järjestelmäpilotin yhteydessä ja sitä ja sen tuloksia käsitellään luvussa 4.

Toimenpidekortit ovat fyysisiä A4-kokoisia papereita, joihin opiskelijat keräävät suoritemerkintöjä suoritettuaan lääketieteen opintoihin liittyviä tehtäviä. Toimenpidekortit ovat eräänlaisia läsnäololistoja, jotka ovat korvamerkattuja juuri tietyille opintojaksoille ja näiden opintojaksojen tehtäville. Näitä tehtäviä on mm. lääketieteellisten toimenpiteiden seuraaminen ja suorittaminen, ryhmätöiden tekeminen tai läsnäolo pakollisilla luennoilla. Toimenpidekorteille ja erilaisille läsnäololistoille kertyviä merkintöjä tulee paljon yksittäistä opiskelijaa kohti opiskelijan opintojen aikana ja ne kertaantuvat opiskelijoiden lukumäärään mukaan. Opiskelijat palauttavat toimenpidekortit saatuaan siihen kaikki tarvittavat merkinnät, tyypillisesti opintojaksojen lopussa ja opetuskoordinaattorit suorittavat manuaalisesti korttien tarkistamisen ja opintosuoritusten kirjaamisen opintorekisteriin.

Opinnäytetyön yhteydessä toteutettu tietojärjestelmä on vahvasti kehitetty juuri lääketieteen opintojen opintojaksorakenteisiin sopivaksi. Lääketieteen opinnoissa on poikkeuksellinen ennalta määrättyihin lukujärjestyksiin pohjautuva opintosuunnitelma. Opinnot ovat jaettu vuosikursseihin ja tiettyihin vuosikursseihin kuuluvat tietyt opintojaksot. Opiskelija suorittaa ennalta määrätyt perusopinnot, eikä opiskelijalla ole juurikaan mahdollisuuksia vaikuttaa opintojen sisältöön. Esimerkiksi omiin tekniikan opintoihin verrattuna lääketieteen opinnoissa on vähän valinnanvapautta. Tekniikan opinnot ovat opintojaksopohjaisia ja niissä tehdään paljon opiskelijan vastuulla olevia opintojakso-ilmoittautumisia. Lääketieteen opinnoissa ei tehdä juurikaan vastaavanlaisia ilmoittautumisia, vaan opiskelija kuuluu tiettyihin opintojaksoihin automaattisesti ollessaan tietyn vuosikurssin opiskelija. Opiskelijat suorittavat opintoihin kuuluvia toimenpiteitä tai ryhmätyötä tietyissä pienryh-

missä, jotka ovat kiinteitä lähes koko opiskelijan opintojen ajan.

Toimenpidekorttien aiheuttaman manuaalisen työn lisäksi tietojärjestelmä vastaa paperisten dokumenttien perusongelmaan – paperit hukkuvat ja tuhoutuvat helposti tai niitä ei muisteta ottaa mukaan. Lisäksi paperidokumentteja on helppo väärentää ja juuri väärentämisen estämiseksi toteutettuun järjestelmään on kehitetty uniikkeja menetelmiä. Työssä toteutetut menetelmät toimivat vilpin ehkäisyssä ja lisäksi järjestelmä tarjoaa sähköisiä menetelmiä läsnäolokirjausten tekemiseen. Toteutettu järjestelmä on itsenäinen toteutus, eikä se riipu mistään muusta tietojärjestelmästä kuin TUNI-organisaation kirjautumispalvelusta. Työn tulosten yhteenveto löytyy luvusta 5.

2 TEOREETTINEN TAUSTA

Tässä luvussa avataan työn taustalla olevaa teoriaa ja luvun tarkoitus on auttaa lukijaa ymmärtämään työn lopputuotteen eli verkkosovelluksen arkkitehtuurillisia ratkaisuja. Luvussa esitetyllä teorialla on vahva sidos työn käytännön osuuteen ja sisältö auttaa ymmärtämään työn käytännön osuuden arkkitehtuurillisia ja teknologisia ratkaisuja.

2.1 Web-arkkitehtuureista

Verkkoselainpohjainen sovellus sisältää monia teknisiä ja arkkitehtonisia ratkaisuja. Verkkoselaimella käytettävä sovellus jakautuu käyttöliittymään, palvelinrajapintaan ja niiden väliseen kommunikaatioon. Jako on hyvin yleinen, mutta siitä on olemassa monia variaatioita – käyttöliittymä ja palvelinpuoli voidaan jakaa useaan osaan, mikä vähentää järjestelmän monoliittisuutta. Käyttöliittymän ja palvelinpuolen kahtiajako voi olla hyvin selkeä SPA-sovelluksen ja APIn välillä tai se voi olla hyvin häilyvä jos käyttöliittymä generoidaan ajonaikaisesti palvelimella sivupohjamoottoreihin pohjautuen. Variaatioita on myös monia näiden välillä. Kommunikaatio verkkosovelluksen ja palvelimen välillä tapahtuu nykypäivänä joko HTTP-protokollilla (HTTP/1.1 tai HTTP/2) tai WebSocket-yhteydellä. HTTP-pyyntö lähetetään verkkopalvelimelle käyttäjän pyytäessä jotain resurssia saapuessaan sivulle hyperlinkillä, lähettäessään lomakkeen tai suorittaessaan haun [22].

Verkkopalvelimet odottavat asiakasohjelmien palvelupyyntöjä ja vastaavat niihin HTTP-pyynnön vastausviestillä. Vastausviesti sisältää statuksen, mikä kertoo pyynnön onnistumisesta tai epäonnistumisesta [22]. Status on numerokoodi, josta 100-alkuiset ovat informatiivisia, 200-alkuiset onnistumista kuvaavia, 300-alkuiset uudelleenohjaukseen viittaavia, 400-alkuiset asiakaspyynnön virheellisyydestä kertovia ja 500-alkuiset kuvastavat palvelinpään virhettä [21]. Tarkempi numero kuvastaa yksityiskohtaisemmin virheen syytä, mutta asiakassovelluksen ei tarvitse ymmärtää kaikkia virhetyyppejä vaikka se olisikin selkeästi tavoiteltavaa [21].

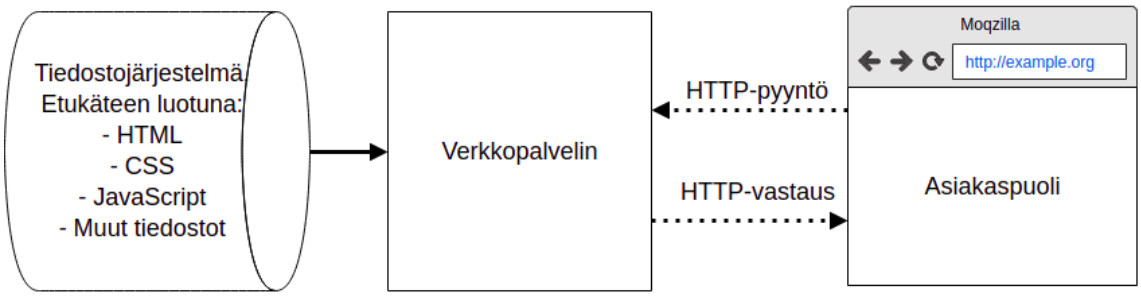
Vastausviesti sisältää myös pyynnön Headerin ja viestin sisällön Bodyssä. HTTP-pyynnön Headeriin voidaan sisällyttää pyyntöön sisältyviä ylimääräisiä informaatiota kuten autentikointiin, välimuistittamiseen, asiakasvinkkeihin, ehtoihin ja kekseihin liittyviä tietoja [20].

HTTP-pyyntö sisältää myös resurssiin osoittavan URLin. Pyyntöihin voidaan sisällyttää lisätietoja myös enkoodattuna URL-parametreihin avain-arvo menetelmällä URLin query-merkkijono -osassa [22]. Lisätietoja voidaan sisällyttää myös kekseihin [22].

HTTP-viestin body-osa sisältää pyynnön tai vastauksen kuorman eli sisällön ja sen pituus ja tyyppi voidaan kertoa HTTP Headerissa [21]. Tyyppi kerrotaan headerissa 'Content-Type' avaimella [21] ja se on JSON-viestityypissä 'application/json' [43]. Viestin sisällön ollessa HTMLlää on arvo samalla avaimella on 'text/html' [21]. Eri tyypit on standardisoitu ja niitä kutsutaan MIME-tyypeiksi (Multipurpose Internet Email Extensions type) ja IANA (Internet Assigned Numbers authority) on vastuussa virallisten MIME-tyyppien ylläpidosta [26, 34].

Web-sovellusarkkitehtuurit voidaan jakaa karkeasti kolmeen osaan. Ensimmäinen päätyyppi on dokumentti, joka on valmiiksi luotuna tiedostojärjestelmään. Toinen päätyyppi on osittain valmiiksi tiedostojärjestelmään luotu templaatti-dokumentti, johon lisätään vielä ohjelmallisesti sisältöä palvelimella ja palautetaan HTTP GET pyynnön mukana valmiina dokumenttina selaimelle. Näistä ensimmäistä vaihtoehtoa kutsutaan staattiseksi verkkosivuksi ja toista dynaamisiksi verkkosivuiksi. Näissä molemmissa tapauksissa sovelluslogiikkaa ei joko ole tai se sijaitsee pääosin palvelimen päässä. Dynaamista verkkosivua kutsutaan useasti palvelinpuolen HTML:ksi (server-side html), palvelinpuolen verkkosivuohjelmoinniksi [22] tai joissakin lähteissä MPA:ksi (Multi-Page Application).

Kuvassa 2.1 on esiteltynä tyypillinen staattisten verkkosivustojen arkkitehtuuri. Verkkoselain pyytää HTTP GET pyynnössä tiettyä dokumenttia. Palvelin pyrkii selvittämään pyydetyn tiedoston tiedostojärjestelmästä ja antaa vastauksen sisältönä dokumentin ja pyynnön statuksena onnistumista kuvaavan numerokoodin (yleensä 200 OK) [22]. Jos dokumenttia ei saada haettua tiedostojärjestelmästä, palautetaan virhetilannetta kuvaava numerokoodi [22] ja mahdollisesti myös sisältönä sanallinen viesti tai virhesivu. Arkkitehtuurin mukaisessa verkkosovelluksessa ei ole varsinaista sovelluslogiikkaa eli sovelluksella ei voida muuttaa järjestelmän tilaa verkkopalvelimen kautta. Sen tila on staattinen eli muuttumaton.

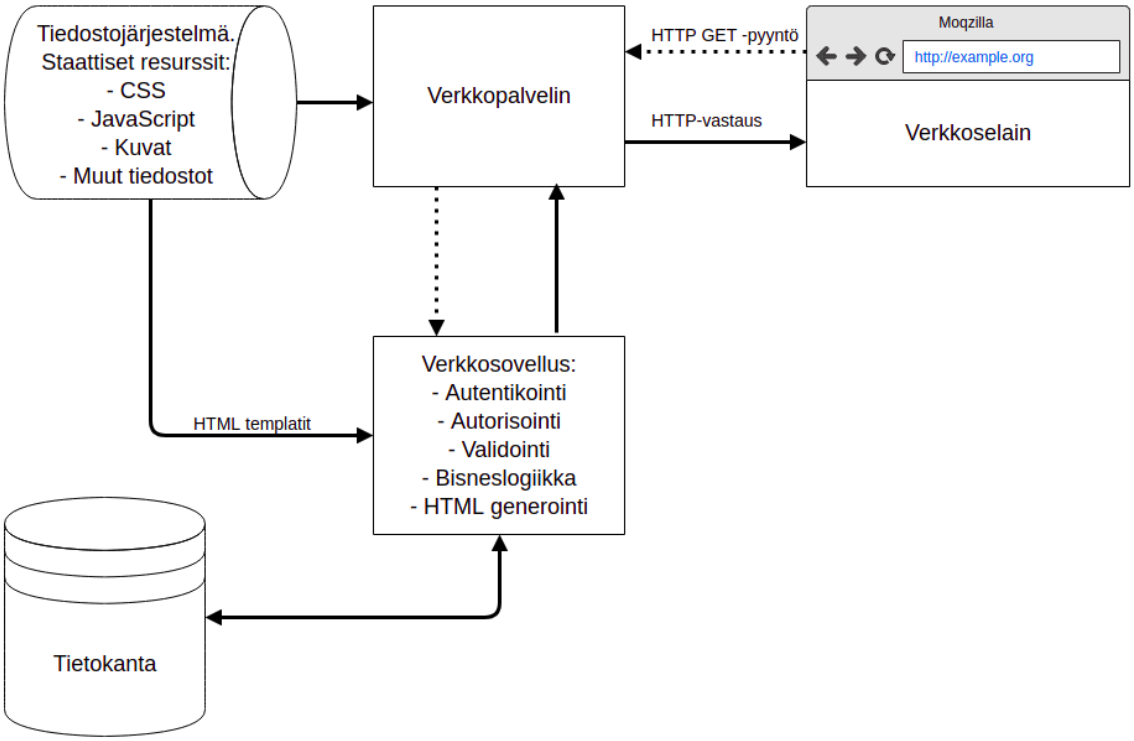


Kuva 2.1. *Tyypillinen staattisten verkkosivujen arkkitehtuuri*

Kun palvelinpuolen staattisiin verkkosivuihin lisätään ohjelmalogiikkaa saadaan aikaiseksi dynaamisia verkkosivuja. Dynaamisissa verkkosivuissa osa sisällöstä luodaan dynaamisesti, jos sille on tarvetta [22]. Ohjelmalogiikka käyttää yleensä jotain ulkopuolista tietolähdettä kuten tietokantaa tai muita ulkoisia rajapintoja, jotka vaikuttavat asiakkaalle

lähetettävän vastauksen sisältöön. Tyypillisesti dynaamisissa sivuissa HTML-dokumentit luodaan käyttäen HTML-templateja, joita täydennetään lisäämällä sisältöä niissä oleviin vakioituihin paikkoihin esimerkiksi käyttämällä tietokannasta ladattua tietoa [22]. Sovelluksen logiikka, autentikointi, autorisointi ja validointi sijaitsevat palvelimella ja verkkoselain esittää käyttäjälle vain palvelimen määrittämiä ja luomia HTML-näkymiä [33, s. 8]. Tällöin palvelin hyvin tyypillisesti palauttaa HTTP-pyynnön sisällössä HTML-dokumentin, johon on voitu liittää tyylejä CSS-tyylikielellä ja interaktiivisuutta tuovia ominaisuuksia JavaScript-ohjelmointikielellä.

Asiakassovellus voi myös antaa verkkopalvelimelle palvelupyynnössä toiveita siitä, minkälaista sisältöä halutaan vastauksessa olevan. Dynaaminen sivu voi palauttaa eri tietoa riippuen siitä, mihin URL:n resurssiin asiakassovellus viittaa [22].



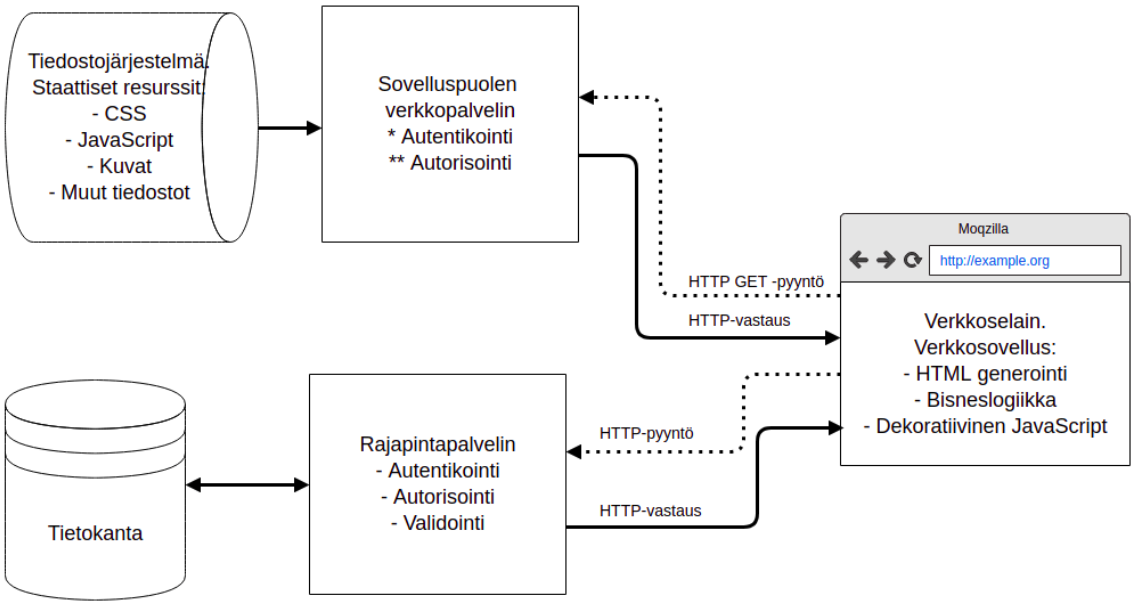
Kuva 2.2. *Tyypillinen dynaamisten verkkosivujen arkkitehtuuri*

Kuvassa 2.2 on esiteltynä tyypillinen dynaamisten verkkosivujen arkkitehtuuri. Verkkoselain tekee HTTP GET pyynnön ja verkkopalvelin antaa ohjelmalogiikan suoritettuaan vastauksena valmiiksi luodut HTML-dokumentit, CSS-tyylit ja JavaScriptit HTTP-vastauksessa.

Dynaamisissa verkkosivuissa käytetään myös staattisten verkkosivujen tavoin staattisia muuttumattomia resursseja, tyypillisesti CSS-tyylejä, JavaScriptejä, kuvia, ennalta luotuja PDF-tiedostoja jne. [22].

Kolmas Web-arkkitehtuurien päätyyppi on malli, jossa näkymään vaikuttavat ohjelmalogiikat on siirretty verkkoselaimeen asiakkaan laitteelle. Tässä mallissa SPA:ksi (Single-Page Application) kutsuttu sovellus toimitetaan verkkoselaimelle, joka ei lataa sivua uudestaan käytön yhteydessä [33, s. 4]. SPA:n voidaan kuvitella olevan lihava asiakaspuo-

len sovellus, joka ladataan verkkopalvelimelta [33, s. 4]. Sovelluksen käytön kuormaa siis siirretään verkkopalvelimelta asiakaspuolelle tehden verkkopalvelimesta laihemman, eli verkkopalvelin joutuu käyttämään vähemmän prosessointitehoa ja muistia palvellessaan asiakaspyyntöjä, koska asiakaspuoli vastaa pääosin verkkosivun generoinnista ja DOM:iin kirjoittamisesta.



Kuva 2.3. *Tyypillinen SPA-verkkosovellusten arkkitehtuuri*

Kuvassa 2.3 on esiteltynä tyypillinen SPA-verkkosovellusten arkkitehtuuri. Kuvassa on eriytetty palvelinpuoli kahteen osaan: sovelluspuolen verkkopalvelimeen ja rajapintapalvelimeen. Kuvaan on tarkoituksellisesti tehty selkeästi eriytetty jaottelu, jotta voidaan havainnollistaa kuinka merkittävästi SPA-sovelluksen arkkitehtuuri voi erota edellä esitellyistä staattisista ja dynaamisista palvelinpuolen verkkosivuista. Kuvan tilanteessa siis käyttöliittymään liittyvät HTML-dokumentit, CSS-tyylitiedostot ja JavaScriptit ladataan kerralla yhdellä haulla ja tämän jälkeen verkkoselaimeen ladattu sovellus hoitaa jatkokommunikoinnit rajapintapalvelimen kanssa. Rajapintapalvelin voi olla esimerkiksi REST API tai GraphQL API. Kuvasta poiketen myös rajapintapalvelin voi käyttää tiedostojärjestelmässä olevia tiedostoja ja myös palautella niitä SPA-verkkosovellukselle. Lisäksi sovelluspuolen verkkopalvelin voi olla pelkästään verkkolevypalvelin, jolloin verkkoselain käy noutamassa suoraan dokumentit tiedostojärjestelmästä. SPA-sovellus sisältää käytännössä vain yhden HTML-dokumentin, vaikka siinä on useita näkymiä; tavallinen dynaaminen tai staattinen verkkosivu on kokoelma useita HTML-dokumentteja [35].

Kuten työpöytäsovellukset, myös SPA-sovellukset eroavat huomattavasti tavallisista verkkosivuista. SPA-sovellus tarjoaa paljon enemmän kuin perinteinen verkkosivun käyttöliittymä. Joidenkin mielestä on jopa täsmällistä kutsua SPA-sovelluksia työpöytäsovelluksiksi, koska ne voivat olla yhtä responsiivisia kuin perinteiset työpöytäsovellukset [33, s. 59].

Edellä esitetyillä päätyypeillä on erilaisia variaatioita ja välimalleja. Yksi tällainen välimalli

on dynaamisten verkkosivujen ja SPA-sovellusten hybridi: SSR (Server Side Rendering), jossa otetaan askel takapakkia SPA-mallista (kuva 2.3). Ideaalisessa SPA-mallissa sovelluspuolen palvelimen ja rajapintapalvelimen välillä on selkeä jako, kun SSR:ssä näin ei täysin ole. Ironisesti tässä mallissa hoidetaan vähintään osa DOMiin kirjoittamisesta palvelimella (SPA:ssa pyritään tästä eroon). SSR:ssä sovelluspuolen palvelin lataa rajapintapalvelimelta suoraan dataa ja esitäydentää SPA-sovellukseen tietoja, jotta asiakaspuolen verkkoselaimen ei tarvitse ladata niitä kaikkia. Tästä voi olla etua esimerkiksi, jos asiakas on hitaan verkkoyhteyden päässä, koska palvelimelta palvelimelle on nopea ja luotettava yhteys [17, 45]. Hitaille asiakaslaitteille tästä on myös etua, koska suoritusta siirretään asiakaspuolelta takaisin palvelinpuolelle [45].

SPA:n perinteistä mallia, jossa palvelimelta ladataan JavaScriptit kutsutaan CSR:ksi (Client Side Rendering). Pääero SPA:n ja SSR:n välillä on se, että SSR-sovelluksen pyyntöviestin vastauksessa on HTML-dokumentti, joka on verkkoselaimelle valmiiksi esiteltävässä muodossa. Verkkoselain voi aloittaa sivuston esittämisen ilman, että sen pitää odottaa linkattujen JavaScriptien lataamista tai suoritusta. CSR-mallissa pitää odottaa virtuaalisen DOM:n siirtämistä verkkoselaimen DOM:iin, jotta sivustosta tulee luettava. [17]

SSR:n etuina on myös parempi hakukoneoptimoituvuus. Hakukoneet kuten Google ja Bing osaavat indeksoida perinteistä verkkosovellusta hyvin, mutta kun sekaan liitetään ulkopuolisesti lähteestä asynkronisesta ladattavaa sisältöä, niin hakukonerobotti ei jää odottamaan sisällön latausta. Tämän seurauksena voi olla tarpeellista käyttää SSR:ää jos haluaa verkkosivun tulevan indesoiduksi sisältöineen. [45] Pelkästään hakukoneoptioiminnin takia ei välttämättä tarvitse siirtyä SSR:iin, vaan tämän tehtävän voi ulkoistaa ylimääräiselle palvelininstanssille tai tehtävän voi myös ostaa ulkoisesta palvelusta. Asiakaspuolen selaimen lataamaa SSR-renderöityä SPA-sovellusta voidaan myös yksilöidä asiakaskohtaisesti palvelimella tätä mallia käyttäen.

SSR:n haitoiksi voidaan luetella se, että palvelinpuolella voi tulla vaikeudeksi jotkin yksittäiset verkkoselaimessa ajettavaksi tarkoitetut ohjelmakoodit, joita SSR:ssä ajetaan ainakin osittain palvelimella. Kun SPA-sovellusta voidaan tarjota staattiselta tiedostopalvelimelta niin SSR renderöity SPA-sovellus tarvitsee jotain ajoympäristöä, josta sovellusta voidaan tarjota. SSR lihavoi myös palvelinta ja tekee palvelimesta laskentapainotteisemman verrattuna staattisten tiedostojen jakamiseen. Tästä voi seurata se, että verkkoliikennekuormaan ja välimuistittamiseen pitää kiinnittää enemmän huomiota. [45]

2.2 REST

Tim Bernes-Leen luotua WWW:n ensimmäisen verkkosivun vuonna 1991, lähti WWW:n suosio eksponentaaliseen kasvuun. Viiden vuoden kuluttua käyttäjiä oli jo 40 miljoonaa ja käyttäjämäärä kaksinkertaistui jokaisen kahden kuukauden välein. WWW:stä oli kasvamassa liian iso liian nopeasti ja se oli matkalla tuhoonsa. Tuhon aikaan internetin infran kapasiteetti oli jäämässä jälkeen kasvutahdin kanssa. Tuohon aikaan verkossa käytetyt

protokollat eivät tunteneet välimuistittamista eivätkä kuorman tasaamista. Epävarmuutta oli WWW:n skaalautuvuudesta jatkossa ja kyvystä palvella kysynnän kasvun jatkuessa. [23, s. 2]

Vuoden 1993 lopulla yksi Apache-yhtiön perustajista, Roy Fielding huolestui WWW:n skaalautuvuudesta [23]. Tutkiessaan tilannetta Fielding etsi käytännönläheistä ratkaisua ja päätyi joukkoon rajoitteita, joilla mahdollistetaan WWW:n kasvu. Fielding esitteli RESTin rajoitteet väitöskirjassaan Kalifornian yliopistolla vuonna 2000, mutta se ei herättänyt vielä silloin paljon huomiota. Mutta nyt vuosien jälkeen REST-arkkitehtuurimalli ja sen rajoitteet ovat saaneet laajan suosion internetissä ja se tarjoaa helpomman vaihtoehdon SOAPille ja WSDL-pohjaisille verkkopalveluille. Suuryritykset kuten Yahoo, Google ja Facebook ovat ottaneet pois käytöstä SOAP- ja WSDL-pohjaisia palveluitaan ja korvanneet niitä helpommin käytettävillä ja resurssipohjaisilla REST-arkkitehtuurimalleilla. [37]

Fielding jakoi REST-arkkitehtuurimallin rajoitteet kuuteen eri kategoriaan, joilla pyritään standardoimaan REST:n malli. Nämä kuusi rajoitetta ovat [10, s. 76-85]:

1. asiakas-palvelin -malli
2. tilattomuus
3. välimuistittaminen
4. yhdenmukainen rajapinta
5. kerroksittainen järjestelmä
6. ladattava koodi (Code-On-Demand)

Seuraavaksi avataan jokaista yksittäistä rajoitetta omissa kappaleissaan.

Asiakas-palvelin -malli (rajoite 1) tarkoittaa käyttöliittymän liittyvien asioiden erottamista dataan liittyvistä asioista. Tällä jaolla parannetaan skaalautuvuutta, kun palvelinkomponentteja saadaan yksinkertaistettua sekä saadaan parannettua siirrettävyyttä useiden alustojen välillä. Ehkä tärkeintä tässä WWW:lle on se, että jaottelu mahdollistaa komponenttien kehittämisen itsenäisesti toisistaan riippumatta. Tämä parantaa myös internetkokoluokan vaatimusta, jossa organisaatiolla voi olla useita domaineja. [10, s. 78]

Tilattomuus (rajoite 2) tarkoittaa sitä, että asiakas-palvelin-kommunikoinnissa viestit asiakkaalta palvelimelle sisältävät kaikki tarvittavat tiedot viestin ymmärtämiseksi palvelimella, ja asiakassovellus ei pysty käyttämään pyynnössään hyväksi palvelimen omaan kontekstiin sidottua tietoa. Tämä rajoite sisällyttää sisäänsä läpinäkyvyyden, luotettavuuden ja skaalautuvuuden. Läpinäkyvyys parantuu, koska monitorointijärjestelmän ei tarvitse vertailla monen pyynnön välisiä yhteyksiä ymmärtääkseen, mitä pyynnössä tapahtuu. Luotettavuus parantuu, koska tilattomuus helpottaa palautumista osittaisista häiriöistä. Skaalautuvuus parantuu, koska palvelimen ei tarvitse säilyttää muistissaan useiden pyyntöjen välisiä asiayhteyksiä ja tämän takia resursseja vapautuu. [10, s. 78-79]

Välimuistitusta (rajoite 3) tarvitaan verkkoviestinnän tehostamiseen. Välimuistirajoite vaa-

tii vastausvietin datan olevan implisiittisesti tai eksplisiittisesti merkitty välimuistitetuksi tai välimuistittamattomaksi. Vastauksen ollessa välimuistitettu on asiakkaalla oikeus käyttää samaa dataa uudestaan vastaaviin käyttötarkoituksiin. Välimuistituksen etuina on mahdollisuus osittaisesti tai kokonaan poistaa joitakin yhteydenottoja, parantaa tehokkuutta, skaalautuvuutta ja vähentää asiakkaan kokemaa keskimääräistä latenssia. Välimuistittaminen voi kuitenkin vähentää luotettavuutta, jos välimuistin data poikkeaa merkittävästi datasta, joka olisi lähtetty suoraan palvelimelta välimuistittamattomana. [10, s. 79-80]

Yhdenmukainen rajapinta (rajoite 4) on keskeinen REST-arkkitehtuurin ominaisuus verrattuna muihin verkkoyhteyspohjaisiin tapoihin yhdenmukaistaa rajapintaa eri komponenttien välillä. Järjestelmän arkkitehtuuria saadaan yksinkertaistettua ja komponenttien välisen kommunikaation läpinäkyvyyttä parannettua, kun rajapintoja yhdenmukaistetaan. Yhdenmukaistaminen vähentää kuitenkin tehokkuutta, koska standardoidussa muodossa lähetetty viesti ei välttämättä vastaa asiakassovelluksen tarpeita. Yhdenmukaistaminen optimoi kuitenkin yleistä WWW-pohjaista käyttötapaa, mutta ei ole optimaalinen muiden tyyppisille sovellusarkkitehtuureille. [10, s. 81-82]

Kerroksittainen järjestelmä (rajoite 5) hierarkisoi järjestelmän eri kerrokset niin, että komponentti eivät näe sen komponentin ohitse, jonka kanssa komponentti kommunikoi. Kun komponenteille annetaan tietyt vastuualueet, saadaan järjestelmän kompleksisuutta vähennettyä ja yhden yksittäisen kerroksen itsenäisyyttä korostettua. Kerrostamalla voidaan myös kapseloida legacy-järjestelmiä ja suojata uusia järjestelmiä siirtämällä harvoin käytettyjä ominaisuuksia jaetuille välittäjille. Välittäjät voivat parantaa järjestelmän skaalautuvuutta mahdollistamalla palvelimien kuormantasauksen usean tietoverkon ja prosessorin välillä. Suurin haitta kerroksisuudesta on sen lisäämä ylimääräinen datan prosessointi ja sen tuoma ylimääräinen latenssi, mikä heikentää käyttäjän kokemaa palveluntasoa. [10, s. 82-83]

Ladattava koodi (rajoite 6) on rajoitteista ainoa valinnainen ja se mahdollistaa asiakassovellusten toiminnallisuuksien laajentamisen lataamalla toiminnallisuuksia palvelimelta ohjelmakoodien muodossa. Tämä yksinkertaistaa asiakassovelluksia, koska se vähentää vaadittujen ennestään toteutettujen toiminnallisuuksien määrää. Osa toiminnallisuuksista voidaan ladata palvelimelta järjestelmän toimituksen jälkeen. [10, s. 84-85]

Verkkopalvelun APIa, joka noudattaa REST-arkkitehtuurimallia kutsutaan REST APIksi. REST APIt ovat rajapintoja, joita asiakassovellukset voidat hyödyntää. API tarjoaa dataa ja toiminnallisuuksia mahdollistaen interaktiot tietokoneelta-tietokoneelle ja kommunikoinnin niiden välillä. APIja ovat verkkopalveluiden julkisia rajapintoja, jotka kuuntelevat ja vastailevat asiakkaiden palvelupyyntöihin. REST-arkkitehtuurimallia käytetään usein modernien verkkopalveluiden APIen suunnittelussa. Verkkopalvelua kutsutaan RESTfuliksi sen sisältäessä REST APIn. REST API sisältää joukon toisiinsa linkattuja resursseja. [23, s. 5-6]

Fielding työskenteli myös muiden tutkijoiden kanssa parantaakseen verkkopalveludien skaalautuvuutta. Joukkoon kuului myös WWW:n luoja Tim Berners-Lee. He määrittelivät

uuden version HTTP:stä ja uusi määritelmä sai nimekseen HTTP/1.1. He myös formalisoivat URI-syntaksin (Uniform Resource Identifiers) RFC 3986 standardiin. [4, 11, 23, s. 5]

REST APIt käyttävät URIa jaotellakseen resurssit osoitteiden muodossa omille käsittelijöilleen (controller). URI:n formaatti on määritelty standardissa ja geneerinen URI on muotoa [4, s. 15]:

$$\begin{aligned}&\text{URI} = \text{scheme ":" hier-part ["?" query] ["\#" fragment]}\\&\text{hier-part} = \text{"//" authority path-abempty}\\&\qquad\qquad / \text{ path-absolute}\\&\qquad\qquad / \text{ path-empty}\end{aligned}$$

URIn skeema- ja path-osat ovat vaadittuja osia, vaikkakin path voidaan jättää tyhjäksi. Jos authority osa on mukana, niin pathin pitää joko olla tyhjä tai se alkaa vinoviiva ("/") merkillä. Jos authority ei ole mukana, path-osa ei voi alkaa kahdella vinoviivalla ("//"). [4]

Hyvä resurssien jako URLilla on johdonmukainen ja sillä on selkeä jaottelu. Hyviä suunnittelutapoja on monia, mutta niitä ei käsitellä tässä työssä tämän enempää, koska ne eivät ole merkityksellisiä työn osalta. Esimerkki HTTP URIsta voisi kuitenkin olla seuraavanlainen:

$$\text{http://api.example.com/v1/todos?level=important}$$

Esimerkin URI tai tässä tapauksessa URL (Uniform Resource Locator) sisältää skeeman, omistajan, polun ja toiminnon. Skeema on 'http' ja se on erotettu '://'-merkein omistaja-osasta, joka on 'api.example.com'. URIn standardoitiin aikanaan kaksi täysin turhaa vinoviivaa '//' (skeeman ja kaksoispisteen jälkeen) ja standardoinnissa mukana ollut Berners-Lee on ollut siitä pahoillaan [3]. Vinoviivat ovat kuitenkin pysyneet mukana spesifikaatiossa, koska niiden poistaminen jälkikäteen rikkoisi niitä käyttäviä järjestelmiä. Kyseisen esimerkin polku-osa on '/v1/todos' ja toiminto-osa 'level=important'. Skeema kuvastaa käytettävää protokollaa ja omistaja-osa on DNS-osoite, joka vastaa tiettyä ennalta määritettyä IP-osoitetta internetin osoiteavaruudessa. DNS-osoite auttaa käyttäjää muistamaan tietyn palvelimen osoitteen ihmiselle luettavammassa muodossa. Polku-osa osoittaa käyttäjän osoittamaan haluttuun resurssiin, tässä esimerkissä todo-resursseihin. Polun alussa oleva 'v1' tarkoittaa versionumeroa ja versioinnin voisi tehdä monella muullakin tapaa. Rajapinnan muuttuessa niin paljon, että se voisi rikkoa asiakassovelluksien toimintaa, tehdään muutosta koskevalle resurssille yleensä kokonaan uusi osoitin, josta uusiin toiminnallisuuksiin tai muuttuneeseen datamalliin pääsee käsiksi. Toiminto-osa on avain-arvo-pariin pohjautuva hakumerkkijono, jossa asiakassovellus voi toimittaa palvelimelle pyyntöön liittyviä lisätoiveita.

Tavalliset dynaamiset verkkosivut (MPA:t) käyttävät yleensä HTML-lomakkeita (form) viestien välittämiseen, mutta REST APIt toteuttavat yleensä CRUD (create, retrieve, update ja delete)-toiminnallisuudet, koska joitakin sovelluskohtaisia toimenpiteitä ei voida toteuttaa pelkästään standardimetodeilla. [23, s. 16]

REST API käyttää hyväkseen kaikkia HTTP/1.1-protokollan toiminnallisuuksia sisältäen pyyntöjen eri metodit, vastauskoodit ja viestien otsikot. [23] Pyyntöjen metodi on määritelty RFC 2616 standardissa ja se liitetään mukaan HTTP-pyyntöön 'Request-Line'-osassa [21]:

$$\text{Request-Line} = \text{Method} \quad \text{SP} \quad \text{HTTP-Version} \quad \text{CRLF}$$

Tässä yhteydessä SP:llä tarkoitetaan polkua ja CRLF:llä rivinvaihtoa. Jokaisella HTTP verbillä (method) on tarkasti määritelty semantiikka REST APIn resurssimallissa. GET on noutamista tarkoittava verbi, PUTilla kuuluu lisätä uusi resurssi tai päivittää jo olemassa olevaa, DELETE:llä kuuluu poistaa resurssi, POSTilla kuuluu luoda uusi resurssi kokoelmaan. Lisäksi on HEAD, jota on käytetty resurssin metadatan noutamiseen. OPTIONSia käytetään metadatan noutamiseen niin, että vastaukseen liitetään sallittujen verbien joukko. [23, s. 23-27] Lisäksi on olemassa PATCH-verbi, jolla kuuluu päivittää jo olemassa olevaa resurssia.

REST API vastaa asiakassovellukselle HTTP-paluuviestissä operaation tuloksesta statuskoodilla. Statuskoodeja käytiin läpi luvussa 2.1. GET-, POST-, PUT- ja PATCH-operaatioiden onnistuessa palvelin lisää paluuviestin runkoon mukaan resursseihin liittyvää dataa, esimerkiksi JSON muodossa. REST APIt yleensä käyttävät tekstiformaattia esittääkseen resurssin tilaa, tekstiformaatti sisältää yleensä joukon merkityksellisiä datakenttiä. Nykypäivänä juuri XML- ja JSON-tietoformaatit ovat useimmiten käytettyjä tekstiformaatteja vastauksen sisällölle [23, s. 47].

Kuva 2.4. *Esimerkki RESTful-kommunikoinnista*

Kuvan 2.4 esimerkissä on asiakassovellus kuvattuna tietokoneen symbolilla ja REST API-rajapintapalvelin laatikoituna ja pilvi-symbolilla esitettynä. Kuvan tarkoituksena on esitellä kevyesti asiakassovelluksen kommunikointia REST APIn kanssa tilanteessa, missä asiakassovellus hakee ensin käyttäjätietoa ja sen jälkeen käyttäjään sidottuja Posteja. Kuvan tilanteessa 1 asiakassovellus hakee käyttäjän tiedot tunnisteen (id:n) perusteella. Saatuaan tiedot käyttäjästä asiakassovellus tekee seuraavan palvelupyynnön kohdassa 2 ja pyytää käyttäjän tekemiä artikkeleita käyttäjän tunnisteen perusteella. Yleensä RESTful APIn kanssa kommunkointi on hyvin vastaavanlaista, eli kyseisen esimerkin tavoin haetaan tietoja paloittain aina tarpeen mukaisesti. Monesti pyyntöjen järjestyksellä on merkitystä eli on suoritettava joku tietty pyyntö ja vastauksen sisällön avulla voidaan tehdä jatkopyyntöjä.

2.3 GraphQL

GraphQL on Facebookin kehittämä kyselykieli asiakassovelluksille ja se tarjoaa intuitiivisen ja joustavan syntaksin sekä tavan kuvata tietomallit ja niiden väliset vuorovaikutukset. GraphQL ei ole ohjelmointikieli, mutta sen avulla voidaan kysellä rajapintapalvelimelta tietoja palvelimilta, jotka noudattaa GraphQL-määrittelyä. GraphQL-kyselykieli ja sen ajoympäristö kehitettiin alunperin vuonna 2012 Facebookin omaan käyttöön ja avoimen lähdekoodin versiota alettiin kehittämään vuonna 2015. GraphQL ei vaadi taustalleen mi-

tään tiettyä ohjelmointikieltä tai tiedon varastointimenetelmää. Sen sijaan rajapintapalvelin liittää omat tietomallinsa ja tyypitykset GraphQL:n kanssa yhteensopiviksi. Tämä tarjoaa yhtenäisen palvelurajapinnan ja rikkaan tuotekehitysympäristön sovelluksien kehittämiselle. [15]

GraphQL:n kehityksen taustalla olleet suunnitteluperiaatteet [15]:

- **Hierarkkisuus**: Hierarkkisen tiedon lisäys ja muokkaus. GraphQL-kysely muodostetaan vastaamaan paluuviestissä olevaa tietorakennetta. Tämä on luonnollinen tapa kuvailla kyselyvaatimuksia.
- **Tuotekeskeisyys**: GraphQL on luotu käyttöliittymien ja käyttöliittymäkehittäjien tarpeita ajatellen.
- **Vahva tyypitys**: Kyselyt suoritetaan palvelimen oman tyypityksen mukaisesti. Työkaluilla pystytään tarkistamaan kyselyn syntaktinen oikeellisuus ennen sen suoritusta. Kehityksen aikana palvelin voi taata paluuviestin muodon ja sen luonteen.
- **Asiakassovellukselle yksilöidyt kyselyt**: GraphQL-rajapinnan tarjoava palvelin määrittelee vain kyvyn tarjota tiettyä informaatiota. On asiakassovelluksen vastuulla määritellä itse, miten se haluaa hyödyntää tarjottua informaatiota. Asiakassovellus määrittää itse, mitä tietoa se haluaa palvelimen palauttavan eikä tyydy vain palvelimen palauttamaan dataan palvelimen päätöksen mukaisesti. GraphQL:ssä asiakassovellus kysyy tarkasti, mitä se haluaa, eikä palvelin palauta yhtään enempää kuin mitä on pyydetty.
- **Itseään kuvaileva**: GraphQL-palvelimelta pystyy aina hakemaan dataa GraphQL-kyselykielen määritelmien mukaisesti. Kyselykielen tarkat määritelmät helpottavat erilaisten työkalujen ja kirjastojen kehittämistä teknologian ympärille.

GraphQL on protokollariippumaton eikä riipu HTTP-protokollasta. HTTP-protokollan metodeja tai HTTP-vastauskoodeja ei käytetä sisäisesti GraphQL:n toteutuksessa. Sen sijaan HTTP on yksi kanava GraphQL kyselyjen ja vastauksien välittämiseen, ja HTTP on luonnollisesti suosittu viestiväylä WWW-maailmassa. [5, s. 11] GraphQL:n eri dokumentaatiot usein suosittelevat kuitenkin toteuttamaan rajapinnan käyttäen HTTP:n GET- ja POST-verbejä [38].

GraphQL-rajapintaa tarjotaan usein yhdestä HTTP-päätepisteestä (polusta). Tässä on ero REST API-rajapintoihin, jotka tarjoilevat yleensä joukon HTTP-päätepisteitä, joista jokainen tarjoaa vain yhden resurssin. GraphQL-palvelut vastaavat tyypillisesti käyttäen JSON-formaattia viestimuotona, vaikkakaan GraphQL-määrittely ei vaadi sitä. JSON pakattuna GZIP:llä tarjoaa hyvän suorituskyvyn, koska JSON on pääosin tekstiä ja teksti yleensä pakkautuu erittäin hyvin GZIP:llä. [14] Tekstipohjainen sisältö saadaan pakatta parhaimmillaan 70-90% pienemmäksi GZIP:llä [16]. Paluuviestin pakkaaminen aiheuttaa palvelimelle hieman ylimääräistä laskentaa, mutta paluuviestin kokoa saadaan pienennettyä - välimuistuksen jälkeen sisältöä ei edes tarvitse pakata uudestaan vaan se voidaan lähettää välittömästi pakattuna. Lisäksi asiakassovellus selviää lyhyemmällä latausajalla etenkin, jos isokokoinen paluuviesti on pakattuna tai asiakassovellus viestii

palvelimen kanssa hitaan verkkoyhteyden päästä.

Asiakassovellukset tekevät kielen mukaisia kyselyitä GraphQL-palvelimelle. GraphQL-tietolähteitä kutsutaan asiakirjoiksi. GraphQL-palvelin tarjoaa asiakirjoille haku-(Query), editointi-(Mutation) ja tilaus-(Subscription) -palveluita. Kyselyitä voidaan myös koostaa osakyselyistä. [15] GraphQL-tietolähteet esitellään ihmisille luettavissa olevasta skeemasta. Skeema määritellään GraphQL:n SDL:llä (Schema Definition Language) ja siinä esitetään saatavissa olevat tietotyypit ja miten ne ovat relaatiossa keskenään. [44]

Kyselystä riippumatta HTTP:tä käytettäessä vastauksen tulisi olla liitettynä mukaan JSON-formaatissa. Kyselyn tuloksesta riippuen paluuviestissä saattaa olla dataa tai virheviestejä. Niiden kuuluu tulla osana JSON-formaatissa olevaa paluuviestiä [38]:

```
{
    "data": { ... },
    "errors": [ ... ]
}
```

GraphQL-rajapinnat voidaan versioida REST APIen tapaan, mutta sitä pyritään välttämään kaikin mahdollisin keinoin tarjoamalla työkaluja jatkuvaan GraphQL-skeeman päivittämiseen. REST APIa versioidaan, koska asiakassovelluksella on hyvin vähän kontrollia dataa, jota rajapinnan tarjoamat metodit palauttavat. Kaikkia rajapintaan tehtäviä muutoksia pitää ajatella hajoittavana muutoksina ja hajoittavat muutokset tarvitsevat aina uuden version. GraphQL:n tapauksessa rajapinta palauttaa vain dataa, jota asiakassovellus eksplisiittisesti pyytää, eikä yhtään enempää, joten uusia kenttiä ja tyyppejä voidaan lisätä tekemättä hajoittavaa muutosta olemassa olevaan järjestelmään. GraphQL-rajapinnan kehityksessä on yleinen tapa olla tekemättä hajoittavia muutoksia ja tarjoilla versiotonta rajapintaa. [14] Asiakirjojen kenttiä voidaan kuitenkin poistaa käytöstä käyttämällä @deprecated-direktiiviä. Direktiivin ansiosta käytöstä poistettua kenttää voidaan edelleen käyttää kyselyssä, mikä varmistaa etteivät asiakassovellukset hajoa muutoksesta [15].

GraphQL:n Query-tietotyyppi on tiedon hakemiseen tarkoitettu vain-luku-tyyppi ja on verrattavissa REST APIn GET-verbiin. Sillä määritellään mitkä tietokyselyt ovat mahdollisia palvelimelta ja Query määritellään SDL:ssä. Query on yksi juuritason tietotyypeistä, jotka määrittelevät palveluita asiakassovelluksille ja toimivat tulokohtana (entry-point) muille skeemassa tarkemmin määritellyille tietotyypeille. [44]

Mutation-tietotyyppi on tarkoitettu tiedon muokkaamiseen, lisäämiseen ja poistamiseen. Mutation on verrattavissa REST APIn PUT-, POST-, PATCH- ja DELETE-verbeihin. Mutation määritellään vastaavanlaisesti SDL:n juuritasolla kuin Query ja se toimii tulokohtana kaikille datan muokkausoperaatioille. [44]

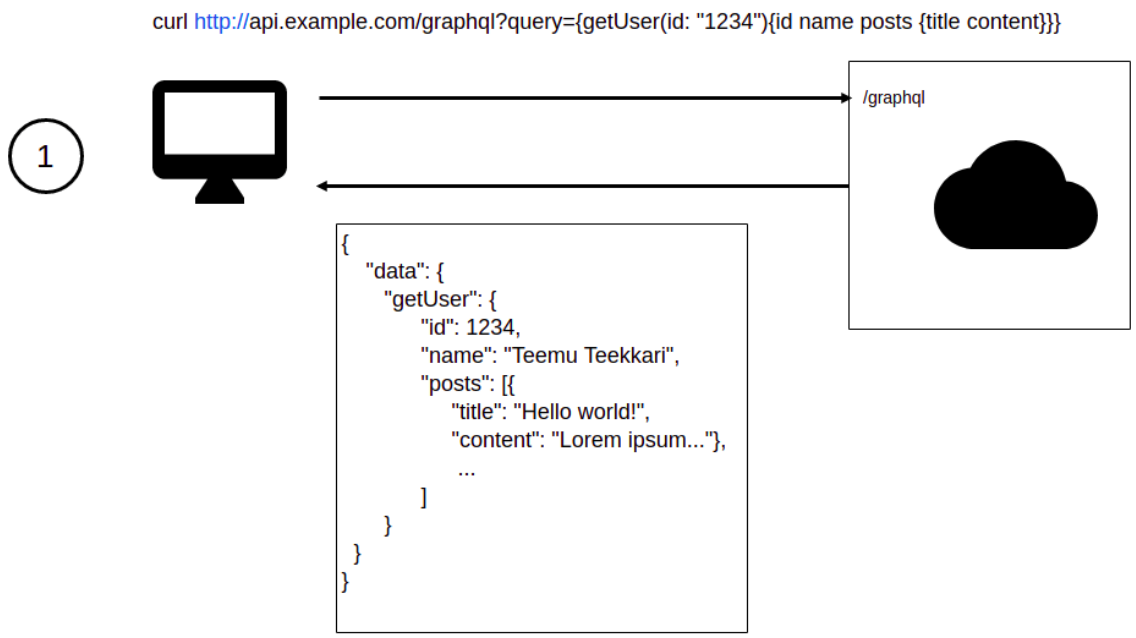
Subscription-tyyppi mahdollistaa GraphQL-operaatioiden seurannan ja sen avulla on mahdollista tilata lähetyksiä tapahtumista (event) [42]. Subscriptionit mahdollistavat reaaliaikaisen seurannan ja ne on voitu toteuttaa esimerkiksi WebSocket-tekniikalla [2]. WebSocket-teknologissa yhteyden avaamisen kättely (handshake) tehdään usein HTTP-

```
type Book {
    id: ID!
    title: String!
    content: String!
}

type User {
    id: ID!
    name: String
    posts: [Post]
}

type Query {
    getUser(id: ID!): User
}

type Mutation {
    addPost(title: String!, content: String!): Post
}

type Subscription {
    postAdded: Post
}
```

Ohjelma 2.1. *Esimerkki GraphQL-skeeman typeDefs osasta.*

protokollan kautta ja asiakassovellus pyytää palvelimelta protokollan vaihtamista 'Connection: Upgrade' ja 'Upgrade: websocket' Headereja käyttäen [9, s. 6]. HTTP- ja WebSocket-protokollilla on yhteistä se, että ne käyttävät vakiona samoja portteja. Porttia 80 käytetään salauksettomaan tiedonvaihtoon ja salattuun tiedonvaihtoon porttia 443 (SSL/TLS) [9, s. 11]. WebSocket-protokolla mahdollistaa jatkuvan tiedonvaihdon molempiin suuntiin yhtä TCP-sockettia käyttäen toisin kuin HTTP, jossa asiakassovellus ottaa toistuvasti uusia yhteyksiä ja eli siinä käytetään useaa TCP-sockettia [9, s. 4].

Ohjelmassa 2.1 on esitettynä GraphQL typeDefs-osa, jossa on kaksi GraphQL-oliotietotyyppiä: Book ja User. Molemmilla tietotyypillä on omia tietokenttiä kuten esimerkiksi Book:lla on kentät id, title ja content. Oliotyyppiä User voidaan hakea id-argumentin perusteella GraphQL-kyselyllä, koska tyypin Query alle on määritelty getUser-metodi. Huutomerkki tyypin perässä tarkoittaa pakollista (non-null) kenttää [15]. Esimerkistä on jätetty pois resolvers-osa, jossa toteutetaan skeemaan typeDefsin Queryssä määritetyt metodit. TypeDefs-osassa määritellään, mitä rajapinta pitää sisällään, ja asiakassovelluksien ohjelmistokehittäjät pääsevät jo pelkästään sitä lukemalla hyvin pitkälle työssään. Ohjelma pitää sisällään myös tyypin Mutation, jonka alla on esiteltynä addPost-metodi. Tämä metodi ottaa parametreinaan pakolliset title- ja content-muuttujat, jotka ovat String-tyyppisiä. Metodi lupaa palauttaa onnistuneen lisäyksen jälkeen paluuviestissä Post-olion. Ohjelman Subscription-tyyppi tarjoaa postAdded-metodin, jolla on mahdollista tilata ilmoituksia uusien Post-olioiden lisäyksistä. Käyttämällä metodia voi pyytää yhteyden vaihtamista HTTP:sta WebSocket-prokollaan, mikä mahdollistaa jatkuvan TCP-yhteyden. Palvelin valikoi oman logiikkansa mukaisesti auki olevat yhteydet, joihin se jakaa näitä ilmoituksia aina, kun uusi Post on lisätty.



Kuva 2.5. *Esimerkki GraphQL:n query-kyselystä*

Kuvassa 2.5 on esimerkki ohjelman 2.1 getUser-metodin käytöstä. Yhdellä HTTP pyynnöllä voidaan hakea myös Useriin liitetyt Postit. Kuvassa 2.4 haettiin samat datat REST APIn kautta kahdessa osassa, GraphQL-rajapinnasta kaikki tarvittava saatiin yhdellä pyynnöllä. Kuvassa huomionarvoista on myös se, että posts-taulukon sisältämissä Post-olioissa ei ole mukana id-kenttiä, koska niitä ei CURL-pyynnössä eksplisiittisesti pyydetty.

2.4 Tunnisteiden tunnistaminen optisesti

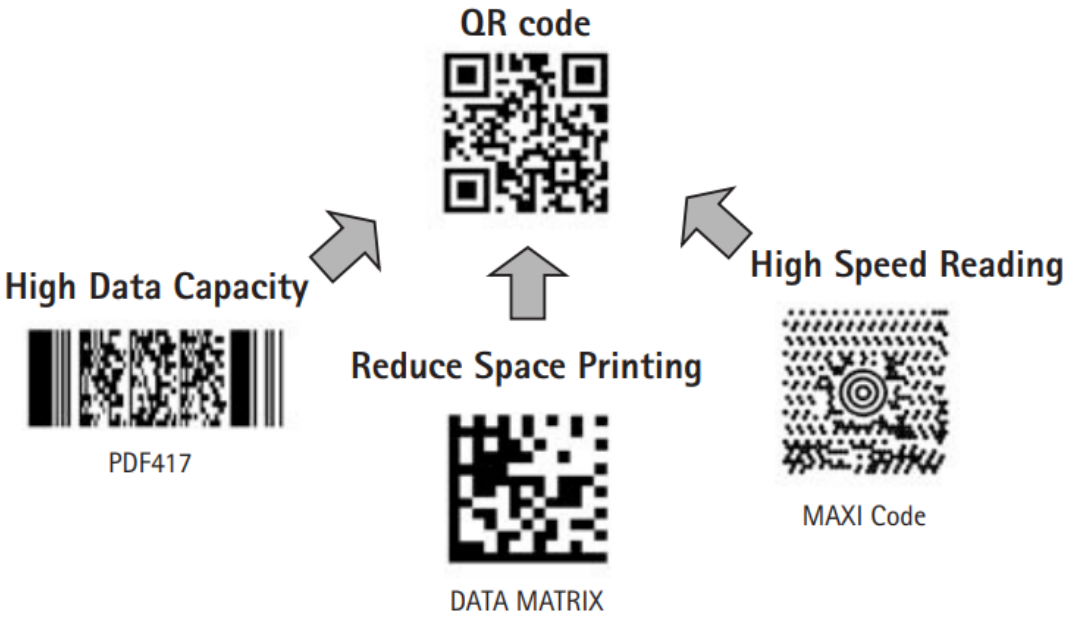
Optisella tunnistamisella tarkoitetaan tietoalkioiden lukemista koneellisesti esimerkiksi digitaalikameraa hyödyntämällä. Digitaalikameralla voidaan ottaa valokuvia, joita voidaan siirtää tietoväylien avulla tietokoneen prosessoitavaksi hyvin nopeasti, ja tietokone voi erilaisten algoritmien avulla etsiä kuvista haluttua informaatiota. Kuvantunnistaminen on laaja sovelluskohde, jota voidaan tehdä esimerkiksi algoritmisesti tai koneoppimisen avulla. Tässä työssä valokuviin liittyvä tunnistaminen rajataan algoritmiseen viivakoodien ja QR-koodien lukemiseen.

2.4.1 QR-koodi

QR-koodi on 2-ulotteinen symboli, jonka Toyotan tytäryhtiö Denso keksi vuonna 1994 [40, s. 60]. QR-koodista on olemassa ISO-standardi (ISO/IEC18004), joka on hyväksytty kesäkuussa 2000. Tämä kaksiulotteinen symboli kehitettiin alunperin autoteollisuuden tarpeisiin auton osien kontrollointiin auton valmistuksen tuotantoketjussa. Nykyisin QR-

koodista on tullut suosittu muissa käyttötarkoituksissa. [40, s. 60] Tässä luvussa käydään QR-koodin periaatteet läpi, mutta tarkempiin teknisiin yksityiskohtiin ei mennä.

Vuonna 1970 yhdysvaltalainen yritys IBM kehitti UPC-symbolit, joihin oli mahdollista mahduttaa 13 numeroa mahdollistaen automaattisen tunnistamisen tietokoneella. Nämä lineaariset viidakoodit ovat edelleen laajasti käytössä erilaisten myyntipisteiden järjestelmissä. Vuonna 1974 kehitettiin Code 39, johon pystyi koodaamaan noin 30 aakkosnumeerallista merkkiä. Sen jälkeen vuonna 1980 tuli kerroksellisia koodeja, joihin pystyi mahduttamaan noin 100 merkkiä: esimerkiksi Code 16K ja Code 49 kehitettiin tällaisiksi. Informaation ja teknologian kehittyessä heräsi tarve kehittää symboleja, joihin voidaan tallentaa yhä enemmän tietoa ja myös muun kielen tekstejä kuin englantia. Vanhoihin symboleihin voitiin sisällyttää pääsääntöisesti osajoukkoa ASCII-merkistöstä Code 128:n tukiessa täyttä ASCII-merkistöä. Tuloksena syntyi QR-koodi, johon voidaan mahduttaa noin 7000 merkkiä sisältäen myös mahdollisuuden lisätä Kanji-kielen merkkejä (Japanissa käytettyjä kiinan kielen merkkejä). QR-koodi kehitettiin vuonna 1994. Yleisesti ottaen 2-ulotteiset symbolit pystyvät sisällyttämään noin 100-kertaisen määrän dataa verrattuna lineaarisiin symboleihin kuten viivakoodeihin. 2-ulotteisuuden mahdollistaman suuren datamäärän takia datan käsittely on hitaampaa ja monimutkaisempaa. Tästä syystä QR-koodissa on erityinen osumakuvio, joka mahdollistaa nopean koodin lukemisen. [40, s. 60-61]



Kuva 2.6. *QR-koodin kehitys*
[40, s. 61]

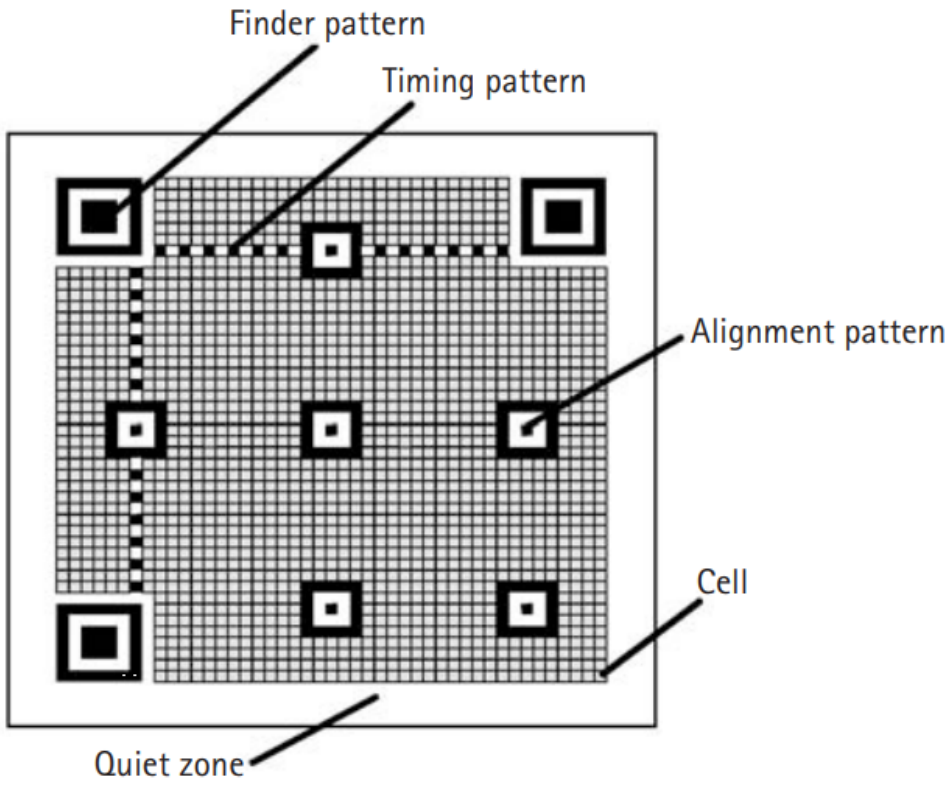
QR-koodi on matriisisymboli, joka kehitettiin yhdistämään standardien PDF417, Data Matrix ja MAXI Code hyvät puolet. PDF417 mahdollisti suuren kapasiteetin, Data Matrix suuren tallennustiheyden pieneen tilaan ja MAXI Code oli nopea lukea sen erityisen osumakuvion ansiosta. [40, s. 60-61] Kuvassa 2.6 ovat esiteltynä nämä kolme ja ylimpänä itse QR-koodi.

QR-koodia voidaan lukea nopeasti kaikista suunnista tai kulmista riippumatta kameran asennosta. Matriisisymbolin lukeminen toteutetaan CCD-anturilla. Anturilla skannatut rivit tallennetaan muistiin ja sitten tietokoneen ohjelmistoa käyttäen yksityiskohdat erotellaan ja eri kuviot tunnistetaan. Kun sijainnit, koko ja kulmat tunnistetaan symbolista, voidaan koodauksen purkamisprosessi aloittaa. QR-koodissa on kolme osumakuviota kolmessa sen kulmassa, mikä mahdollistaa sen nopean lukemisen kaikista 360 asteen kulmista. [40, s. 62]

QR-koodissa oleva kohdistuksessa auttava kuvio mahdollistaa myös koodin taittamisen ja sen lukemisen vaikka symboli ei ole aivan suorassa tai se on kupruilla. Tämä mahdollistaa symbolin lukemisen pinnoilta, jotka eivät ole aivan suoria. [40, s. 63]

Virheenkorjaus auttaa QR-koodien lukemisessa, kun symboli vahingoittunut osittain. QR-koodiin on mahdollista liittää joku neljästä eri virheenkorjauksen tasosta (7%, 15%, 25% tai 30% per symboli). Virheenkorjaus on toteutettu erilaisia vaurioita varten ja virheenkorjaus käyttää apunaan Reed-Solomon-algoritmeja, joka auttavat symbolin lukemisessa ja virheenkorjauksessa symbolin vaurioiduttua. QR-koodin tekijä voi vaikuttaa siihen, minkä virheenkorjauksen tason valitsee symbolia luodessaan. Jos QR-koodeja käytetään ympäristössä, missä on suuri riski symbolin vaurioitumiselle, on suositeltavaa valita 30% virheenkorjaustaso. [40, s. 63]

QR-koodi voidaan jakaa myös osiin useaksi luettavaksi koodiksi. Useat symbolit linkkautuvat toisiinsa ja yksi symboli voidaan jakaa maksimissaan 16 osaan. Symboli voidaan myös helposti kryptata. Sen luonnissa voidaan myös käyttää laserpolttomenetelmää tai pistemerkintälaitetta. Mustien ja valkoisten värien paikkoja voidaan myös vaihtaa ja symboli voidaan lukea takapuolelta jos symboli on tulostettu läpinäkyvälle materiaalille. [40, s. 64-65]



Kuva 2.7. *QR-koodin rakenne*
[40, s. 66]

Kuvassa 2.7 on esiteltynä QR-koodin rakenne. Tässä matriisisymbolissa solut on jaoteltu neliöittäin. Symboli koostuu toiminnallisista kuvioista ja datakuvioista. Etsintäkuvio (Finder pattern) sijaitsee symbolin kolmessa eri kulmassa. Etsintäkuviosta voidaan tunnistaa sijainti, koko ja kulma. Etsintäkuvion ansiosta symboli voidaan tunnistaa kaikista kulmista (360 astetta). Sijoittautumiskuviot (Alignment pattern) auttavat korjaamaan tehokkaasti epälineaarisia vääristymiä. Keskimmäistä sijoittautumiskuvioiden pistettä käytetään hyväksi tunnistamaan epämuodostumia kuviosta. Yksittäisessä sijoittautumiskuvion pisteessä on valkoisella taustalle eristettynä musta piste, mikä helpottaa sen keskipisteen löytämisessä. Musta-valkoinen ajastuskuvio (Timing pattern) auttaa jokaisen yksittäisen solun löytämisessä. Ajastuskuviota käytetään apuna virheenkorjauksessa, kun solun keskipisteen löytämisessä on jotain ongelmaa sen ollessa taittunut tai symboli on jollain tapaa epäsuorassa. Hiljainen alue (Quiet Zone) on välttämätön QR-koodin lukemiseksi. Se tekee symbolin erottamisesta muusta kuvasta helpompaa. Harmaa alue symbolissa kuvastaa datakuviota (Data Area). Data on koodattu binäärisinä numeroina (0 ja 1) soluihin pohjautuen koodauksen eri sääntöihin. Binääriset numerot muutetaan mustiksi ja valkoisiksi soluiksi ja sen jälkeen ne järjestetään oikeille paikoille. Data alueeseen kirjattu data sisältää myös Reed-Solomon koodeja ja virheenkorjaukseen liittyviä ominaisuuksia.

2.4.2 Kameran käyttö verkkoselaimella

HTML5-standardi mahdollistaa pääsyn erilaisiin sovellusrajapintoihin verkkoselaimesta. W3C:n verkkosivuilla on HTML5-standardin tarkat määrittelyt, jotka on tehty eri selaintoimijoiden yhteistyönä. Kirjoittamisen hetkellä uusin standardi tunnetaan nimellä HTML 5.2, mihin liittyvät uusimmat suositukset on annettu 17 joulukuuta 2017 [19]. Eri verkkoselaintoimittajat kuten Google, Mozilla ja Apple pyrkivät toteuttamaan määrittelyn mukaisia toiminnallisuuksia omiin tuotteisiinsa. Eri selaimet ovat hieman eri vaiheissa standardin mukaisten toiminnallisuuksien toteuttamisen kanssa. Esimerkiksi tässä työssä käytetty HTML5 standardin mukainen MediaDevices.getUserMedia()-metodi on tullut mukaan sovellusrajapintaan eri verkkoselaimilla eri vaiheissa [28]. Firefox-verkkoselaimeen tämä toiminto tuli mukaan versiossa 36, joka on julkaistu helmikuussa 2015 [1] ja Chrome-verkkoselaimelle tuki tuli versiossa 52, joka on julkaistu heinäkuussa 2016 [41]. Kyseisestä rajapinnasta on olemassa myös käytöstä poistettu vanha sovellusrajapinta. Tässä luvussa keskitytään HTML5-standardin mukaiseen median (äänen ja kuvan) käyttöön verkkoselaimen kautta. Luvussa sivuutetaan vanhat Flash- ja Silverlight-plugineihin pohjautuvat ratkaisut.

Moderneissa HTML5-standardia tukevissa verkkoselaimissa on mahdollisuus käyttää laitteeseen kytkettyjä medialaitteita MediaStream-ohjelmarajapinnan (API) kautta. Tämän ohjelmarajapinnan kautta voidaan pyytää verkkoselaimen ulkopuoliselta taustajärjestelmältä lokaalia mediaa kuten audiota, valokuvia tai videota [24]. Mediaa hyödyntääkseen sovelluksen on pyydettävä verkkoselaimen kautta lupaa käyttäjältä [24]. Kameran puuttuessa tai luvan puuttuessa viritetään poikkeus [24]. Jotkin verkkoselaimet kuten Chrome vaativat myös suojattua https-yhteyttä (vaihtoehtoisesti localhost) median hyödyntämiseksi verkkosovelluksessa [7]. Myös Firefox-verkkoselain selain vaatii suojattua ympäristöä getUserMedia()-metodin käyttämiseksi, muussa tapauksessa navigator.mediaDevices-ohjelmarajapinta on tyypiltään Undefined, eikä sitä voida käyttää [28].

Verkkoselaimelta voidaan pyytää MediaStream-sovellusrajapinnan kautta kuvavirtaa tai valokuvia taustajärjestelmältä. MediaDevices-ohjelmarajapinnan enumerateDevices()-metodilla voidaan pyytää verkkoselaimesta listaa saatavilla olevista median input- ja output-laitteista. Listassa voi olla mikrofoneja, kameroita, kuulokemikrofonilaitteita ja muilla laitteilla. Kyseinen metodi on Promise-pohjainen ja paluuviestinä on lista MediaDeviceInfo-olioita [24, 27]. MediaDeviceInfo-oliossa olevaa deviceId:tä voidaan käyttää media-pyynnöissä tietyn medialaitteen käyttämiseksi [24, 27, 28].

Valokuvan ottamista voidaan pyytää MediaStreamTrack-olion kautta. MediaStreamTrack-olio kuvastaa yksittäistä median lähdettä käyttäjäagentilla. Valokuvaa voidaan pyytää MediaStreamTrack-olion takePhoto()-metodin avulla, joka palauttaa Blob-olion tai grabFrame()-metodilla, joka palauttaa ImageBitmap-olion [30]. Blob-olio sisältää kuvan tavujoukkona (byte sequence) pienin kirjaimin ASCII-koodattuna. Blob-oliot ovat sarjallistuva olioita. Niitä voidaan sarjallistaa ja sarjallisuus voidaan purkaa [12]. ImageBitmap-

olio on bittikarttagrafiikkaa, joka voidaan piirtää kuvapohjalle ilman kohtuutonta latenssia. Täsmällinen arvio siitä, mikä on kohtuuton latenssi, jää yleensä selaintoteuttajan määriteltäväksi. Jos bittikartta luetaan I/O:n kautta on latenssi luultavasti kohtuutonta, mutta GPU:lta tai RAM-muistilta luettaessa latenssi on yleensä hyväksyttävissä rajoissa [18]. ImageBitmap-oliot ovat myös sarjallistuvia, mutta myös siirrettäviä eri prosessien välillä (transferable) [18].

Jatkuvan (stream) videokuvan ja ääninauhan saamiseksi verkkoselaimen sovellukseen voidaan käyttää jo aiemmin mainittua MediaDevices.getUserMedia()-metodia. Metodi kysyy lupaa medialaitteen käyttämiseksi käyttäjältä. Tämän jälkeen metodi palauttaa MediaStream-olion, jonka kautta pääsee käsiksi kuva- ja äänivirtaan. Kuva- ja äänivirtaan päästään käsiksi vastaavasti kuin valokuvan ottamisessa eli metodi palauttaa listan MediaStreamTrack-olioita [29]. MediaStream-olion kautta voi kysyä kuva- tai äänivirran (stream) tilasta, sekä tilata ilmoituksia (event), kun uusi MediaStreamTrack lisätään tai poistetaan [29]. Lisäksi on useita metodeja, joilla päästään käsiksi nauhoitettuihin video- ja äänitallenteisiin. MediaStreamTrack olion kind-attribuutti on asetettu arvoon "audio", jos kyseessä on äänitallenteen osa ja arvoon "video", jos kyseessä on videotallenteen osa [29]. getUserMedia()-metodille voidaan antaa ’constraint’-parametrilla tieto siitä, halutaanko audiota vai videota vai molempia. Parametrilla voidaan myös vaikuttaa esimerkiksi saatavan videotallenteen resoluutioon medialaitteeseen sen ’deviceId’-avainta käyttäen ja vaikuttaa esimerkiksi kuvaussuuntaan ’facingMode’-avainta käyttäen, jos kyseessä on älypuhelin, jossa on etu- ja takakamerat [28].

Kameran zoomaukseen ja tarkennukseen voidaan myös vaikuttaa MediaDevices-ohjelmarajapinnan kautta. MediaStreamTrack-olion kautta päästään käsiksi kameran ominaisuuksiin ja siltä voidaan pyytää tietoja kameralaitteiston kyvykkyyksistä. MediaDevices-rajapinnan getSupportedConstraints()-metodilla voidaan kysyä selaimelta yleisesti tietoja saatavilla olevien laitteiden ominaisuuksista [25]. MediaStreamTrack-olion getCapabilities()-metodilla voidaan tarkemmin kysyä tietoja tietyn medialaitteen kyvykkyydestä sen ollessa aktiivisena. Paluuarvona saatava MediaTrackCapabilities-olio kertoo mm. kameralaitteen ollessa kyseessä sen saatavilla olevista resoluutioista, zoomitasoista, kuvataajuudesta, tarkennusmoodista ja muista kevyistä kuvausominaisuuksista [25]. MediaStreamTrackin applyConstraints()-metodia voidaan käyttää uusien asetuksien asettamiseen medialaitteelle. getConstraints()-metodilla voidaan kysyä viimeisimpiä appyConstraints()-metodilla laitettuja asetuksia paluuarvon ollessa MediaTrackCapabilities tyyppiä. getSettings()-metodilla voidaan kysyä parhaillaan käytössä olevia asetuksia MediaTrackSettings muodossa. [25]

Kyseiset medialaitteiden säätömahdollisuudet ovat tämän kirjoittamisen aikaan melko huonosti tuettuja. Dokumentaation perusteella ainoastaan Chrome tukee esimerkiksi zoomausta MediaStramTrack-olion ja sen ohjelmarajapinnan kautta. Firefox- ja Safari-verkkoselaimet eivät tällä hetkellä anna tukea kyseiselle operaatiolle Mozillan tarjoaman dokumentaation perusteella [32]. getCapabilities()-metodi ei ole tuettu Firefoxilla eikä Safarilla. Sen sijaan Chrome on tukenut tätä versiosta 66 lähtien [32]. applyConstraints()-

metodi on Mozillan ylläpitämän listan mukaan tuettuna Chromella ja Firefoxilla, mutta ei Safarilla [31]. Chromessa tuki on ollut versiosta 63 ja Firefoxissa versiosta 50 lähtien [31].

3 TOIMENPIDEKORTIN DIGITALISOINTI

Alkuselvityksen jälkeen oli selvää, että toimenpidekortteja lähdetään sähköistämään toteuttamalla verkkoselainpohjainen sovellus. Syinä tähän olivat käytettävissä oleva aika ja resurssit sekä kokemukseni verkkopalveluiden toteuttamisesta, kun taas mobiilisovellukset olisivat olleet uusi aluevaltaus. Verkkoselaimissa on tunnetusti rajoitteita verrattuna natiivisovelluksiin, koska verkkoselaimien tuki eri asioissa eroaa toisistaan huomattavasti. Verkkoselaimen kautta ei myöskään ole mahdollista päästä suoraan käsiksi taustajärjestelmän ohjelmointirajapintoihin (esim. Android API), jotka mahdollistavat esimerkiksi laajemmin RFID-tagien käyttämisen NFC:n kautta, erilaisten antureiden lukemiset ja paremmat kameratoiminnot. Uusi Chromessa oleva kokeellinen WebNFC-ohjelmointirajapinta ei toiminut opiskelijakorttien lukemisessa alkuselvityksen aikana ja sen käyttöönotto vaati erityisosaamista, mikä olisi ollut vaikeaa käyttöönoton kannalta. Opiskelijakortit eivät sisältäneet varsinaista NFC-sirua vaan olivat NfcA MifareClassic-tyyppiä, joka on kyllä RFID:n alalajia, mutta ei varsinaisesti suoraan niitä tiettyjä NFC-tyyppejä, joita WebNFC tukee. Verkkoselaimen rajoitteista huolimatta oli tiedossa, että myös verkkoselaimen kautta voidaan käyttää kameraa ja reaaliaikaisempi yhteys oli tarjolla WebSocket- ja HTTP/2-teknologioiden avulla.

Projektin tavoitteena oli toteuttaa helppokäyttöinen ja verkkoyhteyksiä hyödyntävä sovellus, joka toimisi alustariippumattomasti. Nykypäivänä niin opiskelijoilla kuin opettajilla on lähtökohtaisesti kaikilla mobiililaite ja tietokone saatavilla, joten niiden tarjoamiin mahdollisuuksiin haluttiin tarttua. Älypuhelimien omistajilla on myös liittymissään datapaketit, mikä mahdollistaa sen, että sovelluksen käytöstä ei tule ylimääräisiä kustannuksia. Verkkopalvelimen kautta tarjottavaa verkkoselainpohjaista sovellusta on myös helppo ja nopea päivittää, ilman että tarvitsee hoitaa sitä ulkopuolisen toimijan kautta (Play Store Androidilla, App Store iOS:lla). iOS:lle natiivisovelluksien julkaiseminen App Storen kautta vaatii lisenssimaksujen maksamisen (99 dollaria vuodessa) [6], kun selainsovelluksille ei ole vastaavaa rajoitetta. Sovelluksien julkaisu kahta kanavaa pitkin (Play Store, App Store) vaatii myös pitkää perehtymistä julkaisujärjestelmän käyttöehtoihin ja julkaisuprosessiin, mikä olisi vähentänyt resursseja itse sovelluksen kehityksestä.

Digitalisoitavana oli suuri määrä toimenpidekortteja, jotka erosivat jonkin verran toisistaan, mutta joissa oli myös selkeitä yhteneväisyyksiä. Toimenpidekortit ovat A4-kokoisia papereita, joita opiskelijat kuljettavat mukanaan opintojensa aikana ja palauttavat täytetyt kortit opetuskoordinaattoreille opintojakson loputtua. Opetuskoordinaattorit kirjaavat merkinnät käsin tietojärjestelmiin ja siinä samalla pyrkivät tutkimaan palautettujen kort-

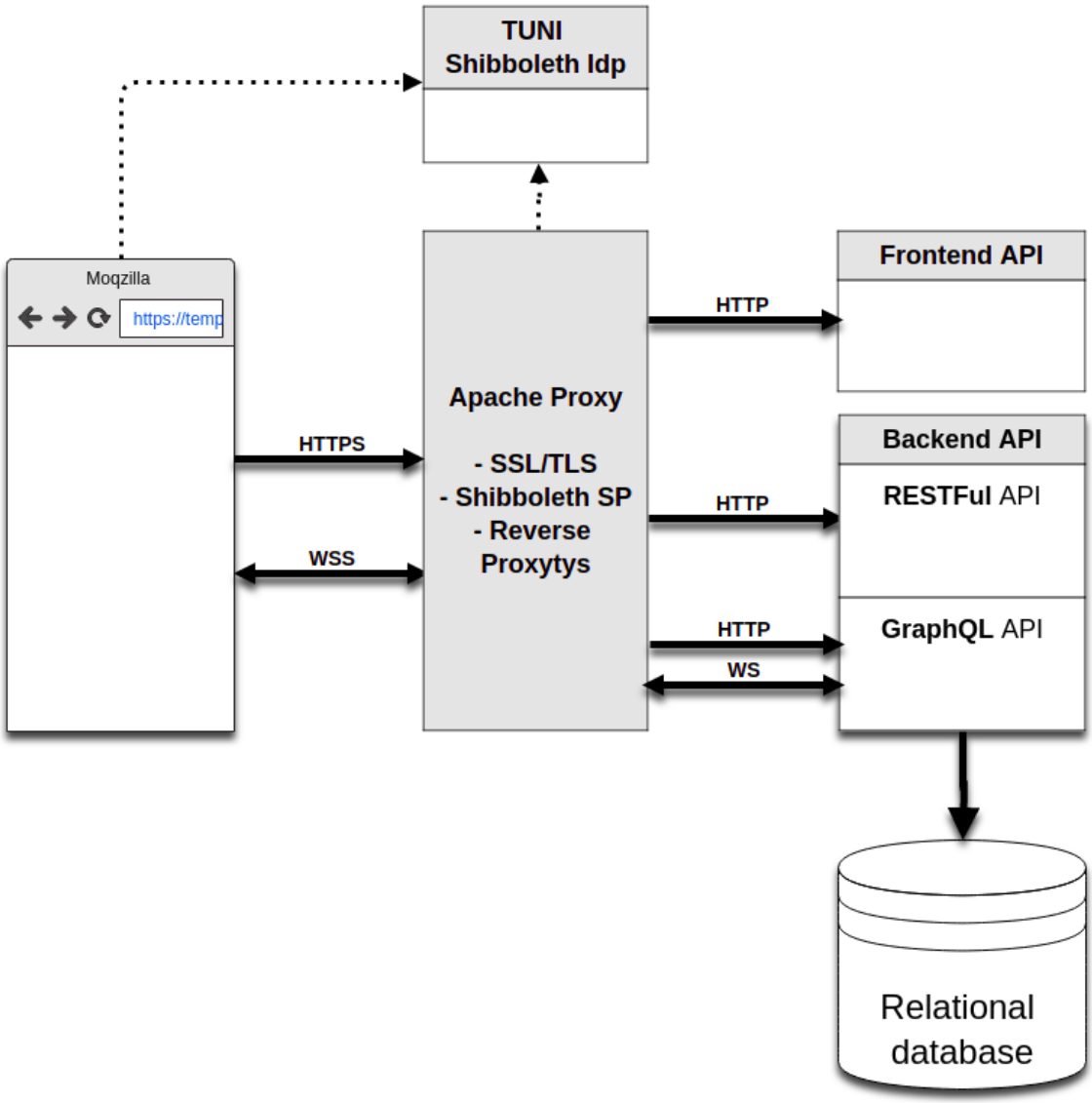
tien oikeellisuutta ja seuraamaan opiskelijoiden etenemistä. Vanhassa mallissa toimenpidekortteja käytettäessä poikkeustapauksia lukuunottamatta opiskelija sai ryhmätyön, seminaarin tms. suoritettuaan päivätyn allekirjoituksen opettajalta. Lääketieteen opinnoissa opiskelijat keräävät näitä merkintöjä hyvin paljon ja seuranta on tarkkaa. Toimenpiteet olivat tyypillisesti kategorisoitu tiettyjen opintojaksojen alle ja opintojaksot kuuluivat kiinteästi tiettyihin opintovuosiin.

Tässä luvussa kuvataan toteutetun tietojärjestelmän keskeisimmät ohjelmistoarkkitehtuurin ratkaisut ja niiden erityispiirteet. Projektin luopputuotteena kehitettiin verkkoselaimessa ajettava SPA-sovellus, joka käyttää REST APIa ja GraphQL-rajapintaa viestien välityksessä keskitettyyn palvelimeen. Palvelin jakautuu Apachen Rerverse Proxyyn, Node.js (HTTP ja WebSocket)-rajapintapalvelimeen ja PostgreSQL-relaatiotietokantaan sekä SPA-sovellusta jakavaan Node.js (HTTP) palvelimeen. Edelle mainitut osat ovat hyvin tyypillisiä nykypäivän uusissa Web-arkkitehtuureihin pohjautuvissa sovelluksissa. Apachen Reverse Proxy huolehtii myös TUNI-organisaation Shibboleth kirjautumisesta, mikä mahdollistaa samojen Tampereen korkeakouluyhteisön organisaatiotunnuksien käyttämisen myös tässä sovelluksessa. Shibboleth on avoimen lähdekoodi Single Sign-on (SSO) identiteetinhallinta ja pääsynhallintapalvelu, joka käyttää taustalla SAML2-standardia [46]. Shibbolethilla oli mahdollista toteuttaa järjestelmään kirjautuminen (autentikointi) ja Apache HTTP Proxy saatiin konfiguroitua vaatimaan aktiivista Shibboleth-istuntoa palvelupyyntöjen tekijältä (sovellus). Istunto todennetaan tässä yhteydessä evästeiden avulla, ja Apache proxy lisää HTTP-kutsujen otsikkotietoihin tiedon pyynnön tehneestä käyttäjästä (osa autorisoinnista). Shibboleth-palvelinosa jakautuu idP- ja SP-osiin, joista tämän työn lopputuotteen palvelin tuoteuttaa SP:n ja keskustelee TUNI-organisaation keskitetyn idP-palvelimen kanssa.

Järjestelmä nimettiin alustavasti Temppukannaksi, mutta sen nimi saattaa muuttua myöhemmässä vaiheessa. Temppukannan pilotointi aloitettiin huhtikuun alussa 2019 ja sen tuloksia käydään tarkemmin läpi luvussa 4.

3.1 Arkkitehtuurikuvaus

Toteutetun SPA-sovelluksen arkkitehtuuri koostuu käyttöliittymästä, joka on toteutettu ReactJS JavaScript-kirjastolla, virtuaalipalvelikoneella olevasta (host machine) Apache HTTP välityspalvelimesta, GraphQL- ja REST -APIsta sekä PostgreSQL-relaatiotietokannasta. Kuvassa 3.1 on visualisoituna arkkitehtuurin pääpiirteinen kuvaus. Joitakin kontittamisiin, varmuuskopiointiin, ylläpidollisiin ja toimintavarmuutta parantaviin ratkaisuihin kuva ei ota kantaa. Työssä on noudatettu REST-arkkitehtuurin käytäntöjä. GraphQL:n puolelta käytössä on vain WebSocket-reaaliaikayhteyksiin liittyvät ominaisuudet (Subscriptions). Koska palvelu noudattaa REST APIn arkkitehtuurin tyyliä, voidaan palvelua kutsuta RESTful-verkkopalveluksi (RESTful Web service). Lyhenteenä RWS.



Kuva 3.1. *Toteutetun järjestelmän arkkitehtuuri*

RESTful-palvelun toteuttaminen oli alussa selkeä valinta palvelun toteuttamiseksi, koska se on nykypäivänä suosittu malli ja hyvin tuettu avoimen lähdekoodin saralla. Kaikki nykyaikaiset verkkoselaimet tukevat XMLHttpRequest (Ajax)-yhteyksiä, mikä mahdollistaa kevyet ja asynkroniset yhteydenotot SPA-sovelluksen ja palvelinrajapinnan välillä.

Kameran käyttö digitaalisessa leimauksessa oli tässä kohtaa yksi vaihtoehdoista erilaisten kertakäyttökoodien lisäksi. Käyttöliittymästä haluttiin tehdä mahdollisimman yksinkertainen, koska sen piti korvata yksinkertainen, mutta työläs vanha toimintamalli. Vanhassa mallissa itse merkintöjen kerääminen kynällä ja paperilla oli yksinkertainen operaatio, mutta koko prosessi paperien suunnittelusta ja tuottamisesta siihen, että merkinnät saadaan itse digitaalisiksi suoritusmerkintöiksi opintorekisteriin on melko raskas. Vanhan prosessin tietoturva on myös melko löysä, koska nimikirjoituksia on helppo väärentää vaikkakin toistuvasta vilpillisyydestä on suuri riski jäädä kiinni. Suorituskuittaus on myös tilaisuus arvioida suorite, paitsi oppijan on mahdollista antaa arvio opettajasta, myös opetta-

jan on mahdollista antaa arvio oppijasta. Tätä mahdollisuutta olisi järkevää käyttää, sillä nykykäytännössä palaute annetaan vasta kuukausia tapahtuman jälkeen ja on todennäköistä että tämä aiheuttaa palautteen vääristymistä. Reaaliaikainen palaute mahdollistaa myös paitsi opetuksen kehittämisen ja tutkimuksen, myös oppijan kehittymisen jakson aikana. Toisaalta reaaliaikaisen seurannan avulla opetusprosessin toteutumista voidaan seurata jatkuvasti.

SPA-sovelluksen kehittäminen oli käytännössä ainoa vaihtoehto responsiivisen käyttöliittymän toteuttamiseksi verkkosovelluksena niin, että sitä ajetaan verkkoselaimesta. Käytettävyyttä parantaa myös se, että sivua ei ladata uudestaan jatkuvasti, kun käyttäjä tekee käyttöliittymässä jonkun verkkoyhteyttä käyttävän operaation. Responsiivisuudella tässä yhteydessä tarkoitetaan sitä, että sovellus skaalautuu hyvin erilaisille päätelaitteille kuten työpöytäympäristöön ja älypuhelimelle.

Kameran käyttö toteutetun verkkosovelluksen osalta on tehokkainta älypuhelimen kameran avulla, koska älypuhelimella on helpompaa ottaa valokuvia kuin kannettavan tietokoneen web-kameralla. Kannettavassa tietokoneessa on myös kamera, mutta se lähtökohtaisesti osoittaa väärään suuntaan ja sillä on epäkäytännöllistä ottaa valokuvia kaukaisemmista kohteista.

Digitaalisen leimauksen toteuttamisessa haluttiin myös kiinnittää huomiota tietoturvaan, koska opintosuoritteiden kerääminen on tietoturvakriittistä. Järjestelmässä on lisäksi henkilöiden kirjautumistunnusten ja nimien käsittelyä. Henkilöitä linkataan tietokantataulujen vierasavaimilla opintosuoritteisiin ja palautteisiin. Tietoturvaongelmiin tartuttiin sallimalla yhteydet vain samasta domainista CORSin avulla käyttämällä SSL/TLS-salausta ja sallimalla yhteydet vain kirjautuneille TUNI-organisaation käyttäjille. TUNI idP:n tarjoamiin HAKA-attribuutteihin ei voinut luottaa, koska attribuutit eivät eritelleet sitä, missä tiedekunnassa ja tutkinto-ohjelmassa henkilö on henkilökunnan jäsen tai opiskelijana. Attribuuteista saatiin vain selville vain se, että kirjautunut henkilö on kategorisoitu organisaation sisällä esim. opiskelijaksi tai henkilökunnan jäseneksi. Tämän takia autorisointi piti rakentaa itse järjestelmään ja käyttöoikeuksia piti pystyä säätämään sisäisesti toteutetussa järjestelmässä. Toteutetun tietojärjestelmän tietoturvaa parannettiin myös lisäämällä jäljitettävyyttä. Jokaiseen annettuu opinsuoritteeseen tallennetaan tieto siitä, kuka suoritteen on alunperin antanut ja kuka sitä on muokannut viimeksi.

Toteutetun järjestelmän uniikkina hienoutena on läsnäolomerkintöjen keräämisesessä käytettävä QR-koodipohjainen läsnäolioiden keräysmekanismi. Opettaja pystyy näyttämään saman sovelluksen kautta QR-koodia ja käyttöliittymän tarjoamien kameraominaisuuksien avulla koodi voidaan lukea ja sen kautta opiskelija voi liittyä mukaan oppimistapahtumaan läsnäolevaksi. Tämä on erityisen hyödyllinen tilanteissa, joissa esimerkiksi luennolla on paikalla 150 läsnäolijaa. Vanhalla mekanismilla paperilappuja kierrätettiin ja opiskelijat allekirjoituksella leimasivat itsensä paikalle. Paperin kierrättäminen kesti jopa 60 minuutia ja häiritsi opetusta. Joissakin tapauksissa opiskelijat joutuvat silloin tällöin jäämään oppimistapahtuman jälkeen vielä antamaan allekirjoitusta tai vaihtoehtoisesti pyytämään opettajalta allekirjoitusta toimenpidekorttiin. Järjestelmän tarjoavaan

QR-koodiin on toteutettu myös eräs tietoturvaa lisäävä toiminto. QR-koodi vaihtuu aikaintervallein, eikä sitä pysty hyödyntämään muulla kuin tarjotulla käyttöliittymällä ja autentikoituneena TUNI-organisaation tunnuksilla. QR-koodilla leimaamisesta lisää aliluvussa 3.6.

3.2 Rajapintatoteutukset

Järjestelmään toteutettiin CRUD (create, read, update ja delete)-sovellusrajapinnat noudattamalla REST-arkkitehtuurimallia. Järjestelmään syntyi siis HTTP-protollan TCP-pyyntöjä käsittelevä sovellusrajapinta, jota tässä yhteydessä voidaan kutsua REST APIksi. Toteutus tehtiin Node.js-ajoympäristön päälle Express.js-moduulin avustamana. Node.js oli itselle Djangon ohella tuttu, mutta kokemukseni oli, että Node.js:llä on vahvempi tuki niin itse V8 JavaScript-ajoympäristön kehityksen osalta kuin avoimen lähdekoodin tarjoamien moduulien osalta. Node.js-ajoympäristö ajaa palvelupyyntöjä yhdellä säikeellä asynkronisesti, mikä mahdollistaa monen yhtäaikaisen pyynnön käsittelyn tehokkaasti. Asynkronisessa Node.js-ympäristössä monia pyyntöjä voidaan palvella yhtäaikaisesti, mutta synkronisten koodiosien ajamista vain yhtä kerrallaan. Kun yhden pyynnön osalta jäädään odottamaan esimerkiksi tietokantakutsun vastausta, voidaan sillä aikaa palvella toista pyyntöä. Synkroninen ajoympäristö pistäisi jonossa olevat pyynnöt odottamaan jonon ensimmäisenä olevan pyynnön käsittelyn valmistumista, jotta se voisi alkaa käsittelemään seuraavana jonossa olevaa pyyntöä. Synkronisessa ajoympäristössä voidaan toki kasvattaa käytössä olevien instanssien määrää, mutta se tarkoittaa myös sitä että isäntäkoneen prosessorista tulisi löytyä enemmän ytimiä (core). Synkroninen ajoympäristö esimerkiksi Djangolla toteutettuna ei olisi välttämättä näkynyt pienellä käyttäjämäärällä mitenkään, mutta paremmasta skaalautuvuudesta ei ole mitään haittaa. Kuorman kasvaessa voidaan asynkronisia instanssejakin lisätä aivan vastaavasti kuin synkronisiakin, mutta sille ei tule niin nopeasti tarvetta palvelupyyntömäärien kasvaessa.

Toteutetussa järjestelmässä on myös GraphQL-rajapinta. Järjestelmän suunnittelun alkuvaiheessa oli vielä epäselvää miten digitaalinen leimaus toteutetaan, joten siinä vaiheessa ei ollut vielä suunnitteilla reaaliaikayhteydet WebSocket- tai HTTP/2-protokollia käyttäen. Myöhemmässä vaiheessa kun idea vaihtuvasta QR-koodista keksittiin, niin vaihtoehtoja etsittiin ja GraphQL Subscriptien käyttö vaikutti helpommalta vaihtoehdolta kuin WebScokettien käyttö suoraan. GraphQL:ssä on yksi abstraktiokerros välissä ja avoimen lähdekoodin moduulit vaikuttivat helppokäyttöisiltä ja hyvin tuetuilta. Käyttämäni GraphQL Subscriptionit käyttävät taustalla WebSocketteja, mutta se ei ole GraphQL:n määritelmän määrittämä yhteysmuoto, vaan se olisi voinut olla toteutettu myös HTTP/2:lla. Työn tekemisen aikaan Node.js:n uusin LTS-versio oli 10.13.0 (Dubnium), mutta tässä versiossa ei ollut vielä natiivisti tukea HTTP/2:lle. WebSocketit olivat sen takia selkeä valinta.

Koko rajapinta olisi voitu toteuttaa pelkästään GraphQL-rajapintana ilman RESTiä. Mutta koska se lisättiin järjestelmään jälkikäteen, niin se olisi vaatinut huomattavaa lisätyötä. Päätin siis poimia GraphQL:stä vain tuon yhden ominaisuuden ja hoitaa CRUD:n REST

APIn kautta.

3.2.1 RESTful API

Järjestelmään on toteutettu REST API, joten arkkitehtuuria voidaan kutsua RESTfuliksi. Sovellus käyttää Express.js-moduulia, mikä on erittäin suosittu APIn toteuttamiseksi Node.js ympäristössä. HTTP-pyynnöillä REST APIn kautta palvelin tarjoaa CRUDin tietokannalle, mikä tarjoaa hallitusti pääsyn kaikkiin tietokannan malleihin. APIn kontrollerit (engl. Controller) toteutin JavaScriptillä ja Objection.js-moduulin ominaisuuksia höydyntäen. Objection.js on ORM (Object-relational mapping) JavaScript-ohjelmointikielelle ja sen ja Knex-kirjaston avulla toteutin tietokantaoperaatiot PostgreSQL-relaatiotietokantaan. Objection.js ORM:lla oli mahdollista myös määritellä tietokantataulujen mallit (engl. Model), jotka sisältävät myös logiikkaa liittyen palvelupyyntöjen validointiin tietokannan skeemaa vasten. Objectionin taustalla käyttämä Knex-moduuli tekee myös Objectionilla tehdyille SQL-kyselylausekkeille sanitaatiot, mikä suojaa SQL-injektioita vastaan. Knex on SQL-lausekkeiden rakentamisessa ja tietokantayhteyksien muodostamisessa käytettävä apukirjasto, jota Objection ORM käyttää riippuvuutenaan. ORMiin oli mahdollista määritellä myös relaatioita toisiin tauluihin ja määriteltyjen relaatioiden avulla on helppoa suorittaa alikyselyitä ja hakea tiettyyn olioon liittyvää dataa muista tauluista.

Knex-kirjastolla sai toteutettua helposti tietokantaskeeman luonnin, migraatiot ja halutut populoinnit. Kaikki nämä olisi voinut toteuttaa vaihtoehtoisesti myös raaoilla tietokantalausekkeilla noudattamalla SQL-kyselykieltä, mutta kirjaston tarjoaman abstraktion avulla niistä sai helpommin luettavamman ja samalla tuen useammalle eri tietokannalle. Knex-moduuli tarjosi myös automaation migraatiotiedostojen hallinnalle ja versionnille.

3.2.2 GraphQL Subscription-rajapinta

Järjestelmän samaan palvelininstanssiin toteutettiin myös GraphQL-rajapinta, mikä tarjoaa hallitusti ulospäin muutaman Subscription-rajapinnan. Tämän toteutukseen käytetään Apollo-organisaation GraphQL-kirjastoja. Subscription-yhteyden avaaminen tapahtuu HTTP-pyynnöllä, jonka jälkeen palvelin pyytää asiakasta jatkamaan yhteydenpitoa WebSocket yhteydellä. Protokollan vaihdon palvelin neuvoo tekemään ’Connection: Upgrade’ ja ’Upgrade: websocket’ otsikkotietoja käyttäen. SPA-sovelluksessa oleva React-komponentti hoitaa kaiken tämän automaattisesti taustalla abstraktiokerroksen takana. Myös käyttöliittymään valikoin Apollo-organisaation kirjastot, koska oletin niiden toimivan luotettavimmin palvelimella olevien kirjastojen kanssa.

Subscription-rajapinnan metodilla voidaan pyytää tilaus uusista suoraistuntojen kuten luentojen kertakäyttökoodeista suoraistunnon avainta käyttäen ja vastauksien viestit voidaan muuttaa luettaviksi QR-koodeiksi. Toisella Subscription-metodilla voidaan tilata tiedot uusista suoraistuntoon liittyneistä käyttäjistä istunnon avainta käyttäen. Istuntoon liit-

tyminen tapahtuu REST APIin kohdistuvalla pyynnöllä, mutta palvelin generoi taustalla livessioiden kertakäyttökoodit automaattisesti aikaintervallien välein. Tietokantaan määritetyt tietokantatriggerit reagoivat uusiin livessioihin liittymisiin ja kertakäyttökoodien luonteihin. Tietokantatriggerit aiheuttavat tapahtumia (Event) Node.js palvelimelle, joka on asetettu kuuntelemaan näitä tiettyjä tapahtumia. Tapahtumat käsitellään ja tiedot uusista tapahtumista kirjoitetaan niihin WebSocketteihin, jotka ovat tilanneet juuri siihen suoraistuntoon liittyviä tapahtumia.

Tietoturvasyistä GraphQL:ään määritetyt mallit eivät ole identtisiä Objection ORMiin määriteltyjen kanssa ja GraphQL-rajapinta palauttaa vain minimaaliset tiedot käyttöliittymälle. GraphQL:n mallit pystytään määrittämään erikseen ohjelman sivun 15 osoittamalla tavalla ohjelmassa 2.1.

Apollon Reactille tarjoama komponentti pitää automaattisesti huolen yhteyden uudelleenmuodostamisesta (reconnect) yhteyden katkettua. Tämä on sovelluksen kannalta melko kriittistä, koska kertakäyttökoodien sovelluskohteessa käytetään reaaliaikayhteyttä. Suoraistunnon yksi kertakäyttökoodi on voimassa vain muutaman sekunnin ajan ja yhteyden katkeaminen huomaamatta ilman virheilmoituksia voisi aiheuttaa käyttäjille käytettävyyteen liittyvää turhautumista.

3.3 HTTP-pyyntöjen autentikointi TUNI-organisaation tunnuksilla

Tuotetun sovelluksen haluttiin käyttävän alusta alkaen TUNI-organisaation omaa kirjautumispalvelua. Vaatimuksen taustalla oli ajatus siitä, ettei uuteen järjestelmään tarvitse luoda omaa identiteetinhallintaa ja omaa kirjautumislogiikkaa. Toinen ajatus oli se, että saadaan rajoitettua pääsy vain TUNI-organisaation henkilöille, jolloin organisaation ulkopuolisilla henkilöillä ei ole pääsyä järjestelmään. Kolmas vaatimuksen taustalla olevista syistä liittyy käytettävyyteen siinä mielessä, että sen käyttäjien ei tarvitse luoda uusia tunnuksia juuri tätä uutta järjestelmää varten.

TUNI-organisaatiossa käytetään Shibboleth-nimistä identiteetin- ja pääsynhallintajärjestelmää, joka on yleisesti käytössä myös muissa Suomen korkeakouluissa Haka-kirjautumisena. Palvelinkoneelle tuli asentaa Shibboleth SP-palveluprosessi, joka keskustelee taustalla TUNIn idP-palvelimen kanssa. SP:n pystyttäminen vaati konfigurointia ja se piti hyväksyttää idP:llä. SP-asennuksen jälkeen Apache HTTP Proxy laitettiin keskustelemaan Shibboleth SP:n kanssa. Kirjautumisen jälkeen käyttäjä tarjoaa evästeet (cookies) verkkoselaimelta palvelupyynnön mukana palvelimelle ja Apache HTTP Proxy Shibboleth SP:n avustamana selvittää keksin avulla, kuka evästeen tarjonnut henkilö on. Kaikki pyynnöt järjestelmän palvelurajapinnoille menevät edellä mainitun Apache HTTP Proxyn kautta, jolloin voimassa oleva kirjautumistieto tulee aina palvelinrajapinnoille pyyntöjen mukana. Kirjautumistieto on sisällytetty Haka-attribuutteina HTTP-viestien otsikkotietoihin. Haluttuihin attribuutteihin voidaan vaikuttaa Shibboleth SP:ltä, mutta täs-

sä vaiheessa attribuuteista tarvittiin vain ’uid’, ’edupersonprincipalname’ ja ’email’. ’uid’ ja ’edupersonprincipalname’ olivat käyttäjän pysyvästi yksilöiviä attribuutteja, mutta ’email’-attribuuttiin ei voinut luottaa, koska sähköpostit voivat vaihtua esimerkiksi henkilön vaihtaessa sukunimeään.

Autentikoitumattoman henkilön saapuessa URLin kautta toteutetun järjestelmän palveluun, hänet ohjataan TUNI idP:n kirjautumissivulle, josta hänet ohjataan takaisin sovellukseen. TUNI idP asettaa kirjautumissivullaan käyttäjän verkkosselaimeen Shibboleth-istuntoon liittyvän evästeen, jota verkkoselain tarjoaa verkkopalveluille pyyntöjen mukana. Käyttäjän kirjautuessa ulos uloskirjautuminen täytyi hoitaa kaksivaiheisena, sillä käyttäjä ei kirjaudu ulos idP:ltä, vaikka hänet kirjataan ulos SP:ltä. Tästä johtuen sovellus ohjaa käyttäjän ensin SP:n uloskirjautumislinkin avulla sovelluksesta pois, josta hänet jälleen uudelleenohjataan idP:n uloskirjautumissivulle.

3.4 WebSocket-yhteyksien autentikointi

HTTP:n tilattomaan luoteeseen ja REST APIin toteutukseen sopi hyvin evästepohjainen autentikointi. Aina jos palvelimelta halutaan jotain dataa tai tietokantaa halutaan muokata, niin pyynnön mukana pitää toimittaa validi eväste, josta saadaan selville pyynnön tehnyt käyttäjä. WebSocket-yhteydet eivät kuitenkaan ole tilattomia, eikä TCP-yhteyttä lähtökohtaisesti katkaista kuin erikseen pyydettäessä.

Apollo-organisaation tarjoaman ’apollo-link-ws’ npm-paketin WebSocketLink tarjosi mahdollisuuden antaa ’options.connectionParams’-avaimella JSON-olion, jolla pystyi määrittämään mitä tahansa parametreja toimitettavaksi palvelimelle yhteyden mukana. GraphQL clientiin määritellään WebSocketLink-tyyppinen olio, jota Subscription-yhteyksissä hyödynnetään. Palvelimella pääsi näihin yhteysparametreihin käsiksi jokaisen yhteyden osalta ja siten oli mahdollista hyödyntää parametreja käyttäjän tunnistuksessa.

Tässä kohtaa tulivat kyseeseen erilaiset token-pohjaiset autentikointimenetelmät. TUNI-kirjautuminen haluttiin pitää ensisijaisena tunnistusmuotona, joten token luovutetaan käyttäjälle vasta TUNI-kirjautumisen jälkeen. Tokenia ei haluttu tallentaa palvelimelle tietoturvasyistä ja koska sille ei edes ollu tarvetta. JWT (JSON Web Token) sopi tähän sovelluskohteeseen erinomaisesti, joten sitä lähdettiin testaamaan tässä yhteysmuodossa.

JWT-token toimi erinomaisesti tässä käyttötarkoituksessa. JWT-tokeniin voidaan enkoodata mitä tahansa dataa, mitä voisi pistää JSON-muotoonkin. Tokenin sisältö koodataan base64-muotoon ja allekirjoitetaan palvelimella käyttäen salausavainta. Salausavainta ei luovuteta eteenpäin. JWT-tokenissa on julkinen osa, joka pystytään purkamaan myös asiakassovelluksessa ja tiivisteosa, joka on luotu palvelimen salausavaimella. Kun asiakassovellus antaa tokenin palvelimelle takaisin WebSocket-yhteyden aloituksen yhteydessä, palvelin tarkistaa salausavaimilla yhteyden mukana tulleen JWT:n validiteetin. Käytännössä tämä tarkoittaa sitä, että token on erittäin vaikea väärentää ja sen aitouteen voidaan luottaa.

Sisällytin JWT-tokeniin käyttäjään liittyvän surrogaattiavaimen, joka yksilöi käyttäjän varmasti. Lisäksi, kun käyttäjän Id:hen ei liity mitään henkilökohtaista tietoa, se on käytännössä turvallista luovuttaa palvelimelta eteenpäin. Kyseinen käyttäjän Id luovutetaan käyttäjälle muutenkin muiden palvelinpyyntöjen yhteydessä. Palvelimen purettua koodatun JWT-tokenin se saa käyttäjän Id:n ja sen avulla GraphQL Resolvereihin voidaan syöttää tieto Subscription-yhteyttä käyttävästä käyttäjästä. Epävalidit yhteydenotot voidaan tiputtaa pois.

3.5 Sovelluksen autorisointi

Työssä toteutettuun MVP-järjestelmään touteutettiin autorisointi eri REST APIn tarjoamille päätepisteille. TUNI idP:stä ei ollut mahdollista saada ensimmäiseen versioon tietoa siitä, missä organisaatiossa henkilö on esimerkiksi opiskelija, työntekijä tai opettaja. TUNI idP:ltä saatiin yleisesti tieto siitä, että henkilö on opiskelija tms. jossain organisaation sisällä. Koska järjestelmä tuotettiin lääketieteen ja terveysteknologian tiedekunnalle, olisi autorisoinnissa voinut käyttää tarkempia tietoja henkilön asemasta organisaatoin sisällä. Mutta koska käyttäjiä ei voida Haka-attribuuttien avulla identifioida esimerkiksi lääketieteen tiedekunnassa työskenteleväksi opettajaksi, niin kyseistä tietoa ei voida käyttää autorisoinnissa.

Edellä mainituista syistä johtuen järjestelmään luotiin oma käyttöoikeusluokittelu. Käyttäjäryhmiä tunnistettiin olevan lähtökohtaisesti kolme: opiskelija, opettajatuutori ja opetuskoordinaattori. Opiskelijalla ei ole oikeutta antaa opintosuoritteita, mutta opettajalla on. Opettajat ja opetuskoordinaattorit tunnistettiin käyttäjiksi, joille voitiin antaa lähtökohtaisesti kaikki käyttöoikeudet. Käyttäjärooleja voidaan järjestelmässä jaella vapaasti opettajan tai opetuskoordinaattorin rooleilla. Yhdellä käyttäjällä voi olla useampi yhtäaikainen rooli. Voi olla myös tilanne, että käyttäjällä ei ole ollenkaan rooleja järjestelmässä, mutta käyttäjä on kuitenkin tietokantaan tallennettuna. Tämä on vakiotilanne käyttäjän kirjauduttua TUNI-tunnuksilla järjestelmään ilman, että käyttäjälle on erikseen annettu laajempia käyttöoikeuksia.

Järjestelmän ensimmäisessä versiossa kyseisiä oikeuksia voitiin jaella lähinnä yksilötasolla. Tämä tarkoittaa sitä, että käyttöliittymän kautta täytyy mennä yksi kerrallaan lisäämään oikeuksia halutuille käyttäjille. Tämä oli riittävä järjestelmän pilotointia varten, koska käyttäjiä oli rajoitettu määrä (n. 30 henkilöä). Tässä on ehdottomasti parantamisen varaa. Esimerkiksi jos Haka-attribuuteista saataisii ulos tarkempi tieto henkilön asemasta TUNI-organisaatiossa, voitaisiin käyttäjäoikeuksia jaella automatisoidusti. Vaihtoehtoisesti myöhemmässä vaiheessa voidaan tehdä ominaisuus, jolla jaetaan oikeuksia useammalle henkilölle kerralla.

Teknisesti käyttäjien autoristointi toteutettiin Express.js-välikerroksella. Halutuille autorisoinneille tehtiin omat välikerrokset ja näitä välikerroksia asetettiin REST APIn eri päätepisteille. Esimerkiksi tekemälläni hasSomeRoleMiddleware-välikerroksella tarkistetaan

onko käyttäjällä jokin rooli sisäisesti järjestelmässä. Jos käyttäjällä ei ole rooleja, palautetaan HTTP:n statuskoodi 403.

Käyttöliittymässä on vastaavia logiikoita. Käyttäjälle voidaan paljastaa tiettyjä näkymiä vain, jos käyttäjällä on tietty käyttäjärooli järjestelmässä. Käyttöliittymä saa nämä henkilön omat käyttäjäroolit pyytämällä niitä REST APIlta. Käyttöliittymän reitityksessä (urleihin liittyvät näkymät) on myös rajoitteita niin, että reittiä ei tunnisteta, mikäli käyttäjällä ei ole sen näkymän näkemiseen vaadittuja käyttäjäoikeuksia.

GraphQL Subscription-yhteyksiä pystytään autorisoimaan 'graphql-subscriptions' npm-paketin withFilter-funktiolla. Funktio ajetaan jokaiselle yhteyssocketille ja sen toisena parametrina annettavan filterFn-funktion palauttaman totuusarvon perusteella voidaan vaikuttaa siihen, kirjoitetaanko tietyn asiakkaan WebSockettiin dataa vai ei. Tähän funktioon sain vietyä JWT-tokenin avulla selvitetyn käyttäjäolion. Käyttäjäolion avulla pystyin määrittämään tarkasti mitä dataa ja mitä tapahtumia (event) voidaan asiakassovellukselle toimittaa.

3.6 Opintosuoritteiden digitaalinen leimaus

Toteutettuun järjestelmään voidaan käyttöliittymän avustuksella rakentaa haluttu opintojaksojen tietorakenne. Opintojaksot voidaan liittää vuosikursseihin ja opintojakson alle voidaan määritellä oppimistapahtumia. Oppimistapahtuma on ikään kuin vanhan paperisen toimenpidekortin yksi rivi, johon opiskelija voi pyytää allekirjoitusta opettajalta.

Tietorakenne on rakennettu niin, että oppimistapahtumiin liittyen voidaan antaa suoritemerkintöjä. Suoritemerkintään liittyy aina tieto, kuka merkinnän sai, kuka merkinnän antoi, mihin liittyen se on annettu, hyväksytty/hylätty -tieto ja mahdollinen opettajan antama palaute suorituksesta. Myös opiskelija pystyy antamaan palautetta oppimistapahtumasta suoritemerkinnän saatuaan, ja se on relaatiossa suoritemerkinnän kanssa. Suoritemerkinnät tallennetaan relaatiotietokantaan, joka on varmuuskopioitu tietoturvan ja toimintavarmuuden takaamiseksi.

Suoritemerkintä on joko kokonaissuoritus tai osasuoritus riippuen siitä onko se oppimistapahtumaan suoraan liittyvä vai oppimistapahtuman osatapahtuma. Tietorakenne on rakennettu niin, että oppimistapahtumalla voi olla osatapahtumia, mutta osatapahtuman alle ei pääse lisäämään osatapahtumia. Tähän vedettiin raja helppokäyttöisyyden takaamiseksi. Näitä opintojaksoihin liittyviä oppimistapahtumia päästään muokkaamaan käyttöliittymän kautta käyttötarkoitukseen erityisesti rakennetussa puurakenteessa.

Prosessi on saatu digitalisoitua melko hyvin, sillä opetuskoordinaattori voi luoda opintojaksojen rakenteet jo syksyisin ennen lukuvuoden aloitusta ja toisaalta nämä pysyvät vuodesta toiseen melkolailla samoina. Opetuskoordinaattorin määrittäessä kaikki rakenteet valmiiksi voidaan niitä käyttää heti suoraan arvostelussa suoritteiden keräämisessä. Paperityöltä vältytään, kun missään kohtaa ei tarvitse enää kerätä paperille läsnäolijoiden

tietoja, mistä ne täytyisi vielä digitalisoida manuaalisella työllä. Pilottivaiheessa keväällä 2019 kaikki suoritemerkinnät annetaan vielä vanhaan tapaan paperisina digitaalisen leimauksen lisäksi. Täten saadaan varmistettua, että järjestelmän alkuvaiheen ohjelmistovirheet eivät vaikuta opiskelijoiden etenemiseen opinnoissaan.

Järjestelmä mahdollistaa tehokkaasti suuren läsnäolijamäärän kirjaamisen esimerkiksi luentotilaisuuksissa tai seminaareissa, joissa saattaa olla jopa 150 opiskelijaa. Näissä tilanteissa opettaja voi näyttää tietokoneelta esimerkiksi videoprojektorin kautta sovelluksen tarjoamaa erityistä QR-koodia, jonka opiskelijat voivat lukea sovelluksen tarjoamalla koodinlukijalla. Läsnäolijat saadaan sen avulla tallennettua parhaimmillaan kymmenissä sekunneissa suoraistunnon aloituksesta sen päättämiseen.

3.6.1 Suoritusten kirjaaminen opettajan roolissa

Opettaja voi käyttöliittymästä aloittaa arvoinnin 'Arvointi'-näkymästä. Tästä näkymästä voidaan valinta haluttu oppimistapahtuma, johon suoritemerkintöjä halutaan kirjattavan. Opettaja saa näkymään listan järjestelmän käyttäjistä ja voi valita listalta halutut käyttäjät suoritemerkintöjen saajiksi. Opettajalla on vaihtoehtoina valita käyttäjä manuaalisesti sovelluksen tarjoamalta listalta, lukea kameralla opiskelijakortin viivakoodi tai lukea kameralla opiskelijan profiilissa oleva QR-koodi, jonka opiskelija voi näyttää omalta laitteeltaan tai tulostettuna. Näiden lisäksi vaihtoehtona on käyttää vaihtuvaa QR-koodia, josta lisää aliluvussa 3.6.3.

Opettaja voi antaa joko hyväksytyn tai hylätyn suoritusmerkinnän riippuen siitä, miten opiskelija suoriutuu oppimistapahtumassa. Opettajalla on mahdollisuus antaa opiskelijalle yksilöllistä palautetta käyttöliittymän kautta. Opettaja voi antaa pisteitä Likert-asteikolla 1-5 ja sen lisäksi antaa sanallista palautetta. Sanalliseen palautteeseen voidaan esimerkiksi kirjata, miksi suoritus hylättiin tai mikä suorituksessa meni erityisen hyvin. Varsinkin hylättyjen suoritusten kohdalla on erityisen tärkeää että syy kirjataan ylös, jotta siihen voidaan palata ja syy-seuraussuhde on selvitettävissä jälkikäteen. Opiskelija voi myöhemmin saada samasta opintotapahtumasta uuden merkinnän hyväksytysti, eli järjestelmä ei ota kantaa siihen, miten merkintöjä jaellaan.

3.6.2 Opiskelijan itseleimaus staattisella tunnisteella

Opetuskoordinaattorin käyttäjäroolissa voidaan määritellä oppimistapahtumille pysyvämpiä tulostettavia QR-koodeja, joita opiskelijat voivat erityistilanteissa lukea sovelluksen avulla. Tämä omininaisuus rakennettiin sellaisia erityistilanteita varten, joissa suorituspaikka on jossakin TUNI-organisaation ulkopuolella (toisella paikkakunnalla oleva terveyskeskus antaa merkinnän) ja siellä ei ole TUNI-organisaatioon kuuluvaa opettajaa, joka voisi merkinnän antaa sovelluksen kautta. Tulostettava QR-koodi on A4-muotoinen paperi, joka luodaan REST APIssa käyttöliittymän kautta sitä pyydettäessä. QR-koodi

generoidaan Node.js palvelimella ’puppeteer’ npm-moduulilla, joka mahdollistaa Chromiumin ajamisen headless-tilassa. Palvelimella on HTML-templaateja näiden papereiden generointia varten. Palvelin aukaisee taustalla Chromium-selaimen ikkunattomassa moodissa ja tulostaa sivun PDF-muodossa. Palvelin palauttaa PDF-tiedoston käyttöliittymälle, joka muuttaa tiedoston formaatin käyttöliittymällä esitettävään formaattiin ja tarjoaa PDF-tiedoston tallennusta.

PDF-tiedostoja voidaan myöhemmässä vaiheessa lähettää sähköisesti haluttuihin paikkoihin ja esimerkiksi ulkopaikkakunnalla olevassa terveyskeskuksessa sitä voidaan näyttää joko tulostettuna tai sähköisesti tietokoneen näytöltä. Opiskelija voi lukea koodin käyttäen käyttöliittymän tarjoamaa koodinlukijaa. Kyseinen koodinlukija avaa verkkoselaimen kautta mobiililaitteessa olevaan kameraan yhteyden ja käyttää MediaStreamTrack-olion tarjoamaa kuvavirtaa hyväkseen. Kun kuvavirrasta tunnistetaan QR-koodi, aiheuttaa se tapahtuman käyttöliittymässä, mikä sitten käsitellään. Käyttöliittymä tarkistaa QR-koodin tiedot ja QR-koodin tietojen perusteella suorittaa palvelinpyynnön suorituksen kirjaamiseksi. Kuvassa 3.2 on esitelty, mitä tämän tyyppiseen QR-koodiin sisällytetty. QR-koodiin on sisällytetty kuvassa esitelty query, jossa ’v’ tarkoittaa versionumeroa, ’t’ koodin tyyppiä (static) ja ’staticCodeId’ on koodin yksilöivä tunniste, jota sovelluksen käyttöliittymä käyttää palvelupyyntöä tekiessään. PDF-tiedostossa on näkyvillä tekstinä, mistä oppimistapahtumasta on kyse ja sen voimassaoloaika. Koodi on versioitu, koska kyseisiä tunnisteita voidaan tulostaa ja levittää, mikä tarkoittaa ohjelmiston päivittäessä sitä, että queryä voidaan muuttaa helposti ja säilyttää tuki vanhaan versioon.

v=0&t=static&staticCodeId=b8202064-ece6-43c6-ae76-0f050603fe43

Kuva 3.2. *Staattinen luettava QR-koodi*

Tulostettua staattista suorituskoodia voidaan editoida myös jälkikäteen. Jokin aikaisemmin tulostettu koodi voidaan ottaa pois käytöstä niin haluttaessa ja sen voimassaoloaikoja voidaan vielä muokata. Edellä määritetyistä ominaisuuksista on hyötyä jos esimerkiksi koodi halutaan ottaa pois käytöstä sen hukuttua tai päädyttyä vääriin käsiin tai jos halutaan esimerkiksi jatkaa koodin voimassaoloaikaa jostain syystä.

3.6.3 Opiskelijan itseleimaus vaihtuvalla tunnisteella

Opettajan voi antaa suoritemerkintöjä opiskelijoille suoraistuntoja hyödyntäen. Suoraistunnoissa opettaja voi höydyntää aikaintervallein muuttuvaa ja ajoittain vanhenevaa QR-koodia. Palvelin tarkistaa aikaintervallien välein tietokannassa käynnissä olevat istunnot ja käy läpi kaikki koodit, jotka täytyy uusia. Uusimisen jälkeen opettajalla auki olevaan käyttöliittymään lähetetään GraphQL:n Subscriptinin avulla ja WebSockettia käyttäen uusi koodi. Käyttöliittymässä tapahtuu tämän vuoksi tapahtuma (Event) ja käyttöliittymä reagoi tapahtumaan renderöimällä uuden koodin näkyville. Vaihtuva, kameralla luettava tunniste on saanut vaikutteita tyypillisistä 2FA-menetelmistä, joissa on tyypillisesti vaihtuva koodi, mikä pitää syöttää kirjautumisvaiheessa sovellukselle tietyssä aikaikkunassa. Esimerkiksi Google Authenticator [13, 8] toimi tämän inspiraationa.

Kuva 3.3. *Ajallisesti vaihtuva QR-koodi*

Kuvassa 3.3 on esiteltynä esimerkki ajallisesti vaihtuvasta tunnisteesta. QR-koodi on hyvin vastaavanlainen kuin staattinen tunniste, joka esiteltiin kuvassa 3.2. Tunnisteeseen on sisällytetty hyvin samanlaisia kenttiä, mutta poikkeamiakin on; 'liveSessionCodeId' on kyseisen tunnisteen surrogaattiavain Id ja 'shortHash' on lyhyt noin kymmenen merkin satunnainen (random) merkkijono. Näitä kahta avainta käyttämällä SPA-sovellus voi tehdä palvelinpyynnön ja pyytää liittymistä suoraistuntoon. Molemmat ovat vaadittuja kenttiä. 'shortHash'-kenttä on olemassa, jotta ajallisesti tulevia tunnisteita ei pysty arvaamaan helposti ja tunnisteen arvaamalla liittymään suoraistuntoon. 'liveSessionCodeId' on Increments-tyyppinen tietotyyppi PostgreSQL-tietokannassa, joten sen seuraava tunniste on helppo arvata, jos samanaikaisesti ei ole muita suoraistuntoja käynnissä.

Samanaikaisesti suoraistunnoilla voi olla olemassa useita tunnisteita. Tunnisteita on tyypillisesti voimassa 1-3 kappaletta käynnissä olevaa istuntoa kohti. Tähän on syynä se, että käytettävyys olisi heikkoa, mikäli koodi vanhenisi välittömästi sen vaihduttua opettajan

kuvaruudulta. Opiskelijoiden tuottamat palvelinpyynnöt menevät vielä hetken aikaa läpi sen tunnisteen tiedoilla, joka on vaihtunut pois näkyvistä. Palvelimelle määritetyn algoritmin mukaisesti yksi tunniste on voimassa noin seitsemän sekunnin ajan ja uusi tunniste luodaan noin kaksi sekuntia ennen koodin vanhentumista. Tunnisteen voimassaoloaika on pyritty määrittämään tarpeeksi lyhyeksi, jotta siitä ei ehditä ottamaan valokuvaa ja jakamaan pikaviestimellä eteenpäin henkilölle, joka ei ole itse paikalla. Tunnisteen lyhyellä kestolla pyritään estämään vilppiä. Älypuhelimen kameralla kuvan ottamisessa menee helposti sekunteja ja ison kuvan lähetyksessä pikaviestimen kautta vie huomattavasti kauemmin. Suoralähetykset voivat olla ongelmallisia vaihtuvan tunnisteen tietoturvalle, mutta tunnisteen uusiutumistahtia voidaan edelleen myös nopeuttaa. Suoraistunnot ovat ongelmallisia, koska opiskelija voi suoralähettää videota tunnisteesta toiselle osapuolelle. Suoralähetyksissä on kuitenkin tyypillisesti jonkin verran viivettä sen kuvaamisesta siihen, että vastaanottaja näkee sen internet-yhteyden välityksellä. Uusiutumistahdin nopeuttaminen voi olla kuitenkin ongelma, jos opettaja jakaa vaihtuvaa tunnistetta etäopetuksessa suoratoiston kautta.

Kuva 3.4. *Ajallisesti vaihtuvan tunnisteen käyttöaika visualisointuna*

Kuvassa 3.4 on visualistoituna yhden suoraistunnon tunnisteiden vaihtumisprosessi. Kuvassa juoksee aika vasemmalta oikealle. Uusin tunniste on kuvassa oikealla ja vanhin kuvassa vasemmalla. Kuvan tunnisteista voimassa olevia tunnisteita ovat vihreällä alueella olevat tunnisteet ja vanhentuneet tunnisteet ovat punaisella alueella olevat tunnisteet. Oletuksena käyttöliittymässä näkyvä tunniste on oikealla oleva uusin tunniste. Se voi kuitenkin olle toinen oikealta jos opettajan käyttöliittymä ei ole vielä saanut WebSocketin kautta uutta tunnistetta. Opiskelijat lukevat näitä tunnisteta sovelluksen kautta ja palvelimelle kohdistuvalla palvelupyynnöllä on aikaa saapua käsiteltäväksi vihreään alueen aikana. Palvelupyyntöön liitetään pyynnön saapumisaika välittömästä synkronisena kutsuna ennen minkään muun Express.js-välikerroksen kutsumista. Tämä optimointi on tehty sen takia ettei kenenkään opiskelijan pyyntö jää kauaksi aikaa jonoon odottamaan toisen opiskelijan tuottamien tietokantakutsujen valmistumista. Optimointi mahdollistaa myös sen, että suuri määrä käyttäjiä voivat liittyä samaan suoraistuntoon lähes samanaikaisesti. Jos optimointia ei olisi tehty, niin usea pyyntö epäonnistuisi ja monet opiskelijat

joutuisivat yrittämään usean kerran tunnisteen lukemista ennen onnistumistaan.

3.6.4 Suoritemerkintöjen jäljitettävyys

Järjestelmän käyttämän relaatiotietokannan suoritemerkitöihin liittyviin tietokantatauluihin on määritelty sarakkeet suoritteen antaneen käyttäjän tiedoille ja viimeksi suoritetta päivittäneen käyttäjän tiedoille. Nämä tiedot tallennetaan aina kun suoritetietoja lisätään tai muokataan hallitusti REST API -rajapinnan kautta. Muokkaaja ja lisääjä voivat olla yksi ja sama henkilö ja yleisimmissä tilanteissa näin onkin.

Suoritteen lisääjän ja sen viimeisimmän muokkaajan tiedon tallennuksella kasvatetaan tuotetun järjestelmän tietoturvaa. Opintosuoritteet ovat kriittisiä tietoturvan kannalta varsinkin lääketieteen opetuksessa. Opiskelijat käyvät läpi suuren määrän pakollisia oppimistapahtumia opintojensa aikana ja on tärkeää, että suoritteen antohetkiin voidaan palata myöhemmin tilanteen jälkeen. Lisääjän ja viimeisimmän muokkaajan tallentamisella esimerkiksi se, että asiaan voidaan puuttua jälkikäteen, jos opiskelija onnistuu antamaan itselleen tai kavereilleen suoritusmerkintöjä. Kyseinen vilpillinen operaatio pystyttäisiin tunnistamaan jälkikäteen tallennetuista tiedoista. Kyseinen operaatio voi myös olla mahdollinen ohjelmistovirheen saattuessa. Poikkeamien tunnistamiseksi voidaan järjestelmän jatkokehityksessä kiinnittää huomiota ja opiskelijoiden käyttäjäoikeuksilla annetut suoritusmerkinnät pystytään tunnistamaan yksinkertaisilla tietokantakyselyillä. Lisääjän tietoa ei voida muokata jälkikäteen rajapinnan kautta tehdyillä palvelupyynnöillä.

Vaikka REST API -rajapinnan kautta opintosuoritteiden tallentaminen on hyvin hallittua ja niiden lisääjistä ja viimeisimmistä muokkaajista jääkin jälki, niin Objection.js ORMiin määritellyillä lifecycle-metodeilla ei voida kuitenkaan vaikuttaa tietokannan muokkaamiseen järjetelmän tarjoamien rajapintojen ohi. Tietokantaa voidaan käsitellä millä tahansa muulla tietokanta-asiakasohjelmalla ja niistä operaatioista ei jää mitään jälkeä. Tämän vuoksi on tärkeää, että tietokanta on varmuuskopiotu hyvin ja tietokantatunnukset on pidetty turvallisesti vain tiettyjen tietohallinnon henkilöiden hallussa.

3.6.5 Palautteen kerääminen oppimistapahtumista

Lääketieteen opetuksen kehityksen ja pedagogisen tutkimuksen tueksi järjestelmään kehitettiin erityinen palautteenantomekanismi. Opiskelija voi antaa jokaiseen oppimistapahtumaan liittyen palautetta heti suoritemerkinnän saamisen jälkeen. Palautteessa opiskelija antaa nopean ja yksinkertaisen palautteen Likertin asteikolla väliltä 1-5. Lisäksi opiskelija voi halutessaan antaa myös sanallista palautetta. Oppimistapahtumista kerätty palaute voidaan esittää erilaisina graafisina kuvaajina sovelluksen kautta. Oppimistapahtumanpalautteiden avulla tyytyväisyyttä opetukseen voidaan seurata.

Opettaja voi käyttää kerättyä palautetta oman ammattiosaamisensa kehittämisessä. Opis-

kelijat voivat antaa palautteen heti oppimistapahtuman jälkeen, mikä eroaa vanhasta mallista, jossa palaute kerätään opintojakson lopussa. Opintojakson päättymiseen voi kulua peräti 3 kuukautta siitä, kun opiskelija on suorittanut oppimistapahtuman. Oppimistapahtumista voidaan siis saada huomattavasti nopeammin palautetta. Lisäksi on huomionarvoista, että oppimistapahtuma on tuoreessa muistissa välittömästi oppimistapahtuman jälkeen, 3 kuukauden päästä voi olla vaikeampaa saada yksityiskohtaista rakentavaa ja harhatonta palautetta. Opettaja voi kolmen kuukauden aikana johtaa useampia oppimistapahtumia, joten eri oppimistapahtumatkin voisivat sekoittua keskenään.

Palautteenantomekanismi toimii toteutetussa järjestelmässä myös toiseen suuntaan. Oppimistapahtuman suoritemerkintöjen leimauksien yhteydessä opettaja pystyy antamaan palautteen myös opiskelijan suoriutumisesta. Tässä on käytössä sama Likertin asteikko 1-5, ja opettaja voi antaa myös vapaata palautetta tekstimuodossa. On tärkeää huomata että kyseessä ei ole arvosana, koska arvosanat ovat lääketieteen opinnoissa yleisesti hyväksytty tai hylätty. Järjestelmässä voidaan antaa suoritemerkintöjä nimenomaan hyväksyttyinä tai hylättyinä. Palautenumerolla voidaan pyrkiä motivoimaan opiskelijaa erityisesti, kun hän suoriutuu oppimistapahtuman yhteydessä erityisen hyvin jostain operaatiosta. Palautenumeron sijaan palaute voidaan esittää esimerkiksi kuvana tai ikonina jos palautenumero on 4 tai suurempi.

4 ARVIOINTI JA JATKOKEHITYS

Tämän diplomityön kirjoittamisen aikaan järjestelmään oli saatu toteutettua kaikki pääominaisuudet (core features). Järjestelmässä pystytään luomaan vuosikurssien opintorakenteet ja luomaan oppimistapahtumia. Käyttäjät voivat kirjautua järjestelmään TUNI-kirjautumisen kautta ja käyttäjäoikeuksia voidaan antaa käyttöliittymän kautta. Suoritemerkintöjä pystytään antamaan käyttöliittymästä ja ajallisesti vaihtuva tunniste (QR-koodi) on toteutettu ja sitä on käytetty onnistuneesti pilottikäytössä.

Järjestelmän pilotti alkoi huhtikuun alussa 2019 Tampereen yliopiston lääketieteen ja terveysteknologian tiedekunnassa. Järjestelmän käyttöä testasi 31 henkilöä lääketietieteen tiedekunnasta. Pilottikäyttäjiin kuului kaksi opiskelijaryhmää, opettajia ja opetuskoordinaattori. Ennen pilottia oli sovittu, että jos isoja ongelmia ei ilmene ensimmäisen testiviikon aikana, pilottia voidaan jatkaa vuosikurssin viimeisen opintojakson loppuun asti. Pilotin ensimmäisenä viikkona ei törmätty isoihin teknisiin ongelmiin, joten pilottia pystyttiin jatkamaan ensimmäisen viikon jälkeen. Pilotin aikana löytyneitä vikoja ja ongelmia pyrittiin korjaamaan siinä määrin kuin resurssit pilotin aikana sen sallivat. Pilotin aikana toteutettiin palautekysely, missä 19 henkilöä vastasi sähköiseen Google Forms:lla tehtyyn kyselyyn. Kyselyyn vastanneista neljä ilmoitti olevansa opettajia ja 15 opiskelijoita.

Palautekyselyn kysymykset löytyvät liitteestä A. Kyselyssä pyrittiin kysymään kohderyhmältä yleistä suhtautumista toimenpidekorttien digitalisoimiseen ja yleisestä suhtautumista järjestelmään. Kyselyssä kysyttiin myös tarkentavia kysymyksiä liittyen QR-koodien lukemiseen ja käytettävyyteen. Palautekyselyn tulokset löytyvät liitteestä B. Avointen kysymysten vastaukset on jätetty pois liitteestä.

Palautekyselyn tuloksista voidaan havaita pilotissa olevien käyttäjien suhtautuvan positiivisti toimenpidekorttien digitalisoimiseen. 74 prosentin mielestä toimenpidekorttien sähköistäminen voisi kehittää opetusta viiden prosentin ollessa eri mieltä. 63 prosentin mielestä läsnäolomerkintöjen kerääminen sähköisesti voisi vapauttaa resursseja opetukseen ja muuhun tekemiseen 11 prosenttia ollessa eri mieltä. 84 prosentin nopea palautteen kerääminen heti opetustilanteen jälkeen järjestelmän tarjoamilla tavoilla auttaisi opetuksen kehittämisessä ja yksilöä kehittämään omaa tekemistä. 79 prosentin mielestä Temppukannassa on potentiaalia toimenpidekorttien korvaajaksi. 84 prosentin mielestä käytössä testattu vaihtuva QR-koodi voisi estää vilppiä.

Palautteen perusteella järjestelmässä on tarpeita jatkokehitykselle ja joillekin korjaustoimenpiteille. Palautekyselyyn vastanneista vain 11 prosenttia piti sovellusta hyvin toimiva-

na mobiililaitteella, 26 prosenttia ollessa eri mieltä. 26 prosentin mielestä Temppukannan eri versioissa oli sen käyttöä estäviä ohjelmistovirheitä 47 prosenttia ollessa eri mieltä. Avoimen palautteen perusteella suurimmat ongelmat johtuivat vanhoista ohjelmistoversioista ja viat esiintyivät erityisesti iOS 10 ja iOS 11 -pohjaisissa laitteissa. Erityisesti zoom-toimintoa kaivattiin ja se saatiin palautteen pohjalta osaan laitteista jo pilotin aikana. Zoom-toiminnallisuuden puuttuminen tarkoitti pilotissa käytännössä sitä, että opiskelijan täytyi mennä lähemmäksi opettajan esittämää QR-koodia, jotta sai koodin luettua sovelluksen kautta. Lisäksi QR-koodin lukeminen ei aina toiminut kaikilla käyttäjillä pilotin aikana.

Muut palautekyselyssä mitatut asiat antavat erittäin hajanaisia tuloksia, eikä niistä voida vetää selkeitä johtopäätöksiä. QR-koodin lukemisesta sovelluksen kautta ja sen helppokäyttöisyydestä ei voida vetää selkeitä johtopäätöksiä. Suoritusmerkintöjen keräämisen tehokkuudesta ja nopeudesta saatiin myös erittäin ristiriitaiset tulokset. Tässä yhtenä syynä saattaa olla se, että pilotissa ei kerätty läsnäolomerkintöjä massaluennolta, joilla on jopa 150 henkilöä.

Jatkokehitystä ajatellen moni esitti avoimessa palautteessa toiveen, että käyttöliittymä tarjottaisiin natiivisovelluksena verkkoselainpohjaisuuden sijaan. Lisäksi klikkausten määrää toivotaan saatavan minimoitua ja käyttöliittymää edelleen yksinkertaistettua. Automaatiota toivottaisiin järjestelmään. Automaatiota voitaisiin lisätä integroimalla lukujärjestyksiä ulkopuolista järjestelmistä Temppukantaan ja käyttämällä sieltä saatua yksilöllistä tietoa hyväksi. Esimerkiksi sovellus voisi tarjota opettajalle juuri sen toimenpiteen kirjaamista, jossa hän on oman lukujärjestyksensä mukaan. Profiilikuvan asettaminen olisi yhden palautteenantajan mielestä kiva lisä ja se voisi olla pakollista. Profiilikuvan perusteella opettajat voisivat tunnistaa opiskelijoita ja tietää kenelle antavat palautetta.

Kehittäjän näkökulmasta ja projektin aikana eri sidosryhmiltä tulleiden palautteiden perusteella järjestelmään jää jatkokehitettävää. Zoom-ominaisuutta ei saatu iOS:lla toimimaan vielä, mutta se johtui iOS:ssa ja Safarissa olevista rajoitteista. Safarin tuorempaan versioon puuttunut getCapabilities() on kuitenkin tulossa [36], mutta aikataulusta ei ole tietoa. Ominaisuus on tullut Safarin Techology Preview Release 70 -versioon marraskuussa 2018, joten on oletettavaa, että se on saatavilla yleisesti lähitulevaisuudessa.

Kameraa ei saatu toimimaan iOS 10 -laitteilla HTML5-rajapintojen kautta, eikä sovellusta ole muilla vanhoilla laitteilla kuten Windows Phonella testattu. Voidaan siis olettaa että kameran käyttäminen tunnisteiden lukemisessa ei lähtökohtaisesti toimi testaamattomilla vanhoilla laitteilla. On mietittävä missä määrin tämän ongelman ratkaisemiseen halutaan käyttää resursseja jatkossa, sillä ongelma on luontaisesti vähenevä ja se koskettaa vain pientä osaa käyttäjistä. Ongelma on luontaisesti vähenevä, koska älypuhelimia vaihdetaan suhteellisen nopeaan tahtiin, joten käyttäjillä oleva laitekanta uusiutuu melko nopeasti. Jos ongelma koskettaa vain pientä prosenttiosuutta tai muutamaa henkilöä, voi olla että tulee halvemmaksi hankkia heille uudet laitteet kuin kehittää vanhalle laitekannalle tuki. Jos kameraominaisuuksista tulee merkittävä kynnyskysymys, tulee käytettyjä teknologioita arvoida uudestaan ja miettiä vaihtoehtoja. Voi olla, että verkkoselaimessa on

mahdotonta saada kameran natiiviominaisuuksia toimimaan lähitulevaisuudessa. Mutta jos nykyiset kameran käyttöön perustuvat ratkaisut ovat riittäviä, niin syitä teknologioiden vaihtamiselle ei välttättä ole.

Avoimessa palautteessa toivottiin, että tarjottaisiin natiivisovellusta verkkosivun sijasta. Tähän voidaan ehdottaa ratkaisuksia myös PWA-sovellusta, jolla nykyiseen verkkosovellukseen saadaan natiivinsovelluksen tyyppisiä ominaisuuksia. PWA-sovellukset ovat vielä melko uusia ohjelmistomaailmassa, mutta ne ovat luonnollinen seuraava askel verkkosovellukselle jos halutaan natiiviominaisuuksia. PWA-sovelluksen saa esimerkiksi "asennettua"älypuhelimen sovellusvalikkoon, jolloin osoiterivi jää pois, eikä sitä tarvitse joka kerta kirjoittaa osoiteriville tai valita kirjainmerkeistä. PWA-sovelluksen kautta ovat mahdollisia myös Offline-toiminnallisuudet ja Push Notifikaatiot. Nykyinen SPA-sovellus on kehitetty jo nykyisellään melko responsiiviseksi, joten se voisi tukea hyvin PWA-sovellukseen siirtymistä.

WebNFC:n käyttämistä tässä sovelluskohteessa kannattaa arvioida uudestaan, kun WebNFC:lle tulee verkkoselaimiin parempi tuki. Nykyisin se on Experimental-toiminnallisuus Chromella ja sen käyttöönotto vaatii erityiskikkailua. Lisäksi tulee tarkistaa tukeeko se NfcA MifareClassic -tyyppisiä kortteja, joita opiskelijakortit edustavat.

Integroituminen erilaisiin järjestelmiin saattaa tulla ajankohtaiseksi jossain kohtaa. Sisu-integraatiosta oli puhetta projektin aikana, mutta projektin aikana sitä ei ollut mahdollista toteuttaa. Sisu on yhteisyliopistollinen uusi opinto- ja opiskelijatietojärjestelmä [39], mikä tulee korvaamaan eri yliopistojen omia tietojärjestelmiä tulevaisuudessa. Esteinä integroimiselle tässä vaiheessa olivat mm. niukka aikataulu sekä Sisu-järjestelmän varhainen kehitysvaihe.

Kirjautumisaikaa voidaan pidentää esimerkiksi JWT:llä, eli ottamalla käyttöön toinen autentikoitumismenetelmä TUNI-kirjautumisen rinnalle. Shibboleth-autentikoitumisen sessiolle tarjottiin tietohallinnosta melko niukat tuntimäärät, mutta ottamalla käyttöön joku toissijainen kirjautumismenetelmä tätä voidaan kiertää. On kuitenkin suositeltavaa, että ensisijainen autentikoituminen hoidetaan edelleen Shibbolethilla. Shibboleth-kirjautuminen tarjoaa hyvän tietosuojan ja vähentää työn määrää, kun kirjautumiseen ja tunnuksiin liittyviä toiminnallisuuksia ei tarvitse toteuttaa itse.

Kaksisuuntaista palautteenantoa ja analytiikkaa voidaan kehittää eteenpäin. Niistä saadaan erilaisia graafeja, joista voi olla hyötyä opetuksen kehittämisessä ja opiskelijoiden opintojen etenemisen seurannassa. Palautteen perusteella voidaan myös suorittaa tutkimuksia esimerkiksi vertailemalla usean ryhmän antamia palautteita keskenään. Tästä voi olla hyötyä, jos eri ryhmille toteutetaan opintoihin ja opintojen kehittämiseen liittyviä kokeiluja. Palautteista voidaan myös muodostaa esimerkiksi opettajille portfolioita, joilla voidaan dataan pohjautuen osoittaa erinäisiä asoita. Opiskelijan etenemistä opinnoissa voidaan myös seurata järjestelmän kautta, mikäli käyttöliittymään kehitetään sitä tukevia näkymiä. Järjestelmän tallentamaa dataa voidaan hyödyntää monella eri tapaa. On enemmänkin kysymys siitä, mitä oikeasti tarvitaan ja onko niihin resursseja tarjolla.

Jos opettajan näkymä halutaan yksinkertaistaa mahdollisimman pitkälle, on mahdollista, että koko arvostelumenetelmä käännetään päälaelleen. Tällöin opiskelijat voisivat valita sovelluksesta oppimistapahtuman, johon haluaisivat merkinnän ja opettaja voisi lukea opiskelijan esittämän tunnisteen kameralla. Tässä mallissa opettaja leimaa ja hyväksyy suoritteen älypuhelimen kameraa hyödyntämällä. Malli on matkittu tilanteesta, jossa matkustaja esimerkiksi junassa esittää tarkastajalle (konduktöörille) junalippua ja tarkastaja lukee lipun käytetyksi. Temppukanta voisi siis käyttöliittymän kautta esittää QR-koodia ja muita tunnistetietoja ja opettaja voisi tarkistaa sen ja skannata koodin lukijalla. Kyseinen malli ei toimi massaluennoilla, mutta pienryhmissä se voisi toimia. Applen iOS -laitteilla koetut kameraongelmat tukisivat myös tätä vaihtoehtoa. Ongelman laajuutta saadaan rajattua vähentämällä vaadittujen laitteiden määrää.

5 YHTEENVETO

Tämän opinnäytetyön yhteydessä toteutettiin MVP-toteutus järjestelmästä, jolla voidaan digitalisoida Tampereen yliopiston lääketieteellisissä opinnoissa käytettävät toimenpidekortit. Työssä toteutettiin verkkoselaimella toimiva SPA-sovellus, jolla opettajat voivat antaa suoritemerkintöjä opiskelijoille. Suoritteet menevät keskitettyyn tietokantapalvelimeen REST APIn kautta. Opetuskoordinaattorit voivat tarkastaa suoritemerkintöjä käyttöliittymän kautta ja antaan niiden pohjalta suoritusmerkintöjä opintorekisteriin opiskelijoille. Järjestelmä tukee myös jatkuvan palautteen keräämistä. Oppimistapahtumiin liittyvää kaksisuuntaista palautetta voidaan kerätä kerätä heti oppimistapahtumien jälkeen järjestelmään kehitetyillä tavoilla. Palautetta voidaan siten käyttää opetuksen kehityksen ja pedagogisen tutkimuksen tukena.

Työ osoittaa sen, että toimenpidekorttien digitalisointi on mahdollista ja sen, että toteutetun järjestelmän avulla voidaan vähentää paperityötä. Luvussa 4 käsiteltiin järjestelmäpilotin tuloksia palautekyselyn pohjalta. Pilotti järjestettiin huhtikuun 2019 alusta lähtien. Palautekyselyn tulokset olivat melko positiivisia ja uusi järjestelmä sai pääpiirteittäin hyvän vastaanoton. Parannettavaa jäi erityisesti käyttöliittymätasolle. Erityisesti järjestelmän tarjoama QR-koodi -pohjainen ajallisesti vaihtuva tunniste sai kiitosta ja sen uskottiin estävän vilppiä. QR-koodien lukeminen ei kuitenkaan onnistunut kaikilla laitteilla, erityisesti vanhoilla iOS 10 -pohjaisilla laitteilla. Lisäksi iOS-pohjaisten laitteiden Safari-verkkoselaimessa käyttämä WebKit-versio ei tukenut zoom-ominaisuuden tekemistä, mutta Android-pohjaisiin laitteisiin tämä tuki saatiin tehtyä Chrome-selaimelle.

Järjestelmään kehitettiin uniikkeja tapoja kerätä suoritemerkintöjä digitaalisin menetelmin. Järjestelmä tarjoaa mahdollisuuden kerätä läsnäolomerkinnät käyttämällä nopeasti vaihtuvia QR-koodeja ja lukemalla niitä QR-koodin lukijalla. Opettaja voi esittää käyttöliittymänsä kautta vaihtuvaa QR-koodia opetustilanteessa ja opiskelija voi lukea sen sovellukseen tehdyllä QR-koodin lukijaohjelmalla. QR-koodi vaihtuu opettajan esittämässä näkymässä GraphQL Subscriptioneihin pohjautuen WebSocketien avulla. Järjestelmä tarjoaa myös mahdollisuuden lukea opiskelijakorttien takana olevia viivakoodeja ja opettaja voi lukea käyttäjien profiileissa olevia käyttäjää yksilöiviä QR-koodeja. Lisäksi sovelluksen kautta voidaan lisätä käyttäjiä manuaalisesti arvostelusessioon.

Toinen merkittävä järjestelmään kehitetty opintosuoritteiden keräämistapa on käyttää QR-koodi-pohjaisia staattisia tunnisteita. Tämä menetelmä kehitettiin erityistilanteita varten. Erityistilanteeksi luokiteltiin tilanne, jossa lääketieteen opiskelija menee vieraalle paikkakunnalle harjoitteluun ja siellä ei ole TUNI-organisaation henkilökuntaa, joka voisi suori-

temerkintöjä antaa. Järjestelmä tarjoaa PDF-tiedoston generointia, jossa on sovelluksen kautta luettava QR-koodi. QR-koodilla opiskelija voi saada suoritemerkinnän näissä erityistilanteissa. Staattinen tunniste voidaan ottaa pois käytöstä käyttöliittymän kautta ja niitä voidaan luoda useita kappaleita jokaiselle oppimistapahtumalle.

Projektin alkuselvittelyssä tutkittiin RFID-pohjaista kirjaamistapaa, mutta HTML5 WebNFC -rajapinnat selaimissa eivät olleet vielä tarpeeksi kypsiä idean toteuttamiseksi käytännössä. Alunperin ideana oli toteuttaa sovellus, jolla olisi voitu lukea opiskelijakorteissa olevia RFID-siruja. Kyseinen tapa olisi ollut mahdollista toteuttaa natiiville mobiilisovellukselle, mutta projektissa päädyttiin SPA-sovellukseen käytettävissä oleviin resursseihin ja omaan pohjakokemukseeni pohjautuen.

Järjestelmässä käytetään TUNI-tunnuksia kirjautumisessa. TUNI-organisaation kirjautuminen on toteutettu Shibbolethilla. Shibboleth saatiin melko kivuttomasti osaksi järjestelmää tietohallinnon tarjoamilla Apache HTTP Proxy-konfiguraatioilla. TUNI-tunnusten käyttäminen mahdollisti sen, että sovellukseen itseensä ei tarvinnut kehittää autentikointiin liittyviä menetelmiä. Shibbolethia pystyttiin käyttämään kaikissa HTTP-pyynnöissä, mutta WebSocket yhteyksissä käytetään JWT-tokenia. Autorisointi jouduttiin kuitenkin rakentamaan itse järjestelmään, koska Shibboleth:n kautta ei ollut mahdollista saada luotettavasti tietoa käyttäjän asemasta organisaation sisällä. Käyttäjä pystyttiin tunnistamaan esimerkiksi opiskelijaksi tai opettajaksi TUNI-organisaation sisällä, mutta tarkempaa tietoa ei saatu siitä, missä tiedekunnassa tai mitä henkilö voisi saada tehdä toteutetun järjestelmän sisällä. Tämä oli ongelmallista kun ristiinopiskelu on mahdollista ja käyttäjällä voi olla useita opinto-oikeuksia.

LÄHDELUETTELO

[1] *36.0 Firefox Release*. Mozilla Corporation. 24. helmikuuta 2015. URL: https://www.mozilla.org/en-US/firefox/36.0/releasenotes/ (viitattu 11. 03. 2019).

[2] *A WebSocket client + server for GraphQL subscriptions*. Meteor Development Group Inc. URL: https://github.com/apollographql/subscriptions-transport-ws (viitattu 27. 02. 2019).

[3] *Berners-Lee 'sorry' for slashes*. BBC. 14. lokakuuta 2009. URL: http://news.bbc.co.uk/2/hi/technology/8306631.stm (viitattu 17. 02. 2019).

[4] T. Berners-Lee, R. Fielding ja L. Masinter. *Uniform resource identifier (URI): Generic syntax*. Tekninen raportti. 2004.

[5] S. Buna. *Learning GraphQL and Relay*. Packt Publishing Ltd, 2016.

[6] *Choosing a Membership*. Apple Inc. 2019. URL: https://developer.apple.com/support/compare-memberships/ (viitattu 05. 04. 2019).

[7] *Chrome 47 WebRTC: Media Recording, Secure Origins and Proxy Handling*. Google LLC. 14. tammikuuta 2019. URL: https://developers.google.com/web/updates/2015/10/chrome-47-webrtc (viitattu 11. 03. 2019).

[8] R. Estourgie ja E. Poll. Analysis of Android Authenticators. Tohtorinväitöskirja. BS thesis. Radboud Universiteit Nijmegen, 2013.

[9] I. Fette ja A. Melnikov. *The websocket protocol*. Tekninen raportti. 2011. URL: https://www.rfc-editor.org/rfc/pdfrfc/rfc6455.txt.pdf.

[10] R. T. Fielding ja R. N. Taylor. *Architectural styles and the design of network-based software architectures*. Vol. 7. University of California, Irvine Irvine, USA, 2000.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach ja T. Berners-Lee. *Hypertext transfer protocol–HTTP/1.1*. Tekninen raportti. 1999.

[12] *File API*. W3C. 8. maaliskuuta 2019. URL: https://w3c.github.io/FileAPI/#dfn-Blob (viitattu 11. 03. 2019).

[13] *Google Authenticator Wiki*. Google LLC. 28. maaliskuuta 2019. URL: https://github.com/google/google-authenticator/wiki (viitattu 01. 05. 2019).

[14] *GraphQL Best Practices*. Facebook Inc. URL: https://graphql.org/learn/best-practices/ (viitattu 26. 02. 2019).

[15] *GraphQL, June 2018 Edition*. Facebook Inc. Kesäkuu 2018. URL: https://facebook.github.io/graphql/June2018/ (viitattu 20. 02. 2019).

[16] I. Grigorik. *Optimizing Encoding and Transfer Size of Text-Based Assets*. Google LLC. 22. helmikuuta 2018. URL: https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/optimize-encoding-and-transfer#data_compression_101 (viitattu 30. 04. 2019).

[17] A. Grigoryan. *The Benefits of Server Side Rendering Over Client Side Rendering*. 17. huhtikuuta 2017. URL: https://medium.com/walmartlabs/the-benefits-of-

server-side-rendering-over-client-side-rendering-5d07ff2cefe8 (viitattu 11. 02. 2019).

[18] *HTML*. WHATWG. 8. maaliskuuta 2019. URL: https://html.spec.whatwg.org/multipage/imagebitmap-and-animations.html#imagebitmap (viitattu 11. 03. 2019).

[19] *HTML 5.2 W3C Recommendation*. W3C. 14. joulukuuta 2017. URL: https://www.w3.org/TR/html5/ (viitattu 10. 03. 2019).

[20] *HTTP headers*. Mozilla Corporation. 15. tammikuuta 2019. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers (viitattu 07. 02. 2019).

[21] *Hypertext Transfer Protocol – HTTP/1.1*. The Internet Society. Kesäkuu 1999. URL: https://www.ietf.org/rfc/rfc2616.txt (viitattu 07. 02. 2019).

[22] *Introduction to the server side*. Mozilla Corporation. 25. syyskuuta 2018. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction (viitattu 07. 02. 2019).

[23] M. Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. "O'Reilly Media, Inc.", 2011.

[24] *Media Capture and Streams*. W3C. 3. lokakuuta 2017. URL: https://www.w3.org/TR/mediacapture-streams/ (viitattu 10. 03. 2019).

[25] *Media Capture and Streams*. W3C. 28. maaliskuuta 2019. URL: https://w3c.github.io/mediacapture-main/ (viitattu 03. 04. 2019).

[26] *Media Types*. Internet Assigned Number Authority. 15. tammikuuta 2019. URL: https://www.iana.org/assignments/media-types/media-types.xhtml (viitattu 08. 02. 2019).

[27] *MediaDevices.enumerateDevices()*. Mozilla Corporation. 9. marraskuuta 2018. URL: https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/enumerateDevices (viitattu 12. 03. 2019).

[28] *MediaDevices.getUserMedia()*. Mozilla Corporation. 8. toukokuuta 2019. URL: https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia (viitattu 10. 03. 2019).

[29] *MediaStream*. Mozilla Corporation. 9. marraskuuta 2018. URL: https://developer.mozilla.org/en-US/docs/Web/API/MediaStream (viitattu 12. 03. 2019).

[30] *MediaStream Image Capture*. W3C. 21. kesäkuuta 2017. URL: https://www.w3.org/TR/image-capture/ (viitattu 11. 03. 2019).

[31] *MediaStreamTrack.applyConstraints()*. Mozilla Corporation. 18. maaliskuuta 2019. URL: https://developer.mozilla.org/en-US/docs/Web/API/MediaStreamTrack/applyConstraints (viitattu 03. 04. 2019).

[32] *MediaStreamTrack.getCapabilities()*. Mozilla Corporation. 23. maaliskuuta 2019. URL: https://developer.mozilla.org/en-US/docs/Web/API/MediaStreamTrack/getCapabilities (viitattu 03. 04. 2019).

[33] M. Mikowski ja J. Powell. *Single page web applications: JavaScript end-to-end*. Manning Publications Co., 2013.

[34] *MIME types*. Mozilla Corporation. 10. joulukuuta 2018. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types (viitattu 08. 02. 2019).

[35] M. Muskardin. *Before You Build a PWA You Need a SPA*. 6. kesäkuuta 2017. URL: https://hackernoon.com/before-you-build-a-pwa-you-need-a-spa-e22770a0f31c (viitattu 12. 02. 2019).

[36] *Release Notes for Safari Technology Preview 70*. 14. marraskuuta 2018. URL: https://webkit.org/blog/8496/release-notes-for-safari-technology-preview-70/ (viitattu 24. 04. 2019).

[37] A. Rodriguez. Restful web services: The basics. *IBM developerWorks* 33 (2008).

[38] *Serving over HTTP*. Facebook Inc. URL: https://graphql.org/learn/serving-over-http/ (viitattu 26. 02. 2019).

[39] *SISU opinto- ja opiskelijatietojärjestelmä*. Aalto-yliopisto. 20. maaliskuuta 2019. URL: https://wiki.aalto.fi/pages/viewpage.action?pageId=118689728 (viitattu 01. 05. 2019).

[40] T. J. Soon. QR code. *Synthesis Journal* 2008 (2008), 59–78. URL: https://foxdesignsstudio.com/uploads/pdf/Three_QR_Code.pdf.

[41] *Stable Channel Update*. Google LLC. 20. heinäkuuta 2016. URL: https://chromereleases.googleblog.com/2016/07/stable-channel-update.html (viitattu 11. 03. 2019).

[42] *Subscriptions*. Meteor Development Group Inc. URL: https://www.apollographql.com/docs/apollo-server/features/subscriptions.html (viitattu 27. 02. 2019).

[43] *The application/json Media Type for JavaScript Object Notation (JSON)*. The Internet Society. Heinäkuu 2006. URL: https://www.ietf.org/rfc/rfc4627.txt (viitattu 07. 02. 2019).

[44] *Understanding schema concepts*. Meteor Development Group Inc. URL: https://www.apollographql.com/docs/apollo-server/essentials/schema.html (viitattu 26. 02. 2019).

[45] *Vue.js Server-Side Rendering Guide*. URL: https://ssr.vuejs.org/#what-is-server-side-rendering-ssr (viitattu 11. 02. 2019).

[46] *What's Shibboleth?* The Shibboleth Consortium. URL: https://www.shibboleth.net/index/basic/ (viitattu 07. 04. 2019).

A PALAUTEKYSELYN KYSELYLOMAKE

Temppukannan pilotointikysely

Temppukannan pilotti alkoi huhtikuun alussa 2019. Toistaiseksi sovellus löytyy osoitteesta https://met-temppukanta.rd.tuni.fi

Tällä kyselylläkartoitetaan pilottiryhmän mielipiteitä järjestelmän käytettävyydestä ja sen tuomasta käyttöarvosta. Vastaaminen tähän on vapaaehtoista.

Temppukanta sähköistää toimenpidekortin, mikä vähentää paperityötä. Sähköinen toimenpidekortti on aina mukana, eikä sitä voi hävittää. Lisäksi vilpin mahdollisuus on minimoitu.

Tämä kysely kartoittaa pilottiryhmän mielipiteitä järjestelmän käytettävyydestä ja käyttöarvosta. Kyselyyn vastaaminen on vapaaehtoista, mutta toivottavaa: tuloksia käytetään anonymisoituna opinnäytetyöstä ja vastaaminen edistää järjestelmän jatkokehitystä. Tämänhetkinen versio on MVP toteutus (minimum viable product); se voi sisältää vielä puutteita ja ohjelmistovirheitä, jotka korjataan mahdollisimman pikaisesti havaitsemisen jälkeen. Hanke on toteutettu opinnäytetyönä kehitetty lääketieteellisen tiedekunnan rahoituksella.

Kyselyn tuloksia voidaan käyttää anonymisoidusti Tuomas Ahon diplomityössä.

1. Oma pääasiallinen käyttäjärooli pilotissa
   (a) Opiskelija
   (b) Opettaja/tutori
   (c) Opetuskoordinaattori
2. Toimenpidekorttien sähköistäminen voisi kehittää opetusta (Asteikolla 1 = Täysin eri mieltä, 5 = Täysin samaa mieltä)

   1 2 3 4 5
3. Läsnäolomerkintöjen kerääminen sähköisesti voisi vapauttaa resursseja opetukseen ja muuhun tekemiseen (Asteikolla 1 = Täysin eri mieltä, 5 = Täysin samaa mieltä)

   1 2 3 4 5
4. Vanhat paperiset toimenpidekortit hukkuvat helposti ja opiskelijat unohtavat ottaa niitä mukaan (Asteikolla 1 = Täysin eri mieltä, 5 = Täysin samaa mieltä)

1 2 3 4 5

5. Suoritusmerkintöjen kerääminen on nopeaa Temppukannan tarjoamilla tavoilla (Asteikolla 1 = Täysin eri mieltä, 5 = Täysin samaa mieltä)

   1 2 3 4 5

6. Temppukanta on helppokäyttöinen tai helposti ymmärrettävä (Asteikolla 1 = Täysin eri mieltä, 5 = Täysin samaa mieltä)

   1 2 3 4 5

7. Temppukanta toimii hyvin mobiililaitteella (Asteikolla 1 = Täysin eri mieltä, 5 = Täysin samaa mieltä)

   1 2 3 4 5

8. Temppukannan käyttäminen häiritsee opetusta tai opetuksen seuraamista jos verrataan vanhaan toimintamalliin (Asteikolla 1 = Täysin eri mieltä, 5 = Täysin samaa mieltä)

   1 2 3 4 5

9. Temppukannan livetilan vaihtuva QR-koodi on helppokäyttöinen (Asteikolla 1 = Täysin eri mieltä, 5 = Täysin samaa mieltä)

   1 2 3 4 5

10. Käytössä testattu vaihtuva QR-koodi voisi estää vilppiä (Asteikolla 1 = Täysin eri mieltä, 5 = Täysin samaa mieltä)

    1 2 3 4 5

11. Palautteen kerääminen heti opetustilanteen jälkeen järjestelmän tarjoamalla tavalla auttaisi opetuksen kehittämisessä ja yksilöä kehittämään omaa tekemistä (Asteikolla 1 = Täysin eri mieltä, 5 = Täysin samaa mieltä)

    1 2 3 4 5

12. Pilotissa olleessa Temppukannan versioissa oli ohjelmistovirheitä, jotka estivät kokonaan suoritusmerkintöjen antamisen (Asteikolla 1 = Täysin eri mieltä, 5 = Täysin samaa mieltä)

    1 2 3 4 5

13. Temppukannassa on potentiaalia toimenpidekorttien korvaajaksi (Asteikolla 1 = Täysin eri mieltä, 5 = Täysin samaa mieltä)

    1 2 3 4 5

14. Sovelluksen toimiminen. Valitse seuraavista vaihtoehdoista

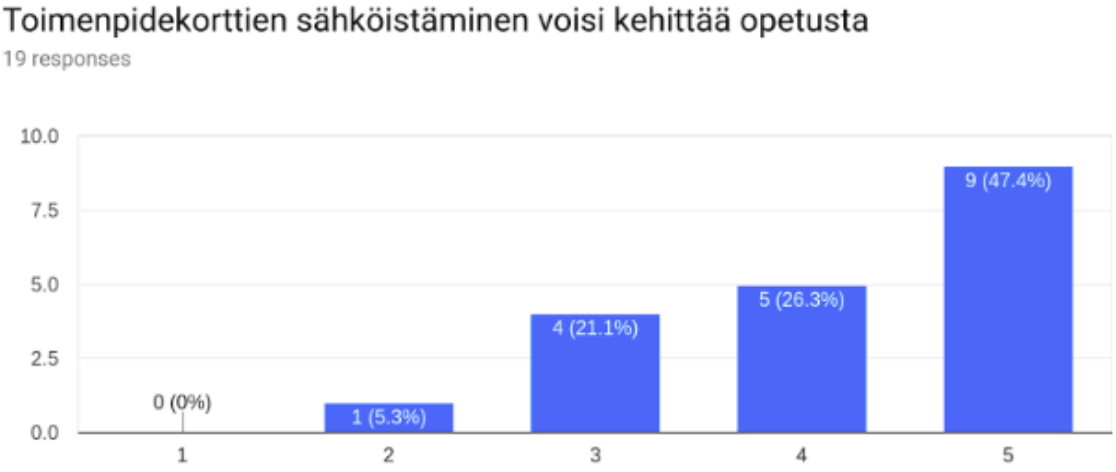
    (a) Sovellus toimi mobiililaitteellani

    (b) En käyttänyt sovellusta mobiililaitteella

    (c) Sovellus ei toiminut laitteessani, jossa on iOS 10

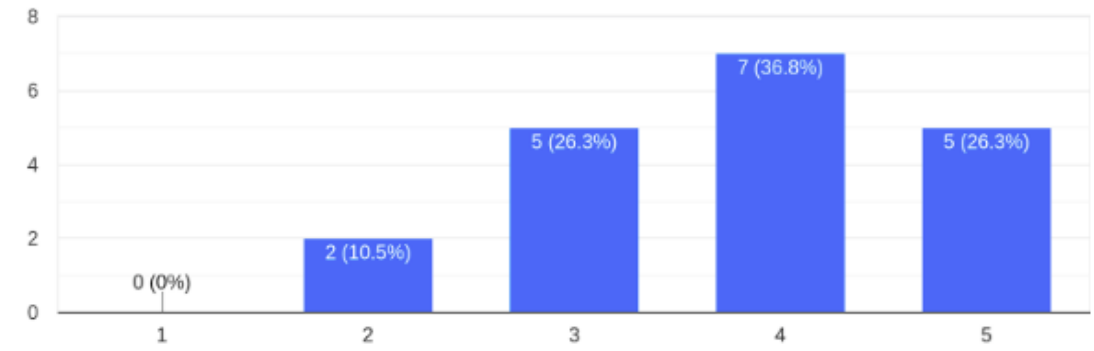
   (d) Sovellus ei toiminut laitteessani, jossa on iOS 11
   (e) Muu, mikä
15. Mielipiteesi mobiililaitteen kameran hyödyntämisestä läsnäolojen merkkaamisessa
16. Havaitsitko pilotin aikana järjestelmässä vakavia puutteita tai vikoja, mitä ei ole vielä korjattu. Mitä järjestelmästä puuttui? Minkä toiminnallisuuden lisäisit järjestelmään?
17. Muu avoin palaute

B PALAUTEKYSELYN VASTAUSJAKAUMAT



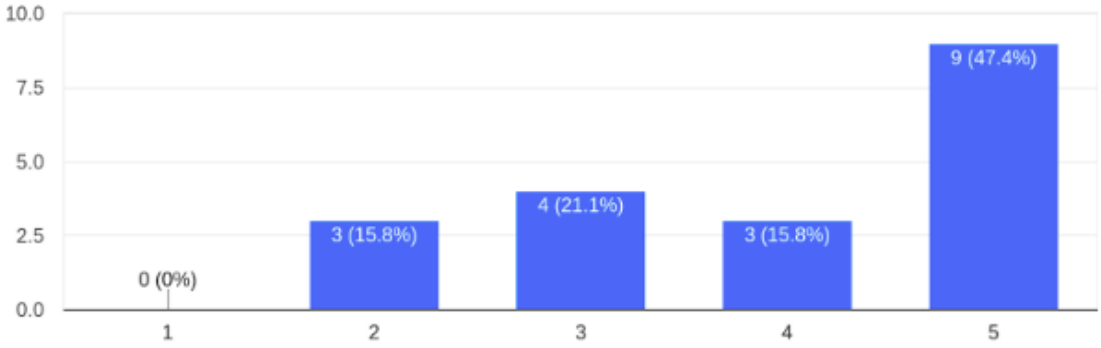
Läsnäolomerkintöjen kerääminen sähköisesti voisi vapauttaa resursseja opetukseen ja muuhun tekemiseen

19 responses



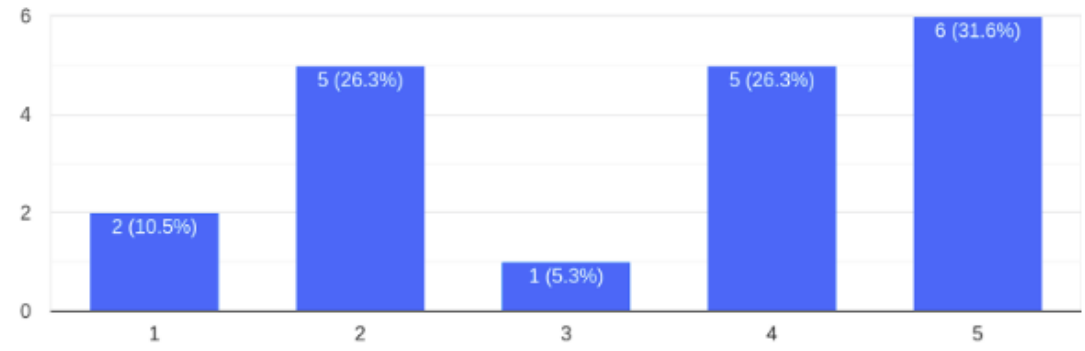
Vanhat paperiset toimenpidekortit hukkuvat helposti ja opiskelijat unohtavat ottaa niitä mukaan

19 responses



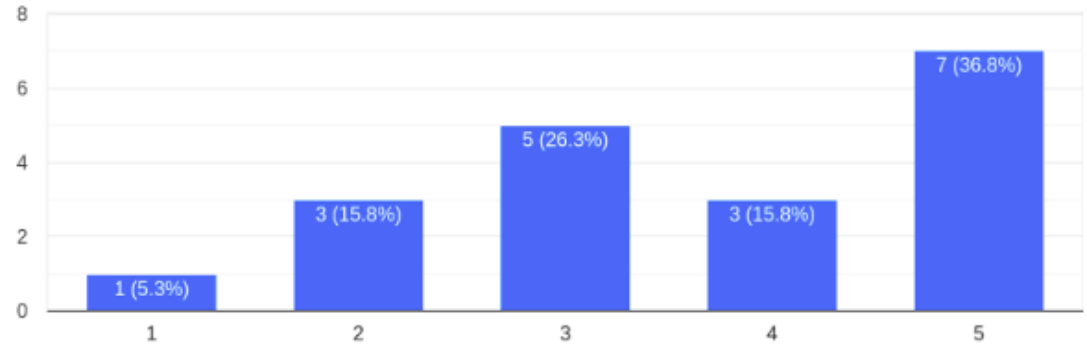
Suoritusmerkintöjen kerääminen on nopeaa Temppukannan tarjoamilla tavoilla

19 responses



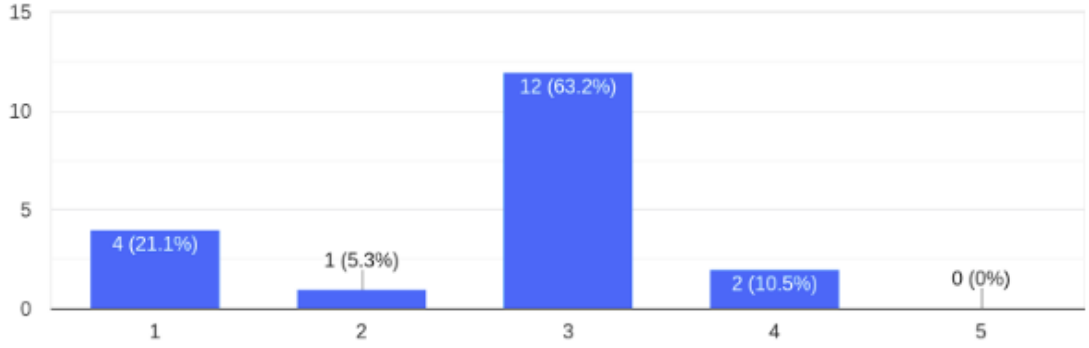
Temppukanta on helppokäyttöinen tai helposti ymmärrettävä

19 responses



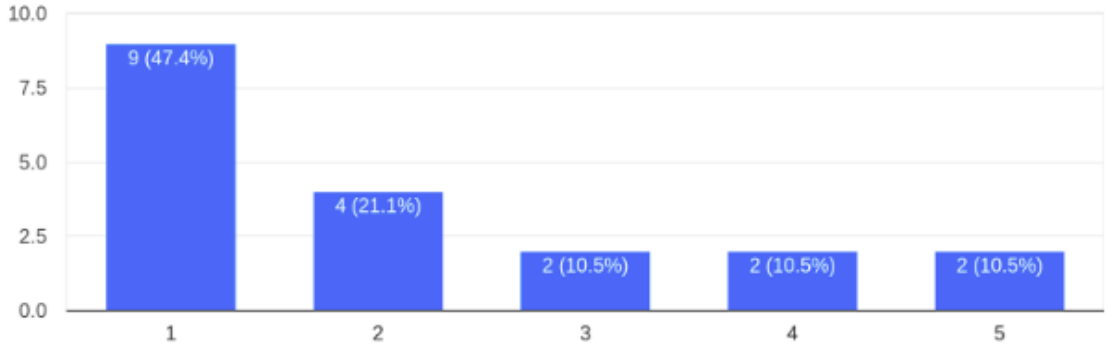
Temppukanta toimii hyvin mobiililaitteella

19 responses



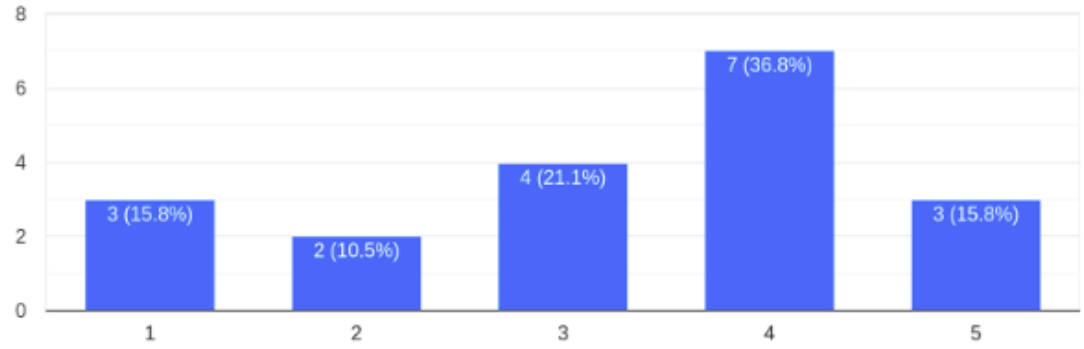
Temppukannan käyttäminen häiritsee opetusta tai opetuksen seuraamista jos verrataan vanhaan toimintamalliin

19 responses



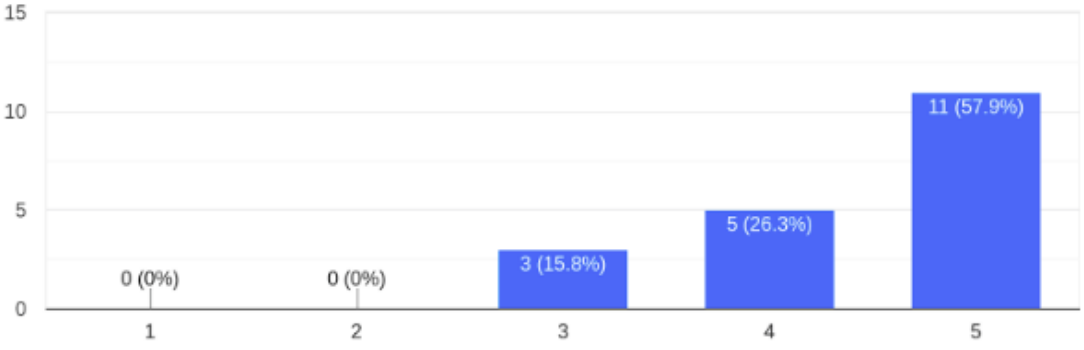
Temppukannan livetilan vaihtuva QR-koodi on helppokäyttöinen

19 responses



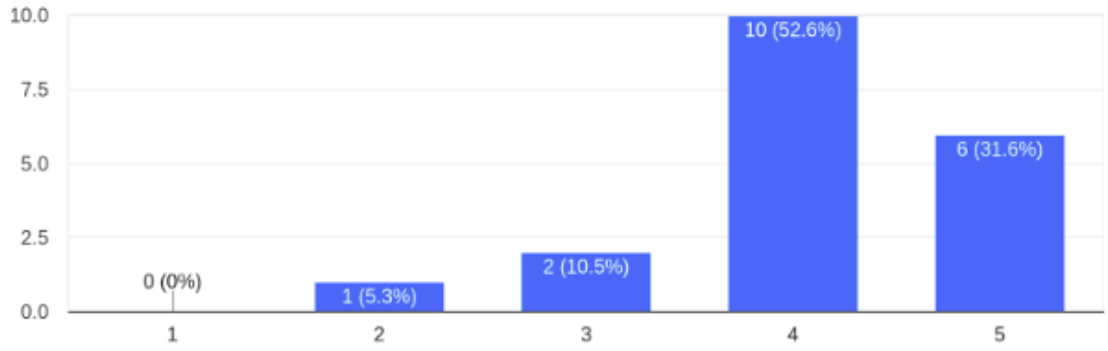
Käytössä testattu vaihtuva QR-koodi voisi estää vilppiä

19 responses



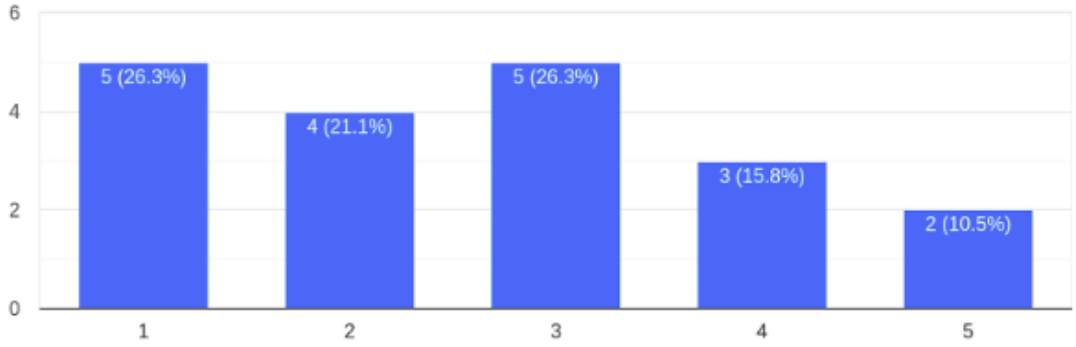
Palautteen kerääminen heti opetustilanteen jälkeen järjestelmän tarjoamalla tavalla auttaisi opetuksen...ja yksilöä kehittämään omaa tekemistä

19 responses



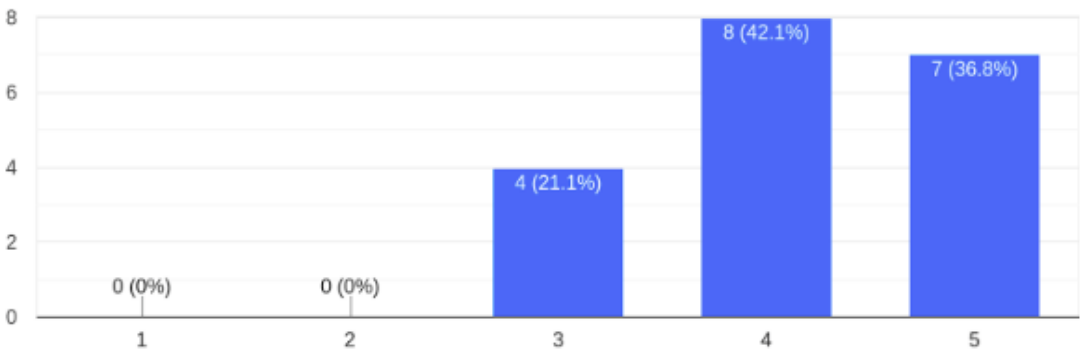
Pilotissa olleessa Temppukannan versioissa oli ohjelmistovirheitä, jotka estivät kokonaan suoritusmerkintöjen antamisen.

19 responses



Temppukannassa on potentiaalia toimenpidekorttien korvaajaksi

19 responses



Sovelluksen toimiminen. Valitse seuraavista vaihtoehdoista

19 responses