Samuel Marie-Louise

DATAMIGRAATIO SOVELLUSOHJELMOINTIRAJAPINTAA HYÖDYNTÄEN

Opinnäytetyö
Tietojenkäsittelyn koulutus

2019

Kaakkois-Suomen
ammattikorkeakoulu

Kaakkois-Suomen ammattikorkeakoulu

Tekijä/Tekijät	Tutkinto	Aika
Samuel Marie-Louise	Tradenomi (AMK)	Syyskuu 2019
Opinnäytetyön nimi		48 sivua
Datamigraatio sovellusohjelmointirajapintaa hyödyntäen		
Toimeksiantaja		
Mikkelin kaupungin liikelaitos Otavia		
Ohjaaja		
Janne Turunen		

Tiivistelmä

Opinnäytetyön tarkoituksena oli yksinkertaistaa ja sujuvoittaa koulutusten ilmoittautumisjärjestelmää. Lyhytkursseille ja vapaatavoitteisiin opintoihin ilmoittaudutaan Eventilla Oy:n tapahtumailmoittautumisalustassa. Nykyisin tiedot siirretään Eventillalta Pyramukseen käsin kopioituina. Toimintatapa ei ole järkevä eikä mielekäs, vaikka Eventilla tarjoaa asiakkailleen hyvin kehitetyt API:t. Oppinäytetyönä tehdyn projektin tavoitteena oli saada asiakkaiden ilmoittautumistiedot siirrettyä Pyramukseen API-tekniikan avulla ja tuoda samalla API-ominaisuus Pyramuksen avoimeen lähdekoodiin.

API:n toteuttamiseen käytetiin REST-arkkitehtuurimallia. Se hyödyntää HTTP-protokollaa, TCP/IP-tiedonsiirtotekniikkaa ja JSON- tai XML-tietoformaattia. REST perustuu kuuteen rajoitteeseen, jotka takaavat arkkitehtuurin luotettavuuden ja tekevät siitä tehokkaan työkalun datan siirtämiseen verkkopalvelujen välillä. Datan siirtämiseksi tuotettiin kaksi prototyyppitasoista sovellusta. Eventilla Pyramus Proxy -sovelluksen avulla haettiin tietoja tuottajalta ja muotoiltiin niitä JSON-tietomalliksi. Otaviassa tavoiteltiin kuitenkin monipuolisempaa API:n toteutusta. Tämän vuoksi kehitetiin Dynaaminen Pyramus API Client (DPAC) -sovellus, joka kykenee hakemaan dataa riippumatta palveluntarjoajasta. Tekniikka perustuu dynaamiseen API:n määritekartoitukseen.

Pyramus-ohjelmistoa täytyy kehittää jatkuvasti, jotta se täyttää kasvavan liiketoiminnan vaatimukset. API:n lisääminen on suuri askel tulevaisuuteen, koska tekniikkaa voidaan hyödyntää eri tilanteissa. Kaikkien alkuvaiheen ongelmien ratkaisujen jälkeen DPAC:lla haettiin tietoa sekä testipalvelimelta että Eventillalta. Verkkosovelluspalvelujen erikoiset oikeuttamiset ja tietojen jäsentämiset onnistuvat dynaamisten moduulien ansiosta. Palautteiden mukaan DPAC-sovellus täyttää tehtävänsä. Sen vuoksi suositellaan prototyypin jatkokehittämistä ja sen integroimista Pyramukseen.

Opinnäytetyö toteutettiin keväällä 2019 Otavian ohjelmistokehitystiimille. Tiimin tehtävänä on muun muassa kehittää Scrum-ketterällä menetelmällä Pyramus-oppilaitoshallintajärjestelmää.

Asiasanat

rajapinnat, prototyypit, tieto, avoin tieto, JSON, JavaScript, REST, API, Node.Js, verkkosovelluspalvelut

South-Eastern Finland University of Applied Sciences

Author (authors)	Degree	Time
Samuel Marie-Louise	Bachelor of Business Administration	September 2019

Thesis title

Data migration using Application Programming Interface

48 pages

Commissioned by

Mikkelin kaupungin liikelaitos Otavia

Supervisor

Janne Turunen

Abstract

The aim of this study was to simplify and streamline the registration system for Liberal Adult Education (LAE). Eventilla Oy's event enrollment platform is used for enrolling for LAE courses. Now information is transferred manually from Eventilla into Pyramus Learning Management System (LMS), while Eventilla is API capable. The objective of the study was to have the customers' registration data transferred to Pyramus using API technology. This thesis was completed in the spring 2019 for Otavia's Software Development team. The main task of the team is to develop Pyramus LMS using Scrum agile methodology.

The REST architectural model was used to implement the API. Data transfer was accomplished by two application prototypes: Eventilla Pyramus Proxy retrieves data from Eventilla and formats them accordingly. However, a more advanced and flexible API was preferred. Therefore, a Dynamic Pyramus API Client (DPAC) was built, which is capable of retrieving data regardless of the data provider. Its functionality is based on the API's attribute mapping.

Pyramus needs to be developed constantly to meet the requirements of a growing business. API also extends Pyramus capabilities. After solving the initial problems, DPAC was used to retrieve data from the Labs as well from Eventilla servers. Web service authorizations and specific data parsing were made possible through dynamic modules. According to feedback, DPAC meets its purpose. Therefore, its further development and integration to Pyramus is recommended.

Keywords

interfaces, prototypes, data, open data, JSON, JavaScript, REST, API, Node.Js, Web Services

SISÄLLYS

1 JOHDANTO

Tässä opinnäytetyössä toimeksiantajana on Mikkelin kaupungin liikelaitos Otavia. Otavia-nimi otettiin käyttöön 1.12.2018, ja se korvasi aiemman, vuodesta 2006 alkaen käytössä olleen nimen Otavan Opiston liikelaitos. Vuonna 1892 perustettu Otavan Opiston kansanopisto jatkaa toimintansa liikelaitoksen alla. Otaviassa työskentelee noin 70 vakituisessa työsuhteessa olevaa työntekijää. Liikelaitosta johtaa johtaja-rehtori ja hänen apunaan vararehtori. Liikelaitoksella on tiimiorganisaatio. Jokaisella tiimillä on tiiminvetäjä. Liikelaitoksessa työskentelee 12 tiimiä, muun muassa ohjelmistokehitystiimi, sisältötiimi, hallintotiimi ja infratiimi. Otavan Opiston kansanopisto järjestää muun muassa aikuislukio- ja peruskouluopintoja internaattimuotoisesti. Maahanmuuttajille suunnattua alkuvaiheen opetusta sekä perusopetusta ja vapaan sivistystyön koulutuksena erilaisia kursseja ja lukuvuoden mittaisia vapaatavoitteisia opintoja. Liikelaitos Otavia puolestaan järjestää mm. nettilukio- ja nettiperuskouluopetusta ja on mukana useissa eri hankkeissa.

Otavian seitsemän hengen ohjelmistokehitystiimi kehittää Pyramus-oppilaitoshallintajärjestelmää ja Muikku-oppimisympäristöä. Tiimiin kuuluu ohjelmoijia, www-suunnittelijoita ja ohjelmistosuunnittelijoita. Tiiminvetäjä vastaa ohjelmistokehityksestä ja tuoteomistaja valvoo tuotteiden kehitystä ja laatua. Koulutussihteeri on Pyramuksen tehokäyttäjä ja aktiivinen sidosryhmän jäsen.

Opinnäytetyön tavoitteena on vähentää käsin tehtävää työtä tietojen siirtämiseksi kahden palvelun välillä. Tehtävän yleinen tarkoitus on hakea tapahtumat ja ilmoittautumiset Eventillalta API-tekniikkaa (Application Programming Interface eli sovellusohjelmointirajapinta) käyttäen REST-arkitehtuurissa (Representational State Transfer) ja siirtää ne Pyramukseen. Pyramuksen API:n kehittäminen on kaksivaiheinen: 1) luodaan sovellus nimeltä Eventilla Pyramus Proxy (EPP), joka hakee dataa tuottajalta, 2) etsitään ratkaisua dynaamiseen API-määritykseen siten, että Pyramuksen toiminta on riippumaton tietojen tuottajasta. Luodaan sitä varten Dynaaminen Pyramus API Client (DPAC) -sovelluksen. Lisäksi sovellusten kehittämiseksi rakennetaan testiympäristö, jossa kaksi palvelinta täyttää verkkosovelluspalvelujen roolia.

Työn rajaus

Toimeksiannon toteutumiseksi rajataan API:n tutkimus koskemaan Pyramus-verkkosovellusta ja Eventilla-verkkopalvelua. Tiedon siirtäminen rajataan yhteen suuntaan, jossa Pyramus lähettää hakupyynnön. Haetaan tapahtumien ja osallistujien ennalta määrätyt tiedot, jotka vastaavat Pyramuksen tietokantarakennetta. Tiedon rajaaminen helpottaa sekä EPP:n ja DPAC:n kehittämistä että mahdollisen API-sovelluksen integroiminen Pyramukseen.

Tehtävän suorituksessa ei oteta kantaa siihen, miten tietoja käsitellään tai tallennetaan palvelinalustalle, vaan tutkitaan ainoastaan palveluntarjoajan API-määrityksiä. Sieltä haetaan vain testidataa, joka ei sisällä Otavian asiakkaiden henkilötietoja. Raportissa esitetään geneerisen ja dynaamisen rajapinnan käyttömalli. Lisäksi rajataan tietojen hakeminen Eventillalta ja tietojen havainnollistaminen prototyyppitasoiseen ohjelmistoon ilman Pyramukseen siirtämistä.

2 OTAVIAN TOIMINTAYMPÄRISTÖN NYKYTILANNE

Pyramus-oppilaitoshallintajärjestelmä on alun perin Otavan Opiston kehittämä ohjelmisto. Sitä kehitettiin Borland Delphi -kehitysympäristössä. Ohjelmiston työasemaversion käyttö alkoi Otavan Opistolla 1990-luvulla. Uusi ohjelmistokehitystiimi kehitti Pyramusta perin pohjin ja sovelluksen saatavuus kasvoi selainpohjaisuuden ansiosta. Pyramukseen tallennetaan opiskelijoiden tietoja, suoritettuja kursseja tai opintojaksoja ja muuta kouluhistoriaa. Sovelluksen toimintamalli pohjautuu projektinhallintatyyppiin, jossa kurssit ja moduulit ovat ikään kuin projektiosia (Seravo 2013; GitHub 2019). Sovelluksen nykyistä versiota kehitetään avoimella lähdekoodilla. Kehitystyö on jatkuva prosessi, jossa käytetään ketterää Scrum-menetelmää. Tämä tarkoittaa, että sovellusta kehitetään pienin askelin iteroiden, jatkuvalla tarkastelulla ja korjaus- ja parantamistoimenpiteillä.

Eventilla Oy -palveluntarjoaja

Eventilla Oy on suomalainen yritys, joka tarjoaa palveluna tapahtumahallintaa. Järjestelmän tavoite on automatisoida ja vähentää tapahtumajärjestäjien käsin

tehtävää työtä. Se kerää tapahtuman osallistujien ilmoittautumiset dynaamisesti luotavalla lomakkeella. Asiakas säästyy laitehankinnoilta ja ylläpidolta, sillä tiedot tallentuvat palveluntarjoajan palvelimelle. Osallistuja saa sähköpostitse tai puhelimeensa lipun, jota voi käyttää sisääntulotarkistuksessa. ZEFin (2014) mukaan sisältömarkkinointi paranee Eventilla-palvelun avulla. Eventillan rajapinnan osalta yhteyshenkilönä toimi yrityksen teknologiajohtaja (CTO).

Nykyinen tiedonsiirtomalli verkkopalvelun ja verkkosovelluksen kesken

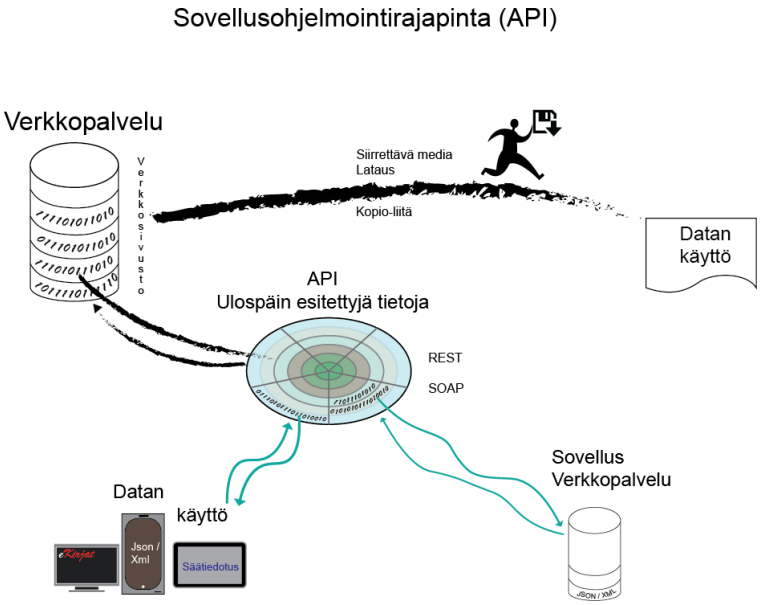
Keskustelussa koulutussihteerin kanssa selvisi, että asiakkaat täyttävät Eventillan tekemän ilmoittautumislomakkeen. Kurssien ja osallistujien tiedot siirretään oppilaitoshallintajärjestelmään. Ongelma ilmenee siinä, että Eventilla on ulkopuolinen verkkopalvelu, jolla ei ole minkäänlaista kanavaa Otavian oppilaitoshallintajärjestelmään. Koulutussihteerillä ei ole muuta vaihtoehtoa tietojen saamiseksi Pyramukseen kuin käsin kopioiminen. Toimintatapa ei ole järkevä eikä mielekäs. Tietojen siirtäminen käsin on raskasta ja hidasta. Toisekseen virhemahdollisuudet ovat korkeat ja tietosuojan ja tietoturvan taso on huono, koska esimerkiksi koulutussihteeri näkee tietoja, jotka eivät ole välttämättä hänelle kuuluvia.

3 SOVELLUSOHJELMOINTIRAJAPINTA JA VERKKOSOVELLUSPALVELUT

Tämä luku aloitetaan määrittelemällä sovellusohjelmointirajapinta, joka on tärkein työkalu käsin tehtävästä työstä eroon pääsemiseksi tietojen siirrossa Eventillalta Pyramukseen. Lisäksi esitetään sen käyttöalue, menetelmän edut ja mahdolliset haitat. Sovellusohjelmointirajapinta määritelmä päätetään tietoturvallisuuteen liittyviin pohdintoihin. Jatketaan alaluvuissa verkkosovelluspalvelun parissa, jossa esitellään sen muodostamia osia, jotka ovat TCP/IP-tiedonsiirtotekniikka, HTTP-protokolla, JSON- ja XML-tietoformaatit. Alaluvussa esitellään tietoliikenteen portin käsitteet, verkkosovellusten resurssien suojausmenetelmiä ja HTTP-pyyntöjen vastauskoodeja. Lopuksi esitellään REST-arkkitehtuurityyliä.

3.1 Ohjelmointirajapinnan määritelmät

Rajapintoja on monenlaisia. Ne mahdollistavat tietojen vaihtamisen laitteiden tai sovellusten kesken. Tietojen vaihto voi olla myös laitteen ja ihmisen välillä (Sanastokeskus s.a.). Sen sijaan sovellusten välistä rajapintaa kutsutaan sovellusohjelmointirajapinnaksi (API), joka kuvailee järjestelmän sisäisten tietojen esittäminen ulospäin jollakin liittymällä. Kuva 1 havainnollistaa sekä API:a että perinteistä datan siirtämistä. Liittymällä on määritteitä, joilla ulkopuolinen saa ladata juuri ne tiedot, jotka verkkopalvelu asettaa käytettäväksi paljastamatta, miten tietolähdejärjestelmä on rakennettu tai mitä se pitää sisällään.



Kuva 1. Sisäisten tietojen näyttäminen ulospäin

Cenno (2014) toteaa, että rajapinta helpottaa tietojärjestelemien rakentamista ja kehittämistä ilman kalliita integrointiprojekteja. Periaatteessa se mahdollistaa sen, että hallinnoitavat järjestelmät eivät vaadi isoja muutoksia, jotta ne saadaan kytkettyä toisiinsa, mutta käytännössä työn määrä riippuu siitä, miten API on implementoitu. API voi olla avoin tai suljettu. Se voi olla rajoitetusti avoin. Näin on esimerkiksi tilanteissa, joissa käyttöjärjestelmien kehittäjät antavat rajoitetusti oikeuksia sovelluskehittäjille.

Avoin rajapinta

Avoimella rajapinnalla tarkoitetaan sellaista API:a, joka on julkinen ja jota voi käyttää ilman rajoittavia ehtoja. Edellytyksenä on, että API on hyvin dokumentoitu ja vaivattomasti ja vapaasti saatavissa siten, että ulkopuolinen pystyy ottamaan sen käyttöön ilman suuria vaikeuksia. (Coss s.a.) Mielestäni tekniikan avoimuuden voidaan ajatella olevan lähellä avoimen lähdekoodin periaatetta. Coss (s.a.) lisää, että jotta rajapinta olisi avoin, sen on täytettävä myös seuraavat ehdot: käyttöönotto on helppoa ja ei vaadi lisenssimaksua. Lisäksi avoimen rajapinnan testaus tulee olla mahdollista. Tähän voidaan lisätä, että palvelun käyttäjäksi tulee rekisteröityä, jotta saa API-avaimen. MyHelsinki Open API (s.a.) mukaan palvelimella oleva data voi sisältää sellaisia tietoja, jotka ovat lisenssin alaisia. Yleensä API:n tarjoaja ilmoittaa lisensseihin liittyvistä asioista.

Edut ja haitat

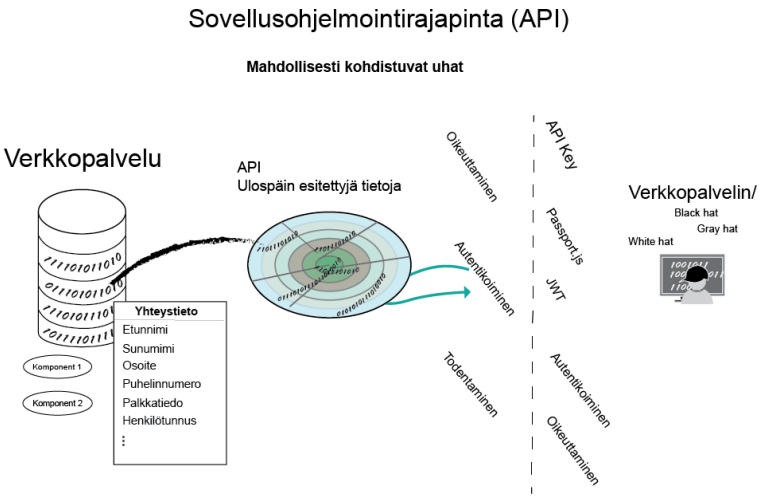
API:n hyödyntämisessä ymmärretään olevan merkityksellisiä etuja, mutta kuitenkin jo suunnitteluvaiheessa on osoitettava riskianalyysillä se, että sen käytöstä on enemmän hyötyä kuin haittaa. Kotkasen (2016, 4) mukaan rajapinta-arkkitehtuurin edut ovat yleisesti modulaariseen arkkitehtuuriin johtaminen, ylläpidon ja jatkokehityksen helpottaminen ja toimittajariippuvuuden vähentäminen.

API:n käyttämisessä ilmenee todennäköisesti haittoja, mikäli sitä ei ole suunniteltu huolellisesti: se voi esimerkiksi päästää vahingolliset koodit suoritettavaksi kohdepalvelimelle, olla sopimattomasti määritelty tai sitä ei ole huolellisesti dokumentoitu. (Pitkänen 2016, 19.)

Tietoturvallisuuden liittyvät pohdinnat

Ohjelmistot ovat perinteisesti suljettuja ympäristöjä. API:n määrityksestä saa jonkinlaisen käsityksen ohjelmiston sisällöstä, ja näin ollen voi ymmärtää ohjelmiston rakennetta. (Pitkänen 2016, 19.) Sovelluksen takaisinmallintaminen (reverse-engineering) käytetään muihinkin epämääräisiin tarkoituksiin. Zume-

ranin (2017) mukaan jotkin ohjelmistokehittäjät ovat takaisinmallintaneet verkkopalvelua saadakseen sille puutteellisen API:n. On siis huomattava, että avoimien rajapintojen yleistymisen myötä tietoturva- ja tietosuojariskit kasvavat entisestään (kuva 2). Tietojärjestelmässä oleva data herättää mielenkiintoa hyviin asioihin, mutta myös väärinkäyttöön.



Kuva 2. Tiedon avaaminen ulkopuoliselle lisää tietoturva- ja tietosuojariskiä.

Pitkänen (2016) lisää, että huonon rajapinnan suunnittelun takia joku voi päästä käsiksi laitteen ominaisuuksiin tai komponentteihin, jotka eivät olleet alun perin tarkoitettu avattaviksi.

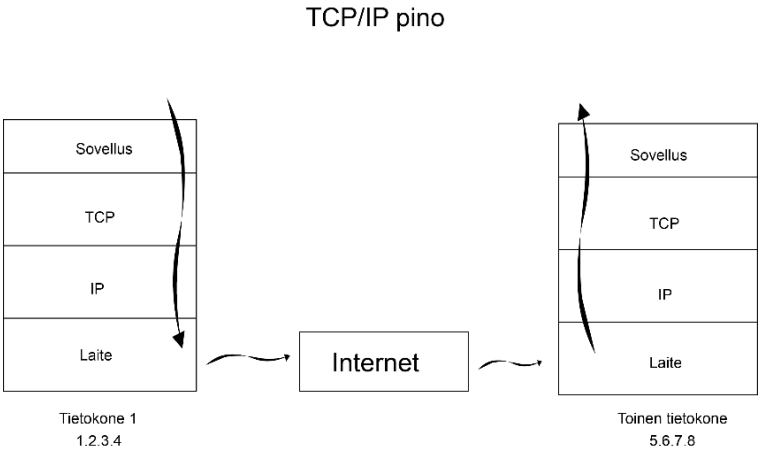
3.2 Verkkosovelluspalvelut

Laite-rajapinnassa tiedot siirtyvät esimerkiksi mikropiiristön kautta. Ulkoisen laitteen tieto voi siirtyä kaapelilla tai Bluetooth-langattoman tiedonsiirtotekniikan välityksellä. Sovellukset ja käyttöjärjestelmä voivat vaihtaa tietoja prosessien sisäisellä kommunikaatiolla. Eri laitteiden tai sovellusten välillä tiedonsiirto voi tapahtua tiedonsiirtoprotokollaa hyödyntäen.

Sovelluspalvelut ovat Internetistä tai sisäverkosta saatavia sovellusohjelmointirakenneosia, jotka käyttävät kommunikaatiokanavina avoimia protokollia (Redhat 2017). Sovelluspalvelut ovat riippumattomia toisistaan, käyttöjärjestelmästä tai ohjelmointikielestä, ja ne ovat itsemääriteltyjä kokoonpanoja. Nämä palvelut ovat julkisesti verkossa, josta niitä voidaan etsiä hakutoiminnoilla ja käyttää. W3C-yhtymä (Word Wide Web Consortium) ylläpitää sovelluspalveluiden määrityksiä.

3.2.1 TCP/IP-tiedonsiirtotekniikka

TCP/IP (Transfert Control Protocol / Internet Protocol) on tekniikka, jolla siirretään dataa laitteiden kesken tietoverkon yli. Tietoverkoilla tarkoitetaan tavallista langallista tai langatonta verkkoa tai muita tekniikoita, joilla TCP/IP-protokolla on implementoitu. Laitteet käyttävät lähtökohtaisesti fyysistä tai virtuaalista verkkosovitinta datan siirtämistä varten. TCP/IP sisältää kaksi protokollaa, jotka kuva 3 esittää TCP/IP pinon osana. Lyhenteet viittaavat protokolliin, jotka tarkoittavat sääntöjä, joilla datan siirtäminen on yhtenäistä ja määriteltyä siten, että osapuolet ymmärtävät toisiaan.



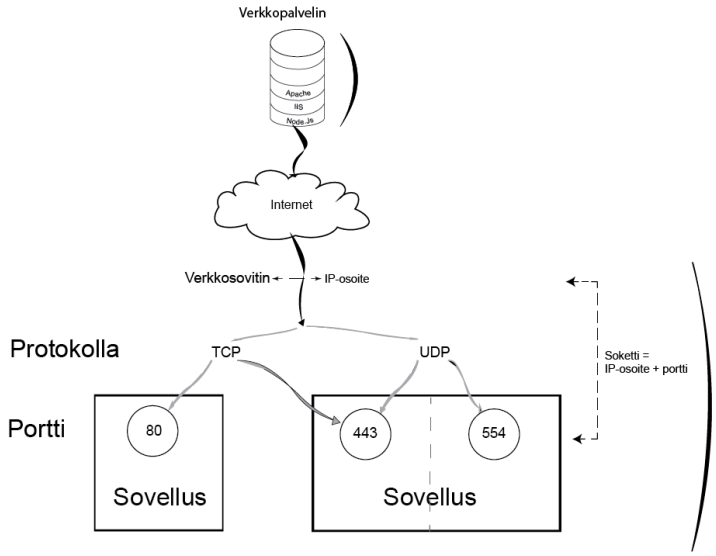
Kuva 3. TCP/IP pinot ja paketit (Shuler 2002)

TCP viittaa siihen, että datan siirtämisessä on virhetarkistus, joka takaa lähetettyjen datan oikeellisuuden. (Kaario 2002, 14-15.) UDP-protokolla (User Datagram Protocol) sen sijaan ei varmista datan perillemenoa. IP on alemman tason protokolla ja tekniikka, joka mahdollistaa maanlaajuisen tietoverkon rakentamisen käyttäen esimerkiksi IP-osoitetta toisen koneen löytämiseksi. TCP/IP-protokollan standardia valvoo IETF-organisaatio (The Internet Engineering Task Force).

TCP- ja UDP-portit

Käyttöjärjestelmä esittää sovellukselle TCP/IP-protokollakokonaisuuden loogisina yhteyksinä. Soketti-rajapinnalla luodaan looginen yhteys sovelluksen ja käyttöjärjestelmän välille (kuva 4).

TCP- ja UDP-portit ja soketit



Kuva 4. Looginen yhteys käyttöjärjestelmän ja sovelluksen välillä

Looginen yhteys on kaksiosainen: yksilöllinen IP-osoite ja porttinumero, joka kertoo, mille sovellusprosessille verkosta tullut datapaketti pitää ohjata (Kaario 2002, 11).

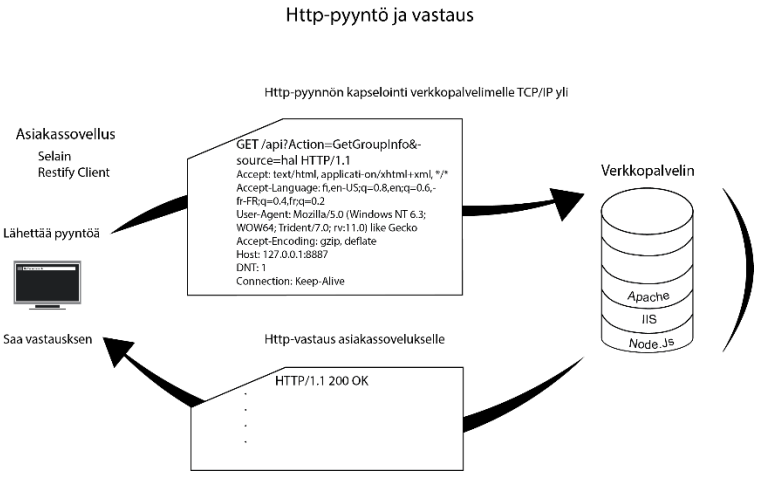
3.2.2 HTTP-protokolla ja metodit

HTTP (Hypertext Transfert Protocol) on protokolla, jonka avulla selain ja verkkopalvelin ovat vuorovaikutuksessa (Ekonoja ym. 2005) tuottaakseen hypermediapalveluita käyttäjille hypertekstin tekniikoilla. HTTP-protokolla on Internetin ensisijaisesti käytetty tiedonsiirtoprotokolla, jonka avulla selain tai asiakasohjelma muodostaa pyyntöjä verkkopalvelimille. HTTP/s on suojattu tiedonsiirtoprotokolla. HTTP-pyynnössä käytetään metodeja, jotka määritelevät pyyntöjen tyypin. Taulukossa 1 esitetään eniten käytettyjä metodeja. Ne ovat yleisimpiä, koska ne ovat perinteisiä tietojenkäsittelyn operaatioita, kuten luomista, lukemista, muokkaamista tai poistamista.

Taulukko 1. HTTP-metodeja

Metodi	**Selite**
GET	Resurssien hakua varten. Verkkosivun hakeminen tapahtuu GET metodilla. Hakuehdot näkyvät esimerkiksi selaimen osoiterivillä.
POST	Sivulla täytetyn lomakkeen tietojen lähettäminen palvelimelle. Tieto on näkyvissä vain pyynnön rungossa (body) .
PUT	Kuten GET-metodi, mutta kerrotaan API-palvelimelle, että halutaan tietojen tilan muutoksia.
DELETE	Pyyntö määrää palvelimen poistamaan tietoa tietovarannosta.
PATCH	Pyyntö määrää palvelimen päivittämään tietoa, joka on useimmiten PUT-metodin toiminto.

HTTP-pyyntö sisältää URL-osoitteen (Uniform Resource Location), otsaketiedon, joka näyttää suunnilleen kuvan 5 mukaiselta sekä siirrettävän tiedon. Tämä tietopaketti kapseloidaan HTTP-pyynnöksi, jonka asiakassovellus lähettää verkkopalvelimelle käsiteltäväksi.



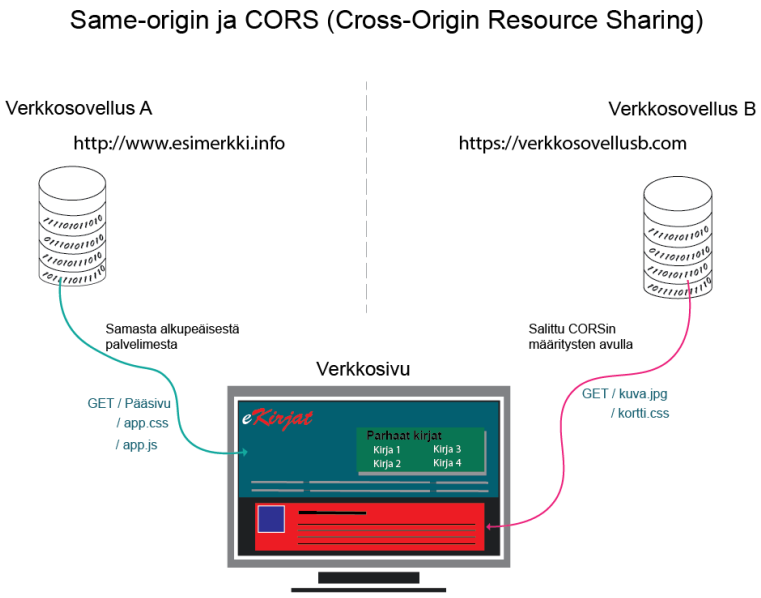
Kuva 5. HTTP-metodin GET-pyyntö ja vastaus

Verkkopalvelin käsittelee pyynnön, rakentaa sivun tai datapaketin ja lähettää ne vastauksena takaisin asiakassovellukselle. Asiakassovellus voi esimerkiksi rakentaa sivun selaimessa tai käsitellä tietoa muilla tavoin.

CORS:n (Cross-Origin Resource Sharing) määritelmät

Tietoturvallisuussyistä vahvistetiin sääntöä, jolla suojellaan verkkosovelluksen arkkitehtuuria siten, että pääsy resursseihin on rajoitettu samaan toimialueeseen (same-origin). Alkuperäisen verkkosovelluksen URL-osoite määritellään protokollasta, toimialueesta ja portista, esimerkiksi *http://www.esimerkki.com:8080*. Jos HTTP-pyynnön otsakkeessa on erillinen URL, selain estää koodin suorittamisen. (W3C 2010.)

CORS-tekniikka mahdollistaa muun muassa verkkosovelluksen suojaamisrajoituksen kiertämisen. Mekanismi mahdollistaa lopuksi yhden verkkosovelluksen pääsyn toisen resursseihin, kuten kuva 6 havainnollistaa.



Kuva 6. HTTP-pyyntö sallitaan CORS-määrityksen mukaan (MDN web docs s.a.)

CORS-pyyntö voidaan tehdä kahdella tavalla: ensimmäisellä yksinkertaisella tavalla, HTTP-pyynnön otsakkeeseen lisätään Origin-tietoa, esimerkiksi *Origin: http://www.esimerkki.com*. Tällöin kohdeverkkopalvelin tarkistaa pyynnön alkuperän, ja mikäli se löytyy CORS-konfiguraatiosta, se lisää tämän tiedon vastaukseen. Selain tarkistaa palvelimen vastauksesta *Access-Control-Allow-Origin* -arvon ja sallii pyynnön. Toisella tavalla, selain lähettää esivalmistellun

pyynnön, joka sisältää olennaisen pyynnön tarvittavan metodin ja otsakkeen. Tällöin kohdeverkkopalvelin palauttaa sallitut metodit ja otsakkeet. Selain tarkistaa ne ja suorittaa pyynnön, mikäli olennaisessa pyynnössä olevat metodi ja otsake ovat sallittuja kohdeverkkopalvelimella. (Google Cloud 2019.)

HTTP-vastauskoodit

Palvelin lähettää vastauksen asiakassovellukselle ja lisää myös vastauskoodin, joka kertoo pyynnön käsittelyn tilasta. Yleisesti suositellaan vastauskoodien käyttämistä sanojen sijaan. Taulukossa 2 esitetään muutama vastauskoodi onnistuneesta ja epäonnistuneesta pyynnöstä.

Taulukko 2. HTTP-vastauskoodeja

Vastauskoodi	**Selite**
200	OK. Pyynnön käsiteltiin onnistuneesti
201	Tietue on luotu. Kaikki on kunnossa
…	…
404	Ei löydy. Esimerkiksi sivua ei löydy
500	Palvelimen sisäinen virhe. Epämääräinen virhekoodi. Poikkeustapauksia voi olla moninaisia. Yleensä virhe vaatii toimenpiteitä verkkopalvelimen ylläpitäjältä.

Vastauskoodit auttavat myös verkkopalvelinta valitsemaan oikean virheilmoitussivun. Esimerkiksi ”sivua ei löydy” kertoo, että HTTP-pyynnöstä on saatu vastauskoodi 404, jolloin verkkopalvelin lähettää tämän virhesivun asiakassovellukselle.

3.2.3 Vastausviestien formaatit

API:n tiedonsiirtokanavana käytetään tyypillisesti HTTP-protokollaa. Sen lisäksi määritellään XML (Extended Markup Language) ja JSON (JavaScript Object Notation) minimaalisina tietoformaatteina tiedonsiirrossa (Redhat 2017.) XML etsii vielä paikkaansa esimerkiksi RSS-syötteissä (Really Simple Syndication). Nykyisin kuitenkin käytetään enemmän JSON-tietoformaattia. Asiakassovellus saa siis yleensä vastauksen joko HTML-, XML- tai JSON-tietoformaatilla. REST-arkkitehtuurissa, jota käsitellään luvussa 3.2.4, ei määrätä tiedon formaattia, vaan JSON on löytänyt paikkansa ohjelmistokehittäjien keskuudessa sen luomisen yksinkertaisuuden ja monipuolisuuden takia.

JSON-tietoformaatti

Esitetyissä sovelluksissa hyödynnetään alustariippumatonta, yksinkertaista ja monipuolisia ominaisuuksia sisältävää JSON:ia. Sen tiedon formaatti on helposti luettava ja koneellisesti helppo kirjoittaa ja tuottaa. Tiedot siirretään verkossa tekstinä, mahdollisia tietotyyppejä ovat esimerkiksi luvut, tekstit (string), taulukot ja totuusarvomuuttujat. (MDN web docs 2019.) JSON-tietorakenteessa tieto voi esiintyä yksittäisenä oliona (Single Level JSON) tai sisäkkäisenä, ja tiedot voivat olla joko kokoelmana tai avain–arvo-parina. Taulukko 3 havainnollistaa kaksi avain–arvo-paria: *malli* ja *sarja*, jotka erotetaan pilkulla.

Taulukko 3. JSON-tietoformaatin avain–arvo-parit

Json-dataa
{ malli: "Ford", sarja: "Escort" }

Taulukko 4 havainnollistaa Autot-kokoelman, jossa on kolme alkiota. Jokainen alkio on omassa nipussaan ja erotettuna pilkulla. Näin voidaan siis tunnistaa helposti automerkkejä, kuten Renault, Volvo ja Cadillac.

Taulukko 4. JSON-tietoformaatin autot-kokoelma

Json-dataa

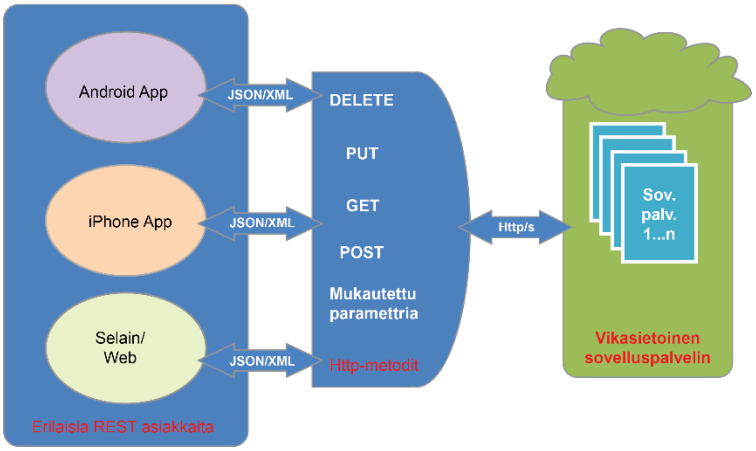
{
Autot:
[
{"merkki":"Renault", ”malli”:”20”,"vuosi":2019,
”vari”:”punainen”,”paino”:”1,230”},
{"merkki":"Volvo",”malli”:”V70”,"vuosi":1999,”vari”:”vir-
heä”,”paino”:”1,854”},
{"merkki":"Cadillac",”malli”:”De Ville”,
"vuosi":1960,”vari”:”valkoinen”,”paino”:”2,300”}
]
}

Taulukon 3 ja 4 mukaisten tietoformaattien muodossa avain on solmu, eli datan polussa, joka voidaan käydä läpi haluttuun tietoon pääsemiseksi. Esimerkiksi taulukosta 4 voidaan päästä Volvo-autoon attribuuttiin käyttämällä seuraavaa polkua: *Autot[1].merkki*.

3.2.4 REST-arkkitehtuurityyli

REST-arkkitehtuuri (Representational State Transfer architecture), on kuuteen rajoitteeseen perustuva arkkitehtuurityyli (Fielding 2000). Ensimmäinen rajoite on asiakassovelluksen ja palvelimen erottaminen toisistaan siten, että asiakassovellus ei ota kantaa tietolähteeseen eikä palvelin asiakassovelluksen tilaan. Näin tekniikassa on käytettävyyttä, skaalautuvuutta ja ympäristön ohjelmistojen riippumattomuutta. Toinen rajoite on asiakassovelluksen ja palvelimen kommunikoiminen tilattomasti siten, että asiakkaan jokaisen HTTP-pyynnön tulee sisältää kaikki tarvittavat tiedot, jotta palvelin voi palauttaa haluttua tietoa. Kolmas rajoite on välimuistin käytön mahdollisuus. Välimuistilla pyritään pienentämään siirrettävää datamäärää, koska muuttamatonta tietoa haetaan paikallisesta tietovarannosta. Neljäs rajoite on yhdenmukainen rajapinta, jossa käytetään yhtenäistä HTTP-metodia (kuva 7) tietojen hakemiseksi. (Fredrich s.a.) Arkkitehtuurin perustajan mukaan uniformaalinen liittymä laskee tehokuutta, koska silloin haetaan enemmän tietoa kuin olisi tarpeellista.

REST API-arkitehtuuri



Kuva 7. REST API -arkkitehtuuri (Dehos 2019)

Viides rajoite on kerroksittainen järjestelmä, jonka koostuu järjestelmien hierarkkisesta kerroksesta. Rajoite pyrkii parantamaan sen toimintaa internetin laajuisesti. Silloin asiakassovellus ei keskustele suoraan ensisijaisen palvelimen kanssa. Kommunikaatio tapahtuu välityspalvelimen kautta hierarkkista ympäristöä kunnioittaen. Viimeinen rajoite on REST-arkkitehtuurin ainoa valinnainen: ladattava koodi (code-on-demand). Se antaa mahdollisuuden käyttää sovelluskoodia sovelman tai skriptin muodossa. Rajoitteessa on tietoturvariski, mutta Fielding (2000) muistuttaa, että edut voivat olla suurempia kuin haitat. Hän myös kertoo, miten sisäverkossa asiakassovellukset voivat hyödyntää ominaisuutta, kun taas yrityksen palomuuri voi estää tällaisen sovelluskoodin ulkopuolisilta.

4 KÄYTETYT TEKNIIKAT JA MENETELMÄT

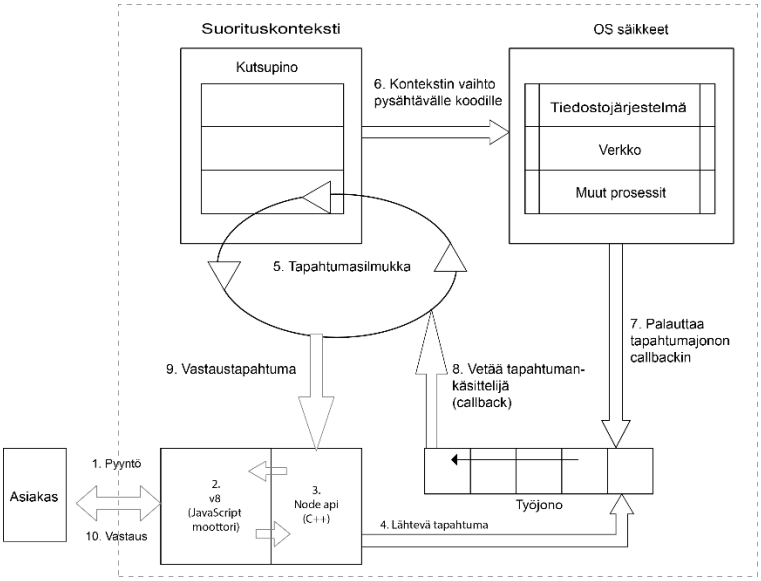
Tässä luvussa esitellään välineitä sovelluksen kehittämiseksi ja testiympäristön luomiseksi. Seuraavaksi määritellään JavaScriptin kehitystä ja roolia eri komponenteissa. Luvussa esitellään Node.Js-verkkopalvelinta, Express-ohjelmistokehystä, työkaluja tietojen salaamiseksi ja Bootstrap-viitekehystä käyttöliittymän kehittämiseksi. Lisäksi esitellään verkkopalvelinympäristössä tarvittavia istunnon ominaisuuksia.

4.1 Ohjelmistokehitysvälineet

JavaScript on olioperustainen ohjelmointikieli, joka Brendan Eich kehitti vuonna 1995. Ohjelmointikieli pohjautuu syntaksiltaan C-kielen ajatteluun, mutta JavaScript on yksinkertaisempi, esimerkiksi muuttujan tietotyyppi määrittyy dynaamisesti. Se tarkoittaa sitä, että määritetyn muuttujan tietotyyppi voi muuttua koodin suorittamisen aikana. (D'mello ym. 2017, 38.) Heidän mukaansa kuudenteen versioon (es6) on lisätty muuttujan tyypityksen ominaisuus, jolloin muuttujan tyypin määritteleminen on pysyvä.

Node.Js, Express ja JSON ovat viitekehyksiä, ajoaikaisia ympäristöjä tai tietomalleja, joilla on JavaScript pohjana. JavaScript kehittyy nopeasti, ja se lisää jatkuvasti enemmän tehokkuutta sovellusten kehitysprosessiin sekä tukea eri alustoille. JavaScriptin standardi nimi on ECMAScript, ja vuodesta 2015 lähtien on käytetty julkaisuvuosia, esimerkiksi ECMAScript 2015.

Node.Js on avoimella lähdekoodilla kehitetty, alustariippumaton ja asynkroninen JavaScript-ajoaikainen ympäristö, joka toimii verkkopalvelimena. Se suorittaa JavaScript-koodia palvelimen puolella, suorittaa pyydetyt toiminnot ja lähettää vastauksen asiakassovellukselle (kuva 8).



Kuva 8. Nods.Js-toimintamalli ja asynkroninen palvelupyyntö (D'mello ym. 2017)

Node.Js-toimintamalli eroaa muista verkkopalvelinohjelmistoista muun muassa siinä, että se lähettää tiedoston avausoperaation tiedostojärjestelmälle ja siirtyy odottamaan uusia HTTP-pyyntöjä. Näin ollen se kykenee käsittelemään

uusia pyyntöjä nopeammin. Tämä toimintamalli rajaa muistin ja verkon käyttöä ja sillä on matala vastausaika. (Node.Js 2019.)

Node.js:n reittien hallitseminen

Node.js:ssa määritellään pääohjelma, yleensä nimeltään *main.js*. Nimi voi olla jotain muutakin, mutta tämä on yleinen toimintamalli. Pääohjelmassa asetetaan sovelluksen tärkeimmät toiminnot, kuten reittien määritelmät, moduulien tuonnit (import) tai palvelimen käyttämä portti. Pääohjelman voidaan ajatella olevan sovelluksen ohjain tai ydin. Puhutaan myös sovelluksen sisääntulosta (entry point), jonka kautta kaikki HTTP-pyynnöt kulkevat.

Express-moduuli

Express on nopea, joustava ja minimalistinen ohjelmistokehys, joka toimii Node.Js:n päällä. Express sisältää paljon tehokkaita ominaisuuksia, joilla voidaan luoda yksittäisiä sivuja tai kokonaisia verkkosivustoja sekä tietokoneille että mobiililaitteille. Lisäksi se tuo mukanaan tehokkaan väliohjelmakonseptin (Middleware), joka toimii HTTP-pyyntöjen käsittelijän ytimessä. Väliohjelma hoitaa esimerkiksi eväisteiden tai istuntojen luomista ja ylläpitämistä. (Keig 2013, 21; Express routing s.a.; D'mello ym. 2017, 73.)

Express-moduuli ladataan myös verkkosovelluksen pääohjelmaan. Se käsittelee reittien avulla HTTP-pyynnössä käytetyt metodit, kuten GET tai POST, jolloin määritellään Express-ilmentymän vastaavat metodit, esimerkiksi *app.get(…)* tai *app.post(…)*. Näin Express ohjaa pyynnön oikeaan metodiin. (Express routing s.a.)

Node.js:n Crypto-moduuli

Node.Js:lle kehitetty Crypto-moduuli sisältää salaukseen liittyviä toiminnallisuuksia. Sen avulla salataan tietoa eri tavoilla ja eri tarkoituksiin. Hash- ja HMAC-metodeilla moduuli luo tiedosta tiivisteen, joka ei ole tarkoitettu purettavaksi. Tiiviste on käytännöllinen tietojen vertailussa, esimerkiksi kaksoistiedon etsimisessä tietokannasta. Cipher- ja Decipher-algoritmeilla moduuli salaa tietoa, joka on purettava Decipher-metodilla. Hash- ja Cipher-metodit hyväksyvät

erilaisia suojausalgoritmeja, kuten SHA1 tai turvallisimmat SHA256 ja SHA512. (Node.Js s.a.)

Käyttöliittymän rakentaminen Bootstrap-viitekehyksellä

Bootstrap-viitekehys on suosittu työkalu modernien verkkosivustojen kehityksessä. Se mahdollistaa nopean sivuston kehittämisen sekä yhtenäisten ja responsiivisten sivujen rakentamisen. Se sisältää valmiita komponentteja ja elementtejä, jotka taipuvat muun muassa laitteen näytön kokoon. Bootstrap on avoimen lähdekoodin työkalupakki. Sivujen tekemiseen käytetään HTML:ää, CSS:ää ja JavaScriptiä. Sisäänrakennettu ruudukko helpottaa sivujen taittamista. Bootstrap toimii ”mobiili ensin” -ideologian pohjalta. (Bootstrap 2019.) Viitekehityksen avulla koodin kirjoittaminen vähenee. Näin sovelluskehittäjä voi keskittyä muihin tehtäviin, esimerkiksi verkkosovelluksen toiminnallisuuteen.

Ohjelmiston prototyyppi

Prototyyppi on yksinkertainen malli sovelluksesta. Se on esituotannon vaihe, joka sisältää muutaman ominaisuuden lopullisesta tuotteesta. Jotta sovellusta voidaan testata mahdollisimman nopeasti, siitä luodaan suppea versio, joka sisältää tärkeimmät ja kriittisimmät toiminnot lopullisen toteutuksen kannalta. Puhutaan myös alfa-versiosta. Alfa-ratkaisulla on etuna se, että suunnitteluvaiheet voidaan siirtää uuteen ulottuvuuteen tuomalla konkreettisia kokeilumahdollisuuksia. Käytännössä silloin opitaan sovelluksen käyttäytymisestä sidosryhmän käyttäjän toimesta. Lisäetuna on se, että silloin saadaan mahdollisuus esitellä asiakkaalle sovellus pienellä kustannuksella. Prototyyppiä ei välttämättä kehitetä loppuun, eikä sovelluksen koodi ainakaan tule käyttöön lopulliseen toteutukseen. (Digitaalinen Helsinki s.a.) Alfa-version jälkeen seuraa beetavaihe, joka yleensä julkistetaan edelleen kokeilua varten. Kehitys jatkuu palautteen kautta.

CSV-tietomuoto

CSV (Comma-Separated Values) on erityinen tiedostotyyppi. Sen pääte on *.csv*, esimerkiksi *autot.csv*. Taulukko 5 havainnollistaa autot-taulukkorakenteita CSV-muodossa. Ensimmäisellä rivillä on yleensä sarakkeiden otsikko. Kentät tallennetaan tekstinä ja ne erotetaan yleensä pilkulla. Välilyönnin tai pilkun sisältämät kentät merkitään tiedostossa lainausmerkeillä. Näin tuontisovellus käsittelee niitä merkkijonona tai yhtenä tietona. (Microsoft s.a.)

Taulukko 5. CSV-muoto autot-taulukosta

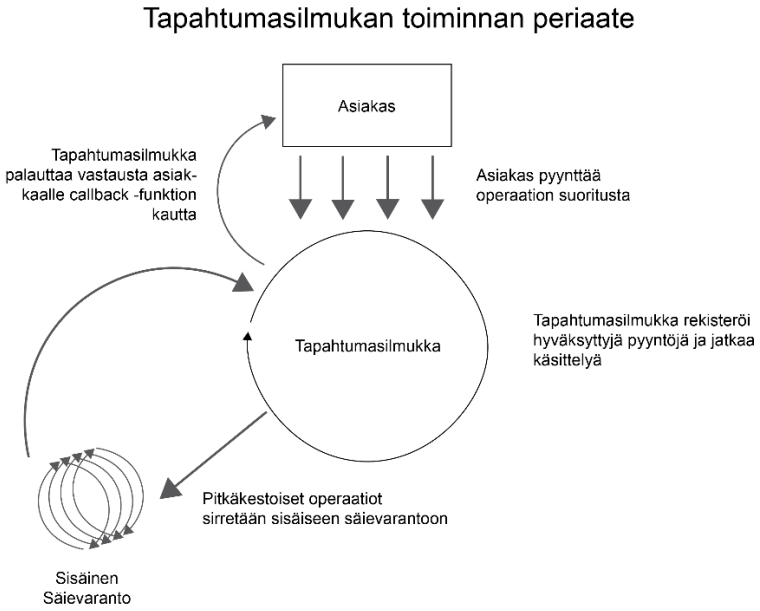
Automalleja
Merkki,Malli,Vuosi,Väri,Paino
Renault,20,2019,Punainen,”1,230”
Volvo,V70,1999,Vihreä,”1,854”
Cadillac,”De Ville”, 1961,Valkoinen,”2,200”

CSV-tietoformaatti eroaa huomattavasti JSON:stä. Esimerkiksi CSV:ssä tieto ei ole jäsennelty avain–arvo-parina. Ennemminkin tieto on taulukkomaisesti esitettynä siten, että pilkku toimii erottavana elementtinä. CSV-tietoformaatti on hyvin tuettu sekä laskentataulukko- että tietokantasovelluksissa.

4.2 Verkkopalvelimen toimintaympäristö, asynkroninen palvelupyyntö, istunto ja eväste

Asynkronisella toiminnalla tarkoitetaan rinnakkaista suoritusta sovelluksen osista, joita voivat olla moduuli, funktio tai metodi. Node.Js ei odota vastausta kutsuvasta funktiosta, vaan jatkaa käsittelemään seuraavaa koodikäskyä. Voidaan ajatella, että sovelluksen pääsäie (main thread) suoritetaan samanaikaisesti kutsutun osan kanssa. Asynkroninen funktio täyttää ja viimeistelee tehtävän ja palauttaa tietoa pääsäikeelle takaisinkutsufunktion (callback, promise tai async/await) kautta (kuva 9). Asynkroninen eroaa synkronisesta toiminnasta siten, että prosessit ovat vuorovaikutteisia: seuraavaa funktiota ei suoriteta ennen edellisen loppumista. Mikäli synkronisen funktion suoritus kestää pitempään, sovelluksen suoritus hidastuu. Silloin käyttökokemus huononee, sillä käyttäjälle näkyy ikään kuin sovellus olisi jumissa. Sen takia synkronista funktiota ei suositella. Toki tietyssä tilanteessa toiminta on sallittu, esimerkiksi

sovelluksen käynnistyessä. (Express performance Best Practices Using Express in Production s.a.)

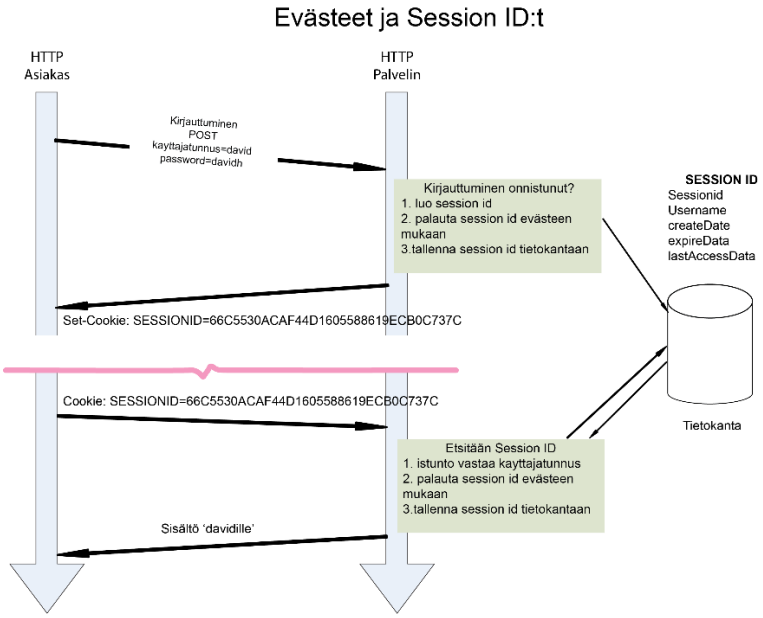


Kuva 9. Tapahtumansilmukan toiminnan ja asynkronisuuden periaate (Bosak 2015)

Esimerkiksi käyttöjärjestelmiin kirjautumisten prosessit voivat osittaisesti tai kokonaan tapahtua asynkronisesti. Tarvittaessa ylläpitäjä voi muuttaa asetuksia, kuten kirjautumiskomentosarjan suorittamisen ennen työpöydälle pääsemistä. Node.Js mahdollistaa molemmat menetelmät. Tiedostojärjestelmän vasteaika vaihtelee laitteen kuormituksen mukaan. Myös verkkoa tarvitseva sovellus on altis verkon nopeuden vaihtelemiselle. Näin ollen metodin, joka hakee tietoa verkon yli, on toimittava asynkronisena. Vastauksen saapuessa Expressin promise-metodin logiikka käsittelee joko siinä olevaa dataa tai virhettä epäonnistuneesta operaatiosta.

Express-session

Verkkopalvelin varaa jokaiselle verkkosivustolle muistialueen, johon voidaan tallentaa tietoja, kuten sivuston asetuksia tai sivujen tilaa (kuva 10). Istunnon elinaika on määriteltävä. Oletuksena on, että istunnon tiedot häviävät verkkosovelluksen tai verkkopalvelimen uudelleen käynnistyessä. Niitä on kuitenkin mahdollista tallentaa varastoon ja ladata uudelleen, esimerkiksi sovelluksen käynnistyessä.



Kuva 10. Eväste ja session ID kirjauttamisprosessissa (CSCI S-12 s.a.)

Verkkopalvelin yksilöi istunnon tunnusluvulla, joka tallentuu istunnon evästeeseen. Testiympäristössä tallennetaan istuntotiedot Express-sessionin oletusvarastoon, mutta tuotantopalvelimella on tapana tallentaa niitä sille tarkoitettuun tietokantaan. (Npm 2019.)

Evästeet ovat yleisesti pienimuotoisia tietokoneen levylle tallennettuja tiedostoja, jotka sisältävät verkkosovellusten tai verkkosivustojen asetuksia tekstimuodossa. Ne pyrkivät parantamaan selaamiskokemusta siten, että kaikkea sivustojen tietoja ei tarvitse syöttää uudestaan. Evästeiden elinikä voidaan määritellä, ja määrittelemättömänä ne ovat voimassa selaimen istunnon ajan. Esimerkiksi voidaan tallentaa käyttäjän kirjautumistiedot. (Microsoft 2019.)

Tietosuojan ja tietoturvan takaamiseksi sovelluskehittäjän on suunniteltava huolellisesti evästeiden käyttö ja se, mitä tietoa tallennetaan ja kuinka kauan tietoa säilytetään. Evästeissä tallennetut tiedot ovat usein henkilötietolaissa määritettyjä henkilötietoja, kuten tietokoneen IP-osoitteita, paikannustietoja tai muita mahdollisia kirjautumistietoja. Sen takia ominaisuuden käytöstä on informoitava sivun vierailijoita. (EU:n tietosuoja-asetus 2016/679.)

5 KEHITETYT OHJELMISTOT

Tarve sovellusohjelmointirajapinnoille liiketoiminnan osana kasvaa jatkuvasti. Nähtävästi useat yritykset eivät ole osanneet hyödyntää API:a, tai niissä ei ole

varattu aikaa aiheen tutkimiselle. Resurssien puute on osasyy. Euroopan Unionin tietosuoja-asetukset (GDPR) muuttavat tilannetta huomattavasti, sillä ne ohjaavat palveluntarjoajia lisäämään sellaisia ominaisuuksia, jotka mahdollistavat käyttäjän tietojen siirtämisen verkkopalvelujen välillä.

Luvussa 3 käsiteltiin sovellusohjelmointirajapintaa ja verkkosovelluspalvelua. Esiteltiin myös tiedonsiirtotekniikoita ja protokollia ja niiden puitteita, eli HTTP-pyyntöjä ja -vastauksia, jotka määrittävät tietoformaatin. Hyväksi toimimistavaksi esiteltiin myös CORS-tekniikka. Tässä luvussa esitellään kahden API:a käyttävän asiakassovelluksen toimintatapaa ja kehittämisvaihetta. Ensimmäisellä sovelluksella konkretisoidaan tietojen hakemista palveluntarjoajalta, mikä on Otavian tarpeena ja toiveena. Toinen sovellus on jatkokehityksenä. Siinä esitellään malli, joka kykenee toimimaan riippumatta palveluntarjoajasta.

Testidata

Eventilla on luonut testiympäristön Otavialle, jotta päästään kehittämään Pyramuksen API kummallekin osapuolelle turvallisesti. Tässä raportissa tulostetut tiedot ovat ainoastaan testidataa, henkilötietojen käsittelysäännöksiä kunnioittaen.

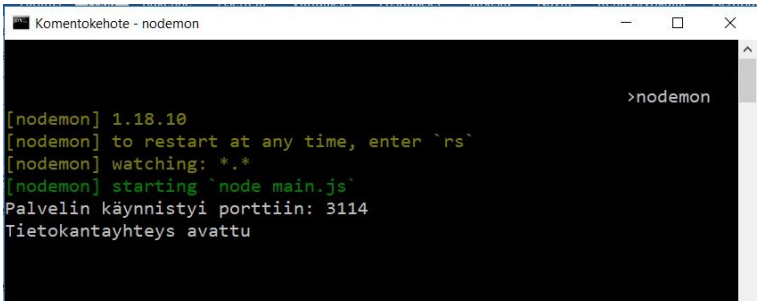
5.1 Eventilla Pyramus Proxy

Tietojen hakemista varten kehitetään modulaarinen API-asiakassovellus Eventilla Pyramus Proxy (EPP), joka voidaan asentaa Pyramus-palvelimelle. EPP hakee tietoja Eventillan palvelimilta tiedonsiirtoprotokollan avulla, jäsentää niitä ja muotoilee sopivaksi JSON-tietorakenteeksi. Moduulit integroidaan käyttöliittymään, jotta voidaan esittää tiedot sidosryhmälle. Sovellus kehitetään pääsääntöisesti Node.js:lla. Yksinään Node.js ei riitä, joten käytetään myös muita tekniikoita. Niitä ovat esimerkiksi Express, EJS, Express-session, Cookie-session, Crypto, Restify (REST-arkkitehtuuri), Jsonexport, File System ja XAMPP-tuotepaketti, josta käytetään MySQL-tietokantamoottoria ja Apache-verkkopalvelinta.

Tietokanta-ohjelmiston ja Node.js-verkkopalvelimen käynnistäminen

EPP-ohjelmiston kehitysvaiheessa käytetään pienimuotoista tietokantaa. Tauluja tarvitaan kaksi sovelluksen vähimmäistietoturvan takaamiseksi. MySQL ja Apache sopivat erinomaisesti tällaiseen tehtävään, sillä ne eivät vaadi järjestelmältä paljon resursseja, ja myös siksi, että ne ovat avoimella lähdekoodilla kehitettyjä. Näin ollen pysytään Pyramuksen tavoin avoimen lähdekoodin maailmassa. Tietokantaan tallennetaan EPP:n käyttäjätunnus ja Eventillan API Accountin tietoja. XAMPP-ohjauspaneelilla käynnistetään MySQL ennen testin aloittamista, jotta kirjautuminen onnistuisi.

Verkkopalvelin käynnistetään Node.exe:n sijaan Nodemonin avulla (kuva 11). Sovellus käynnistää verkkopalvelimen ja monitoroi projektin kansion sisällön mahdollisia muutoksia, jolloin Nodemon käynnistää palvelimen uudelleen.



Kuva 11. Eventilla Pyramus Proxyn verkkopalvelin on kuulolla.

Testin aikana palvelin voidaan halutessa käynnistää Node.exe:lla, sillä muutoksia ei tehdä koodiin, eikä sovellus muuta kansion sisältöä muutenkaan.

Kuten kaikki verkkopalvelimet, Node.js tarvitsee TCP/IP-rajapinnan tietoliikennettä varten portin, josta TCP- tai UDP-paketti siirtyy (ks. kuva 4). EPP-sovelluksessa määriteltiin portti 3114 tietoliikennettä varten. Siihen porttiin asiakasohjelmistot, esimerkiksi selain tai Restify Client, ottavat yhteyttä. Verkkosovellukseen päästään siis osoitteesta http://localhost:3114.

Käyttöliittymä

EPP sisältää kaikki olennaiset ominaisuudet moduuleina, jotka mahdollistavat integroimisen. Käyttöliittymä mahdollistaa tietojen näyttämisen yleisölle. Se pidettiin mahdollisimman yksinkertaisena, koska sillä ei ole suurempaa käyttöä lopullisessa Pyramuksen API-ominaisuudessa. Sivuston tuottamisessa käytettiin Bootstrap-viitekehystä. Tietosuojan vuoksi sovelluksen käyttö rajoitettiin käyttäjätunnuksella ja salasanalla, jotka luotiin apusovelluksella ja syötetiin käsin tietokantaan. Käyttäjän muokkaus- tai lisäämistoiminnallisuutta ei ole. Kirjautumisprosessissa käsitellään myös Eventillalta saatuja API Accountin ja API Keyn, joilla verkkopalvelu tarkistaa käyttöoikeuden. Näillä tiedoilla suoritetaan HTTP-pyyntö palveluntarjoajan palvelimelle, joten tietoturvan kannalta tietoja käsitellään huolellisesti, vaikka ne ovat vain testiympäristöä varten. On huolehdittava myös siitä, että API Account ja Key suojataan Pyramuksessa parhaimmalla tekniikalla.

Edellä mainitut API Account ja API Key syötetään lomakkeelle ja tallennetaan tietokantaan salattuna käyttäen suojausavaimena samaa salasanaa kuin käyttäjätunnuksessa, jotta salauksen purkaminen tapahtuisi samassa prosessissa.

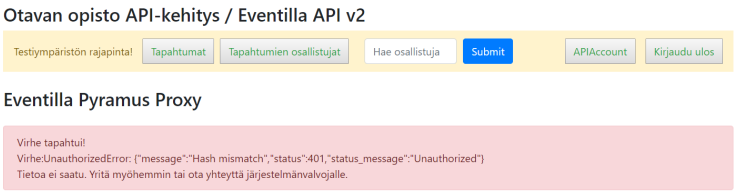
API Keyn salaaminen on ollut haasteellista. Crypton Hash-metodi ei ole sopinut tähän tehtävään, sillä tieto on saatava takaisin alkuperäiseen muotoon HTTP-pyynnön tekemiseksi. Siksi päätettiin käyttää Crypto-moduulin sisältämää createCipheriv-metodia. CreateCipherivillä luodaan salakirjoitettu merkkijono, joka on purettavissa. Taidot eivät riittäneet suoranaisen salaustekniikan koodaamiseen, joten työssä hyödynnetiin soveltaen Node.js Communityn esimerkkiä.

Sisäänkirjautuminen EPP:seen

Käyttäjän tiedot tallennetaan evästeeseen, joka säilyy joko selaimen istunnon ajan tai määräaikaisesti. Sovellus hakee syötetyn käyttäjätunnuksen tietokannasta ja vertaa sitä lomakkeella annettuihin tietoihin. Salasana ei missään vaiheessa ole selkokielisenä eikä purettavissa. Samasta prosessista sovellus hakee API Accountin tiedot tietokannasta. Toisin kuin salasanan, sovellus pur-

kaa API keyn Crypton createDecipheriv-metodilla. Tehtävä ei suoraan onnistunut, monesta yrityksestä huolimatta. Salauksen purkamista vaikeutti virheellisesti käytetty ”backquote/backtick”, joka aiheutti ylimääräisiä merkkejä siten, että tietokannasta haettu salattu tieto ja purkamistieto erosivat merkittävästi toisistaan. Ongelmaa selvitettiin ongelmankorjaustoimenpiteellä (debug), ja käytännössä pulma ratkesi arvojen tulostamisella konsoliin. Ratkaisu löytyi lopulta helposti.

Onnistuneesta kirjautumisesta huolimatta voi ilmestyä (myös englanninkielisenä) virheilmoitus, esimerkiksi ”UnauthorizedError – –”, joka tulee vastauksena Eventilla API -palvelimelta (kuva 12) mikäli avain on virheellinen tai oikeuttaminen epäonnistuu jostakin muusta syystä.



Kuva 12. Luvattoman käytön virheilmoitus

Muihinkin poikkeuksiin varaudutaan ja käsitellään asianmukaisesti, sillä sovelluksen suoritus täytyy jatkaa niin, että käyttäjä ei kokea sovelluksen kaatumisena. Virheilmoitukset esitetään mahdollisimman tarkasti ja ohjataan käyttäjää tekemään sopivaa toimenpidettä. Virheilmoituksen esitetään käyttöliittymällä siten, että käyttäjä löytää sen helposti.

EPP:n valikot ja välilehdet

EPP:n valikosta voidaan hakea sekä *tapahtumia* ja *tapahtumien osallistujia* että *osallistujan tietoja* hakutoiminnon avulla. Lisäksi valikosta löytyy *APIAccount*-optio API Accountin hallinnoimiseksi, ja tietenkin *uloskirjautumis*-optio. Välilehden avulla esitetään Eventillalta haettua tietoa eri muodossa. API:n avulla haettuja tietoja voidaan esittää ja käyttää monella tavalla. Rajausta ei periaatteessa ole, vaan se riippuu käyttötarkoituksesta. Tapahtumien ja osallistujien tietoja päätettiin nimittäin esittää eri muodoissa, joita ovat *kortti*-, *taulukko*-, *JSON*- ja *CSV*-muodot. Teknillisesti sovellus hakee sekä tapahtumat

että osallistujat vain kerran, ja ne tallennetaan muuttujiin myöhempään käyttöön. Näin vähennetään HTTP-pyyntöjä palveluntarjoajan palvelimelta, säästetään palvelujen kuormitusta, tietoliikennettä ja kenties saadaan turvallisempi ympäristö.

Tietojen hakemisen esivalmistelu

Eventilla API -palvelin vaatii käyttöluvan, jonka erikoisuus tekee palvelun käyttöönottamisen odotettua vaikeammaksi. Eventillan teknologiajohtajalta saatiin kuitenkin PHP-ohjelmointikielellä (Hypertext Preprocessor) tehty esimerkki sovellettavaksi. Häneltä saatiin neuvoja ja lisää ohjeita muillakin tavoin.

Oikeuttamisprosessi on monivaiheinen. Käytännössä luodaan datanippu, jossa yhdistetään API AccountId:n, API Keyn, HTTP-metodin, palvelimen päätepisteen, resurssin polun ja päivämäärän UTC-formaatilla (Coordinated Universal Time). Esimerkiksi päivämäärä täytyy olla päivitetty jokaisella HTTP-pyynnöllä, joka tarkoittaa uuden otsakkeen luomista (ks. kuva 15). Datanipussa on kaksi merkittävää osaa: päivämäärän vaativa formaatti ja datanipun salaaminen. Crypton HMAC-tiivisteen luomisprosessissa syötetään salainen avain, tässä tapauksessa API Key, sitten salataan datanippu HMAC:n SHA1-algoritmilla ja viimeiseksi muutetaan tiiviste base64-muotoon. Tämäkin tehtävä onnistui pitkän taistelun jälkeen. Lähinnä selvisi, että koodin toimimattomuus johtui siitä, että etappien välissä oli virheenkorjaustarkistuspiste, joka pysäytti koodin suorittamisen. Tällöin tietojen salausmekanismit eivät toimineet kunnolla.

Tapahtumien ja osallistujien tietojen hakeminen

Käynnistyessä EPP hakee Eventillan palvelimelta kaikki tapahtumat. Osallistujien tietoja haetaan samoilla menetelmillä, joten luvussa esitellään yhtä tiedon hakemista, koska eroina ovat vain HTTP-pyynnöt ja JSON-tietorakenteet. Tiedon haku on asynkroninen, koska data haetaan verkosta, joten on otettava huomioon verkon latenssi (ks. luku 4.2). Node.js:ssa käytetään Promise.all-metodia, joka tekee kokoelman useista Promiseista. Tämä viimeinen luo uuden ilmentymän. Promise.all-metodissa voidaan käynnistää useita hakutoimintoja tai metodeja kerran, ja odottaa sitten niiden vastauksia, jotka voivat olla

hyväksyttyjä tai hylättyjä. Kuva 13 havainnollistaa Promise.all:in koodi-muodolla.

```
// make a promise.all call. NOte that for now we have only on http-request to make.
// input: server endpoint + param, other serve information fron array
return Promise.all( [
                        getEventsWithPromise(
                                                `${EventillaAPIAccount.endpoint}/${serversApiContexts[0].param}`,
                                                serversApiContexts[0],
                                                ''
                                               )
                    ]
                  )
    .then((events) => {                 // promise successful

        // retreive the data form index 0 to remove the extra bracket!!
        // return - with callback - the data to main.js
        callback(null, events[0] );
    })
    .catch((AppError) => {                      // error happened to promise not fullfilled

        // return error
        callback(AppError, null);
});
```

Kuva 13. Promise.all –metodin käyttäminen

Promise-metodissa suoritetaan varsinaisia HTTP-pyyntöjä, mukaan lukien oikeuttaminen. Aikaisemmin puhuttiin siitä, että oikeuttaminen tapahtuu jokaisella pyynnöllä, koska Eventilla API -palvelin tarkistaa pyynnön kellonajan. Eventilla API:n mielenkiintoisuus ja toimintamallin erikoisuus ovat nähtävissä JSON-tietorakenteissa kuvassa 14. Hienous on esimerkiksi kenttien valinnassa kyselymerkkijonolla *param*-muuttujassa. HTTP-pyynnön muodostamista varten käytetään Node.js:lle asennettavaa Restify-clients -moduulia. Sitä voidaan ajatella restify-client selaimena, joka hakee kotisivun verkkopalvelimelta. Restify-clientin kaltainen sovellus antaa kehittäjälle mahdollisuuden käsitellä tietoja ohjelmallisesti.

```
// Eventilla API's paths and params
// fecth = switch while retreiving events or attendees. So, not part of Eventilla's API
const serversApiContexts = [
                            {
                                "fetch" : "events",                 // get events
                                "uriPath" : "/v2/events",           // used for hmac
                                "param" : "events"
                            },
                            {
                                "fetch" : "attendees",              // get all attendees for all events
                                "uriPath" : "/v2/events/eventId/attendees",
                                "param" : `events/eventId/attendees?fields=id,href,firstname,lastname,email,
                                           canceled,modified,data,links,registered,status,event`
                            },
                            {
                                "fetch" : "attendees",              // get attendees for one event
                                "uriPath" : "/v2/attendees/eventId",
                                "param" : `attendees/eventId?fields=id,href,firstname,lastname,email,
                                           canceled,modified,data,links,registered,status,event`
                            }
                           ]
```

Kuva 14. Eventilla API’s polut ja parametrit tietoihin käsiksi pääsemiseen

HTTP-pyyntö aloitetaan määrittelemällä http-asiakas (HttpClient) Restifyn createJsonClient-metodin avulla (kuva 15, rivit 79 - 81). Siihen syötetään Eventilla API -palvelimen päätepiste. Dynaamisuuden saamiseksi korvataan lennossa *uriPath*-attribuutista *eventId*-merkki oikealla tapahtuman koodilla. Http-asiakkaan luomisen jälkeen suoritetaan HTTP-pyyntö restifyClient.get-metodilla, johon annetaan parametrina otsake (kuva 15, rivit 93 - 101). Palvelin vastaanottaa pyynnön ja vahvistaa oikeuttamisavaimen. Se käsittelee

pyynnön ja lähettää pyydetyt tiedot kutsujalle datamuuttujassa, tai virheen err-muuttujassa, mikäli GET-metodi epäonnistuu.

```
77
78      // create a intance of JSON restifyClient. Set the endpoint dynamically
79      let restifyClient = restify.createJsonClient({
80          url: url.replace("eventId", conn[0].escape(eventId).replace('/'/g, ""))
81      });
82
83      // Promise (async) method calling. Return data on resolve or error
84      return new Promise((resolve, reject) => {
85
86          // get date in UTC format
87          let mdate = getEventDate();
88
89          // get the hmac of the string
90          let hmac = getHash(EventillaAPIAccount.apikey, mdate, serverApiContext.uriPath.replace("eventId", eventId));
91
92          // create the header in JSON format including the authentication data
93          let params = {
94                                  "headers" : {
95                                                  "Content-MD5" : contentMD5,
96                                                  "Content-Type" : contentType,
97                                                  "Accept" : "application/json",
98                                                  "Date" : mdate,
99                                                  "Authorization" : `KVWS ${EventillaAPIAccount.accountId}:${hmac}`
100                                 }
101         };
102
103         // use the Restify restifyClient instance to execute the get-method. Error and Data variable are used
104         // providerParam = params passed after he server endpoint
105         restifyClient.get(params, (err, req, res, data) => {
106
```

Kuva 15. HTTP-pyyntö Eventilla API -palvelimelle

Onnistuneesta HTTP-pyynnöstä restifyClient-metodissa vastaanotetaan tietoa JSON-formaatilla. Kuvasta 16 voidaan ihailla API:n tekniikan mahdollisuuksia ja JSON-tietoformaatin selkeyttä. *Events*-haaran voidaan nähdä olevan kokoelma, eli taulukko, joka voi olla joko tyhjä tai sisältää yhden tai useampia tapahtumia. Lisäksi otteessa on *languages*-kokoelma ja yksi *name*-sisäkkäinen attribuutti *location*-attribuutin alaisena.

```
1   {
2     "events": [
3       {
4         "id": "xayED",
5         "url": "https://ssl.eventilla.com/event/xayED",
6         "languages": [
7           "FI"
8         ],
9         "name": "test 2",
10        "description": "<p>T&auml;m&auml; on testi</p>",
11        "timezone": "Europe/Helsinki",
12        "starts": "2019-02-01 01:15:00",
13        "ends": "2019-02-05 23:01:00",
14        "organization": "Otavan opisto API-kehitys",
15        "organization_id": null,
16        "status": "past",
17        "location": {
18          "name": null,
19          "address": null,
20          "postal": null,
21          "city": null,
22          "country": null
23        },
```

Kuva 16. Eventilla API -palvelimelta saatua dataa

Palveluntarjoajan API ryhmittelee tiedot kategoriana, esimerkiksi tapahtuman *id*, *url* ja *languages*. Tämä ryhmittely auttaa tiedon käsittelyä ja tiedon eheyden säilyttämistä.

Tietojen poiminen ja tulostaminen sivulle

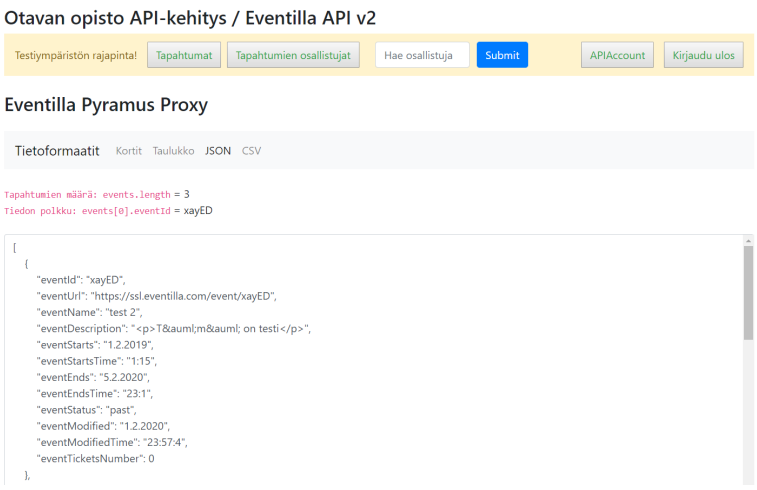
Eventilla API -palvelimelta saatujen tietojen määrä ylittää tietenkin Pyramuksessa tarvittavat. Näin ollen ne joudutaan käymään läpi ja poimimaan vain pakolliset tiedot. Tietojoukko käydään läpi silmukalla. Sillä luodaan uusi muuttuja

JSON-tietoformaattiin (kuva 17, rivit 157 - 170), joka tallennetaan uuteen taulukkoon (kuva 17, rivi 174). Tämä taulukko palautetaan Promise.All-metodille, joka vuorostaan palauttaa sen reittien ohjaimelle.

```
132
133         // walk through the data received from Eventilla
134         data.events.forEach((event) => {
135
136             // get dates end times and format them
137             eventStartDate = new Date(event.starts);
138             eventStartTime = `${eventStartDate.getHours()}:${eventStartDate.getMinutes()}`;
139
140             eventEndDate = new Date(event.ends);
141             eventEndTime = `${eventEndDate.getHours()}:${eventEndDate.getMinutes()}`;
142
143             evtStartDate = `${eventStartDate.getDate()}.${eventStartDate.getMonth()+1}.
144                             ${eventStartDate.getFullYear()}`;
145             evtEndDate =   `${eventEndDate.getDate()}.${eventEndDate.getMonth()+1}.
146                             ${eventStartDate.getFullYear()+1}`;
147
148             // get updated date and time and format them
149             eventModifiedDate = new Date(event.modified);
150             eventModifiedTime = `${eventModifiedDate.getHours()}:${eventModifiedDate.getMinutes()}:
151                                  ${eventModifiedDate.getSeconds()}`;
152
153             eventModified = `${eventModifiedDate.getDate()}.${eventModifiedDate.getMonth()+1}.
154                              ${eventModifiedDate.getFullYear()+1}`;
155
156             // store the event information to a JSON object
157             eventInformation = {
158                                     "eventId" : event.id,
159                                     "eventUrl" : event.url,
160                                     "eventName" : event.name,
161                                     "eventDescription" : event.description.replace(/\\n/g,""),
162                                     "eventStarts" : evtStartDate,
163                                     "eventStartsTime" : eventStartTime,
164                                     "eventEnds" : evtEndDate,
165                                     "eventEndsTime" : eventEndTime,
166                                     "eventStatus" : event.status,
167                                     "eventModified" : eventModified,
168                                     "eventModifiedTime" : eventModifiedTime,
169                                     "eventTicketsNumber" : event.tickets.length
170                                 }
171
172
173             // push the event information to the collection
174             events.push(eventInformation);
175
176         });
177
178         // promise is fullfilled, resolve it with the collection on events
179         resolve(events);
180
```

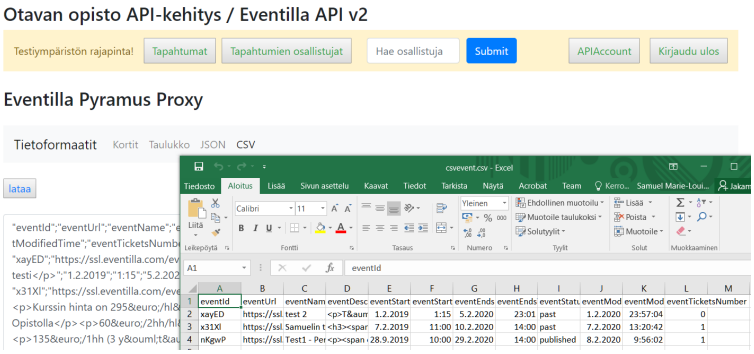
Kuva 17. Saadut tiedot käydään läpi ja poimitaan vain tarvittavat.

Haetun tiedon tulostaminen sivustolle hoidetaan EJS-moduulilla (Embedded JavaScript templating). Moduulin render-metodi mahdollistaa nopean HTML-sivujen rakentamisen dynaamisesti mallipohjasta (EJS s.a.). Kuvassa 18 esitetään haetut tiedot JSON-muodossa, tapahtumien määrä ja esimerkki polusta, jolla voidaan päästä tietoon.



Kuva 18. Tiedon tulostaminen JSON-muodossa

Lisäksi kuvasta 18 löytyy tietoformaattivalikko, josta voidaan valita jokin muu esitystapa, kuten taulukko tai kortti. Tietoja voidaan myös viedä CSV-tiedostoon käsiteltäväksi muualla tavoin. CSV-tiedoston hyödyntäminen havainnollistetaan kuvassa 19.



Kuva 19. Tiedon vieminen CSV-muotoon ja importoiminen Microsoft Exceliin

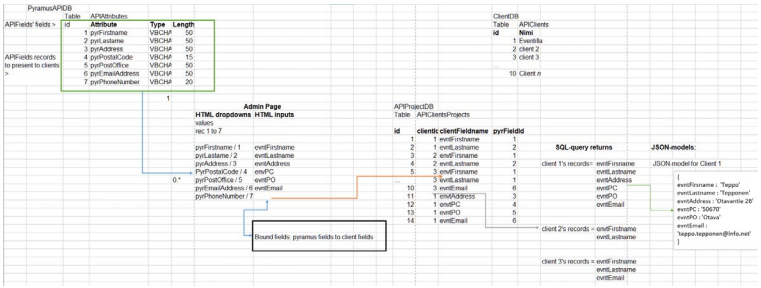
CSV-tietomuodon hyödyntäminen kolmannella sovelluksella käy helposti esimerkiksi Microsoft Excelillä. Tietomuodon yksinkertaisuudesta ja rajoituksista huolimatta monet ohjelmistot, kuten laskentataulukko-, tekstinkäsittely- tai tietokantaohjelmistot tukevat CSV-tuontia.

5.2 Dynaaminen Pyramus API Client

EPP:n toimintamalli ja rakenne sitovat Pyramuksen ja Eventillan toisiinsa. EPP on ratkaisu yhteen ongelmaan: saada tietoja toisesta verkkosovelluspalvelusta ohjelmallisesti. Ratkaisu voi kuitenkin koitua ongelmaksi ajan myötä. Ulkopuolisten tietojen tuottajien lisääntyessä joutuisi rakentamaan erilaisia tietojen käsittelijöitä (data parser) palveluntarjoajakohtaisesti. API:n ympäristön kehittäminen tulisi hankalaksi, ylläpidoltaan raskaaksi ja skaalautuvuudeltaan rajalliseksi, jopa mahdottomaksi.

Ohjelmistokehitystiimin pyynnöstä kehitetään API-malli, jossa Pyramus pystyisi hakemaan tietoja miltä tahansa ulkopuoliselta tuottajalta mahdollisimman pienellä muutoksella sovelluksessa. Sovellus kehitetään muun muassa Node.js:lla, Expressillä, EJS:lla, MySQL:lla ja Restify-clientsillä. Mallissa tuottajan API-attribuutit eivät ole enää kovakoodattuina sovelluksessa, vaan ne tallennetaan tietokantaan. Näin ollen malli perustuu tietokantapohjaiseen ympäristöön. Kuva 20 esittää yksinkertaisesti mallin tietokannan rakennetta ja so-

velluksen prototyypin sisäistä rakennetta ja tehtävän kulkua. Attribuutit muodostavat Pyramus API Scheman (PAS) rungon, joka vastaa isäntäsovelluksen tuotantotietokannan tietorakennetta. On huomattava, että attribuutit voidaan nimetä muutenkin tietoturvan parantamiseksi ja takaisinmallintamisen estämiseksi siinä tapauksessa, että API avataan ulospäin. Etuna tässä mallissa on se, että DPAC voi myös olla integroitu Pyramukseen, sillä suuria koodin muutoksia ei tarvitsisi tehdä.



Kuva 20. Pyramuksen API Schema ja mallin prototyyppi

Pyramus API Scheman attribuutti esitetään DPAC:n ohjauspaneelilla, josta se sidotaan vastaavan tuottajan API:n attribuuttiin. Sidotut attribuutit tallennetaan yhteen tauluun. Pyramus API Schemaan voidaan lisätä uusia attribuutteja ja poistaa tai muokata tarpeen mukaan. Toimintamalli lisää dynaamisuutta ohjelmointirajapinnan käyttöön, kun attribuutteja voidaan muokata lennossa ja SQL-kyselyllä (Structured Query Language) haetaan tietoja käsittelijää varten.

DPAC:n käyttöliittymä

Sovelluksen verkkosivu rakennettiin myös Bootstrap-viitekehyksellä. Käyttöliittymä pidettiin tässäkin yksinkertaisena. Sen tarkoituksena on havainnollistaa mallin ominaisuutta ja toiminnallisuutta, joten rakennettiin yksi sivu, joka jaettiin kahteen osaan. Ensimmäisessä osassa tulostetaan tuottajan rajapinnasta haettua tietoa kaikkine käytettyine attribuutteineen. Tällä testataan mallin toimivuutta syöttämällä hakukenttään tuottajatunnus. Tuottajan tiedot, joihin kuuluvat palvelimen päätepiste, parametrit ja avaimet oikeuttamista varten, tallennettiin toistaiseksi DPAC:an (kuva 21). Seuraavassa versiossa luodaan taulu tuottajan tiedoille, kuten tavallisesti tehdään.

Käyttöliittymän toisessa osassa sidotaan PAS:n ja tuottajan API:n attribuutteja toisiinsa. Pudotuslistalta valittu attribuutti tulostaa tietotyypin ja pituuden auttaakseen ylläpitäjää valitsemaan oikean attribuutin. Koska PAS on Pyramuksen tuotantotietokannan attribuuttien mukainen, niin tekstitietotyypin pituudella on merkitystä, esimerkiksi se leikkaa pois ylimääräisiä merkkejä. Sivulla hallinnoidaan attribuuttien sitomista lisäämällä, muokkaamalla tai poistamalla. Sivustossa käytetään JavaScriptia ja dokumenttioliomallia (DOM), joka mahdollistaa HTML-elementtien muokkausta dynaamisesti. Prototyypissä Pyramus API Schema muokataan käsin suoraan tietokannassa. Kaikki muutokset tulevat voimaan välittömästi tallennuksen jälkeen.

DPAC:n tietokanta

Tässäkin sovelluksessa käytetään MySQL-tietokantaa. DPAC:n tietokantaan kuuluu kolme taulua, vaikka prototyypissä on vain kaksi. Ne ovat *apiattributes* ja *apiclientprojects*. Apiattributes-taulun tietomalli (layout) pidettiin mahdollisimman yksinkertaisena prototyypin tekemiseen. Kentät ovat: *id*, *attribuutti*, *tietotyyppi* ja *pituus*. Nimen lisäksi tärkeitä ovat datatyyppi ja kentän pituus. Pyramuksen tuotantotietokanta vaikuttaa pakostakin API Scheman tauluun. Sidotut attribuutit tallennetaan apiclientprojects-tauluun, jolla on seuraavat kentät: *APIcpId*, *clientId*, *clientFieldname* ja *clientPyrApiAttributeId*.

Testiympäristö

Prototyypin kehittämiseen tarvitaan Eventillan lisäksi avoimia datapalveluja. Avoimia datapalveluja on runsaasti, mutta päädyttiin kuitenkin rakentamaan testiympäristö, jossa DPAC:n lisäksi on kaksi palvelinta. Näin ensinnäkin siksi, että prototyypin alkukehityksellä käsitellään tasaista JSON-tietoformaattia. Toiseksi siksi, että mallin kehitys aloitettaan mahdollisimman yksinkertaisella tietolähteellä. Jatkokehityksessä lisätään Eventilla API -tietolähde testiympäristöön. API-testipalvelimet esitetään tarkemmin Verkkopalvelimet-kappaleessa. Testitietojen tuottajiksi valittiin Espoon asuntokuntien tulot asuntokunnan elinvaiheen ja alueen mukaan 2020- testi (6Aika 2017a) ja paikkatietoja (6Aika 2017b). Testidata imuroitiin CSV-muodossa kehittäjän tietokoneelle ja tuotiin (import) tietokantoihin. Testipalvelimet toimivat näin ollen avoimen datan portaalina.

Verkkopalvelimet

Verkkopalvelimet ovat yksinkertaisia toimivuudeltaan ja kooltaan. Niitä kehitettiin Node.js:lla, Restifylla ja Restify Cors Middlewarellä. Restify Cors-moduuli otetaan käyttöön tietoturvan varmistamiseksi siten, että sovellus oikeuttaisi resurssien käyttöä eri toimialueilta (ks. luvusta 3.2). Palvelimet vastaavat portteihin 3116 ja 3117. Kuvassa 21 (rivit 21 - 37) listataan testipalvelimen kokoonpano ja sen päätepiste, resurssien osoite sekä muita parametreja.

```
20  // External Provider API's server details
21  const ExternalProviders = [ {
22
23                  description: "Testipalvelin 1 - käyttäjälista",
24                  serverEndpoint: "http://localhost:3116",                // Target endpoint
25                  uriPath: "/api/kayttajat",                              // Target uriPath
26                  param: "items",                                         // JSON-data's starting root
27                  hasAuthorization: false,                                // Target server's need authorization
28                  authorizationModule: "",                                // Authorization Module's path
29                  url: "http://localhost:3116",                           // URL for http-request
30                  module: "./runtimeFunctions",                           // Module for customized data parser
31                  buildCollectionOfContent: false,                        // Precollection of item Identification has to be made
32                  collectionBuilderServer: null,                          // Server Configuration that handle the precollection
33                  initializationRoutines: [],                             // Note used. Reserve for Initialization Sequence
34                  accountId : null,
35                  apiKey : null,
36                  nestedJSON: false,                                      // Sets attrib as key/value or array
37                  Id: 1                                                   // Server Configuration Identity
38              },
54              {…
55              },
56              {
57                  description: "Eventillan kaikki tapahtumien listaus",
58                  serverEndpoint: "https://ssl.eventilla.com/v2",         // Target endpoint
59                  uriPath: "/v2/events",                                  // Target uriPath
60                  param: "events",                                        // JSON-data's starting root
61                  hasAuthorization: true,                                 // Target server's need authorization
62                  authorizationModule: "../../models/externalProviders/authorizationModule.js",    // Authorization Module's path
63                  url: "https://ssl.eventilla.com/v2/events",             // URL for http-request
64                  module: "./externalProviders/EventillaRuntimeFunctions",// Module for customized data parser
65                  buildCollectionOfContent: false,                        // Precollection of item Identification has to be made
66                  collectionBuilderServer: null,                          // Server Configuration that handle the precollection
67                  initializationRoutines: [],                             // Note used. Reserve for Initialization Sequence
68                  accountId : null,
69                  apiKey : null,
70                  nestedJSON: false,                                      // Sets attrib as key/value or array
71                  Id: 4                                                   // Server Configuration Identity
              },
```

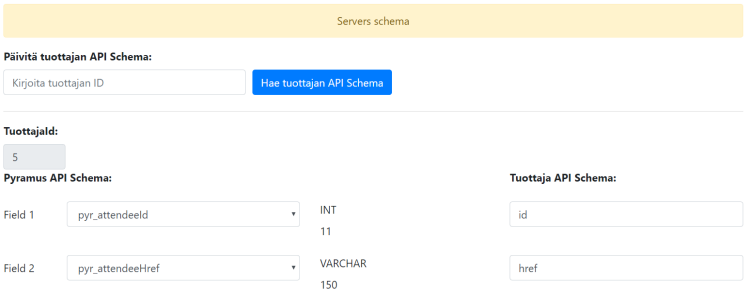
Kuva 21. Verkkopalvelinten päätepiste ja palvelinkoonpanon muut attribuutit

DPAC tekee testipalvelimille HTTP-pyyntöjä. Palvelimet hakevat dataa tietokannoista ja lähettävät vastaukset response.send -metodin avulla. Tietojen hakeminen Eventillasta hoidetaan kuvan 21 (rivit 55 - 71) kaltaisella palvelinkokoonpanolla. Sen attribuutit muodostavat yhden palvelinkokonaisuuden, jolla haetaan yhteenkuuluvaa tietoa. Haettava resurssi määritetään *uriPath*-attribuutilla. Lisäksi attribuuttien avulla käsketään DPAC suorittamaan esitehtävää HTTP-pyynnön oikeuttamiseksi (rivi 60), rakentamaan JSON-dataa tietyllä muodolla (rivi 69) tai lataamaan moduulia (rivi 63), joka suorittaa erikoisia tehtäviä, kuten päivämäärän formatoimisen.

Tuottajan API Scheman määritteleminen palvelinkohtaisesti

Ensin on lisättävä tuottajan tietoja DPAC:an kuvan 21 mukaisesti. *UriPath*-attribuutti vastaa API-palvelimella olevaa pääohjelman reittiä, esimerkiksi ”server.get("/api/kayttajat",– –)”. Tuottajan tietojen olemassaolo edesauttaa API Scheman luomista. Alustavasti haetaan tuottajan tiedot hakutoiminnolla, jolloin

voidaan valita Pyramus API Schemasta attribuutteja, joita sidotaan lopuksi tuottajan API Schemaan (kuva 22).



Kuva 22. Attribuuttien sitominen hallintasivulla

Pudotuslistan vastapäästä tulostetaan attribuutin tietotyyppi ja pituus, jotka auttavat sitomispäätöksessä. Tuottajan API Scheman kentässä voidaan lisätä attribuutin lisäksi ominaisuuksia, jotka määrittelevät miten sen data käsitellään. Ominaisuudet ovat *alias, funktio* tai *operaatio*, ja niitä syötetään kenttään muodolla ”*atribuutti:alias*:*funktio*” (kuva 23).

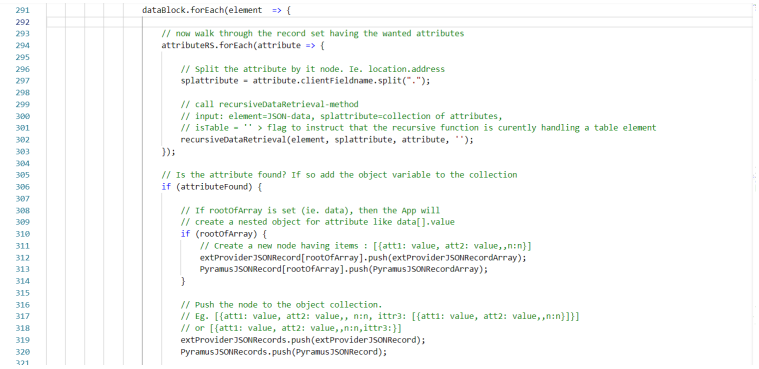
Kuva 23. Attribuutin ominaisuuden lisääminen

Funktiot kutsutaan dynaamisesti ladatusta moduulista (kuva 21, rivit 29 ja 63). Esimerkiksi kuvasta 23 Field 11 -kohdalla suoritetaan ”Time Format” operaatiota registered-attribuutilla, jolloin päivämäärä *"2019-02-01 20:15:06"* muutetaan muotoon ”*20:15:06”*. Seuraava esimerkki on Field 14 -kohdalla, johon lisätiin funktio, joka saa tehtäväkseen suodattaa *data*-taulukon attribuutteja. Attribuuttien sitomisen jälkeen voidaan suorittaa tiedon haku käyttöliittymän ensimmäisestä osasta.

Tiedon hakeminen

Tiedon hakeminen tapahtuu hakutoiminnolla, jolloin DPAC suorittaa HTTP-pyynnön valitun tuottajan API-palvelimelle. Testipalvelimilla ei ole oikeuttamisen tarvetta, joten pyyntö on näin ollen suoraviivainen. Mikäli tuottajan API-

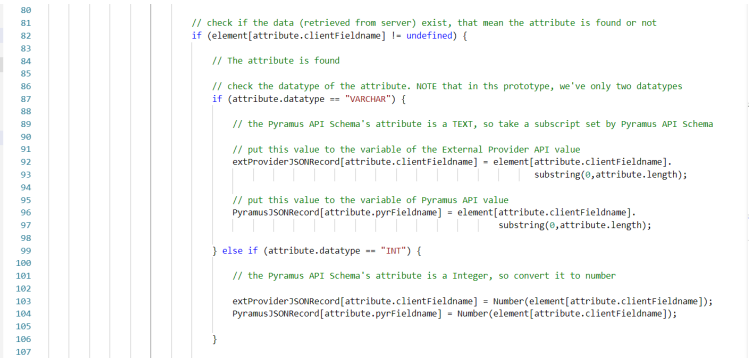
palvelin vaatii oikeuttamisen, niin DPAC luo ladatun moduulin avulla (kuva 21, rivi 61) asianmukaisen HTTP-pyynnön. Haettua tietoa käydään läpi kahdella silmukalla, josta ensimmäinen lukee haettua tietoa ja toinen käy läpi sidottujen attribuuttien tietojoukkoa (kuva 24, rivit 291 ja 294). DPAC käyttää rekursiivisuusmenetelmää JSON-tietorakenteen attribuuttien polun kulkemiseen.



```
291         dataBlock.forEach(element  => {
292 
293             // now walk through the record set having the wanted attributes
294             attributeRS.forEach(attribute => {
295 
296                 // Split the attribute by it node. Ie. location.address
297                 splattribute = attribute.clientFieldname.split(".");
298 
299                 // call recursiveDataRetrieval-method
300                 // input: element=JSON-data, splattribute=collection of attributes,
301                 // isTable = '' > flag to instruct that the recursive function is curently handling a table element
302                 recursiveDataRetrieval(element, splattribute, attribute, '');
303             });
304 
305             // Is the attribute found? If so add the object variable to the collection
306             if (attributeFound) {
307 
308                 // If rootOfArray is set (ie. data), then the App will
309                 // create a nested object for attribute like data[].value
310                 if (rootOfArray) {
311                     // Create a new node having items : [{att1: value, att2: value,,n:n}]
312                     extProviderJSONRecord[rootOfArray].push(extProviderJSONRecordArray);
313                     PyramusJSONRecord[rootOfArray].push(PyramusJSONRecordArray);
314                 }
315 
316                 // Push the node to the object collection.
317                 // Eg. [{att1: value, att2: value,, n:n, ittr3: [{att1: value, att2: value,,n:n}]}]
318                 // or [{att1: value, att2: value,,n:n,ittr3:}]
319                 extProviderJSONRecords.push(extProviderJSONRecord);
320                 PyramusJSONRecords.push(PyramusJSONRecord);
321 
```

Kuva 24. Haetun tiedon läpikäyminen forEach-silmukan ja rekursiivisuuden käyttäen

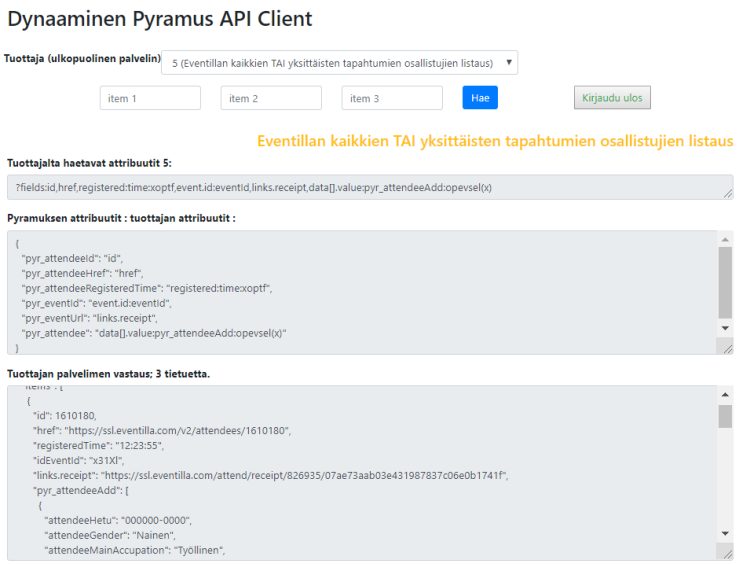
Attribuutin tarkistus tehdään hyödyntäen JavaScript-olion attribuutin käsittelemistä (property accessors, associative arrays). Ominaisuutta käytetään joko []-merkeillä tai tunnisteella, esimerkiksi *kayttaja[”etunimi”]* tai *kayttaja.etunimi*. Tulee huomata, että ”*etunimi*” voidaan korvata muuttujalla. Tämä tarkoittaa, että sidotut attribuutit voidaan korvata lennossa (kuva 25, rivi 82). Esimerkiksi jos sidottu attribuutti on ”*postitoimipaikka*”, niin sovellus luo *kayttaja[”postitoimipaikka”]* -muuttujan. Tämän muuttujan avulla voidaan tarkistaa attribuutin olemassaolo ja tallentaa attribuutin arvo. Kuva 25 (rivit 92, 96, 103 ja 104) havainnollistaa tarkistus- ja tallennusoperaatiot.



```
80 
81             // check if the data (retrieved from server) exist, that mean the attribute is found or not
82             if (element[attribute.clientFieldname] != undefined) {
83 
84                 // The attribute is found
85 
86                 // check the datatype of the attribute. NOTE that in ths prototype, we've only two datatypes
87                 if (attribute.datatype == "VARCHAR") {
88 
89                     // the Pyramus API Schema's attribute is a TEXT, so take a subscript set by Pyramus API Schema
90 
91                     // put this value to the variable of the External Provider API value
92                     extProviderJSONRecord[attribute.clientFieldname] = element[attribute.clientFieldname].
93                                                                        substring(0,attribute.length);
94 
95                     // put this value to the variable of Pyramus API value
96                     PyramusJSONRecord[attribute.pyrFieldname] = element[attribute.clientFieldname].
97                                                                 substring(0,attribute.length);
98 
99                 } else if (attribute.datatype == "INT") {
100 
101                     // the Pyramus API Schema's attribute is a Integer, so convert it to number
102 
103                     extProviderJSONRecord[attribute.clientFieldname] = Number(element[attribute.clientFieldname]);
104                     PyramusJSONRecord[attribute.pyrFieldname] = Number(element[attribute.clientFieldname]);
105 
106                 }
107 
```

Kuva 25. Attribuuttien poimiminen tuottajalta saadusta datasta

Mikäli API Schema ei vastaa tuottajan API:n attribuutteja, niin DPAC palauttaa *null*-arvon. Lopulta DPAC luo kokoelman tiedosta, jotta se voidaan palauttaa Pyramukselle. Käyttöliittymällä havainnollistetaan kuvan 26 avulla haettuja attribuutteja, Pyramus API -attribuutit sidottuina tuottajan API-attribuutteihin ja lopuksi saapunutta tietoa JSON-tietoformaattina.



Kuva 26. Tuottajalta saapunutta tietoa tulostaminen

Oikean tiedon valitseminen JSON-datasta on helppoa ja yksinkertaista. Lisäksi dynaamisen moduulin käyttö helpottaa erityisten toimenpiteiden tekoa attribuutteihin. Palveluntarjoajien API:n määrittelyn tunteminen on tässäkin mallissa alku onnistumiselle. DPAC-mallin kehittäminen mahdollistaisi tiedon hakemisen muiltakin verkkosovelluspalveluilta.

6 JOHTOPÄÄTÖKSET

Opinnäytetyön johtopäätöksessä pohditaan Otavian toimintaympäristön kehitystä sekä Pyramuksen API-ominaisuuksien puutteita. Lisäksi esitellään mahdolliset korjaukset ja lopuksi esitellään valitut ratkaisut. Tämän luvun lopuksi esitellään toimeksiantajalle tarkoitetut jatko- ja kehitysehdotukset.

6.1 Toimintaympäristön ongelman pohdinta

Otavian Pyramus-oppilaitoshallintajärjestelmän jatkuva kehittäminen on erittäin tärkeää, sillä sen täytyy taata liikelaitoksen useiden ja lisääntyvien koulu-

tusmuotojen palveleminen. Ohjelmistokehitystiimi keskittyy ensisijaisesti järjestelmän tärkeimpiin osiin, sillä uusien ominaisuuksien lisääminen vaatii väistämättä lisää resursseja. Järjestelmän täytyy kuitenkin ottaa vastaan tietoa muista lähteistä jollakin automaattisella menetelmällä. Työntehokkuuden lisääminen on myös riittävä syy kehittää Pyramusta. Tietojen siirtäminen käsin verkkopalvelusta toiseen ei ole järkevää kasvavan liiketoiminnan harjoittamisen kannalta. Tietojenkäsittely kokonaisuudessaan on elintärkeä sekä asiakkaille että liikelaitokselle. Näin ollen on järkevää panostaa sopivan työtavan ja työkalun etsimiseen ja niiden käyttöönottamiseen, mukaan lukien tietojärjestelmän turvallisuuden nostaminen, tietojen nopeampi käsitteleminen ja tiedon eheänä pitäminen.

Ongelman ratkaisemiseksi päätettiin käyttää API:n tekniikkaa, joka on tietojärjestelmien järkevin tiedonsiirtomenetelmä, sillä palveluntarjoaja ja Pyramus ovat hajautettuja verkkosovelluksia. API kattaa useiden vuosikymmenien kehityksen ja käytön. Sitä käyttävät esimerkiksi suuret toimijat, jotka avaavat käyttöjärjestelmät kolmannelle osapuolelle joidenkin rajapintojen avulla ohjelmistojen kehittämiseksi. Haasteena pidettiin Pyramuksesta puuttuvaa rajapinta. Ei ollut tietoa, mistä sovelluksen rakentaminen voitaisi aloittaa. Tässä tapauksessa API-toiminnallisuuden lisääminen suunniteltiin tyhjästä. Esimerkiksi työvälineitä, kuten ohjelmointikieltä tai lopullista tietoformaattia, mietittiin tarkasti. API:n lisääminen vaikuttaa myös Eventillan verkkopalvelun tarjoamiin mahdollisuuksiin. Selvitystyössä saatiin tietoa siitä, että Eventilla tarjoaa asiakkailleen sovellusohjelmointirajapinnan, joka mahdollistaa ohjelmistokehityssuunnitelman laatimisen. Viimeisenä haasteena pidettiin ohjelmistokehitystiimin toivomaa geneeristä rajapintaympäristöä. Kun vastaavaa menetelmää ei ollut tiedossa, jouduttiin suunnittelemaan ja kehittämään tyhjästä.

API-toiminnallisuuden toteuttamiseksi kehitettiin keväällä 2019 Eventilla Pyramus Proxy ja Dynaaminen Pyramus API Client -sovellukset. Pyramukseen integroitavat moduulit paketoitiin käyttöliittymillä tietojen havainnollistamiseksi. Sovellukset ovat prototyyppejä. Ensiksi mainitun sovelluksen avulla todettiin, että on mahdollista siirtää tietoja Eventillalta. Alkuvaiheen ongelmien ratkaisujen etsiminen hidasti kehitystä. Suurimmat ongelmat ilmenivät verkkopalvelun oikeuttamisessa, sillä HTTP-pyynnön otsakkeen luominen ja muut API:n määritellyt erottuivat muista tunnetuista palveluntarjoajista. EPP:n avulla voidaan

hakea tietoa vain Eventillasta, eli se on sidonnainen palveluntarjoajaan. Eventillan API:n käyttökelpoisuuden todentaminen avasi tien DPAC-sovelluksen kehittämiseen. Tavoitteena oli kehittää sovellus, joka kykenee hakemaan tietoa eri verkkosovelluspalveluilta. Tekniikka perustuu API:n määritekartoitukseen, jossa verkkosovelluspalvelujen käyttämät attribuutit sidotaan verkkosovellukseen ja tallennetaan tietokantaan. Näin erotetaan Pyramus API toisesta verkkopalvellusta, jolloin datan käsittely on dynaaminen ja se tapahtuu määritekartoituksen avulla.

Työvälineiden ja työkalujen valinnan lisäksi haettiin tietoa erilaisilta tahoilta. Eventillalta saatiin tarvittavat tiedot verkkopalvelun oikeuttamiseksi ja ohjeita rajapinnan käyttöönottamiseksi. Pyramus-ympäristön vastaavalta ja koulutussihteeriltä saadut tiedot mahdollistivat sovelluksen kehittämisen suunnitelman mukaiseksi.

6.2 Jatko- ja kehittämisehdotuksia

Sovellukset esiteltiin säännöllisesti ohjelmistokehitystiimille, tuoteomistajalle ja koulutussihteerille arviointia ja kehitysideoita varten sekä opettaja-ohjaajalle. Toimeksiantajan ja muun sidosryhmän palautteesta voidaan päätellä, että EPP täyttää tehtävänsä. Myös DPAC on käyttökelpoinen ja käyttötarkoituksensa mukainen.

EPP- ja DPAC-sovellukset sisältävät moduuleita, jotka hoitavat oikeuttamisen, haetun tiedon jäsentämisen ja JSON-tietoformaatin luomisen. DPAC pyrkii täyttämään toimeksiantajan pyytämän ominaisuuden, joten sen jatkokehittämistä prototyypistä valmiiksi tuotteeksi suositellaan, moduulien lähdekoodeja kehittäen. Silloin DPAC toimisi tiedon tuottajana Pyramukselle. DPAC:n käyttöönottaminen voidaan toteuttaa kahdella tavalla:

- integroimalla moduulit suoraan Pyramukseen
- itsenäisesti ja kokonaisesti toimivana verkkopalveluna.

Suoralla integroinnilla saadaan moduulit lisättyä Pyramuksen lähdekoodiin. Integroinnin esitehtävänä on moduulien lähdekoodien kääntäminen samalla kääntäjällä kuin pääohjelma virheiden etsimiseksi ja poistamiseksi. DPAC-ratkaisumallin integrointia suoraan pidetään hyvänä, sillä koodin muokkaamisen

määrä ja kääntäminen pienenevät mallin dynaamisuuden vuoksi, lukuun ottamatta moduulien käyttöönoton alkuvaihetta. Etua on myös siinä, että API-ominaisuus siirtyisi avoimeen lähdekoodiin.

Itsenäisellä verkkopalvelulla tarkoitetaan erillisen palvelimen käyttöönottoa. Siinä Node.Js -verkkopalvelinohjelmisto huolehtisi HTTP-pyynnöstä. Tämä verkkopalvelin toimisi Eventillan ja Pyramuksen välillä. Jotta tieto siirtyisi DPAC:ista Pyramukseen, tarvittaisiin liitintä (connector) tai rajapintaa, jonka kautta pyynnöt ja vastaukset kulkisivat. Integraation etu on kahden sovelluksen toisistaan eroaminen, jolloin DPAC:n koodin muokkaaminen ei vaatisi kääntämistä ja kehitys jatkuisi normaalisti. Huonona pidetään toisen verkkopalvelimen pystyttämistä kaikkine kuluineen.

Osallistujien tunnistamiseksi käytetään ensisijaisesti henkilötunnusta (HETU). Eventillan API-määrityksessä, HETU-attribuutti syntyy lomakkeen kentästä, joten se on dynaaminen. Attribuutti voidaan kuvailla JSON-tietoformaatilla seuraavasti:

- *items[i].data[i].name: ”Henkilötunnus (laskutusta varten)”*
- *items[i].data[i].value: "123456-XXXX".*

Listassa olevat *items*, *data*, *name* ja *value* ovat API:n attribuutteja. Osallistujan HETU löytyy *value*-attribuutista. Toisin sanoen HETU-tiedon määrittely ei ole yksiselitteinen, ja se voi vaikeuttaa tiedon saamista, mikäli lomakkeen kenttää muokataan. Tämän dynaamisen ratkaisun kiertämiseksi Eventilla mahdollistaa lomakekenttien merkitsemistä (tag), joka löytyy sitten esimerkiksi JSON-tietoformaatilla seuraavasti:

- *items[i].data[i].tag[”5058”]: ”hetu”*

Mikäli henkilötunnusta ei ole, suositellaan sähköpostiosoitetta, sillä attribuutti on kiinteä, eli se ei riipu lomakkeen rakenteesta. On suositeltava luoda Eventillassa uusi pohjalomake, jotta kaikki lomakkeet olisivat sisällöltään ja rakenteeltaan yhteneviä.

Integroinnin tavasta riippuen lisätään Pyramukseen tai DPAC:n käyttöliittymään osallistujien lista, josta valitaan siirrettävät osallistujat. Tämä toiminnallisuus on koulutussihteerin mukaan hyvä menetelmä, sillä henkilötunnus ja sähköpostiosoite tarkistetaan Eventillalla, ja tarkistuksen jälkeen suoritetaan

manuaalinen synkronointi. Tämä synkronointi voidaan automatisoida Eventillan API:n tiedon muokkauksen laukaisijan avulla, jolloin Eventilla API lähettää ilmoituksen Pyramukselle käyttäen edelleenkin API-tekniikkaa.

7 PÄÄTÄNTÖ

Kommunikaatio on eräs maailman tärkeimmistä asioista. Tiedostaen tai tiedostamatta siirretään monenlaisia dataa. Siirtäminen tapahtuu monella tavalla, muodolla ja kanavalla. Dataa kertyy jatkuvasti, mutta vain pieni osa siitä hyödynnetään. Rikkaus on sillä, joka ymmärtää datan tärkeyden. Tietojenkäsittelyn ydin on, että osaa järjestää, suodattaa, käsitellä sääntöjen mukaisesti ja lopuksi hyödyntää dataa. Organisaatioille datan ymmärtäminen on liiketoiminnan kasvun edellytys, mutta samalla datan hyödyntäminen vaatii myös tehokkaita menetelmiä ja käsittelyn osaajia.

Opinnäytetyöksi valitsin sovellusohjelmointirajapinnan, koska aihe on sekä mielenkiintoinen että tärkeä. Se esiintyy monessa tietotekniikan piirissä. API:n käyttö kasvaa dataekosysteemin muuttuessa, esimerkiksi massadatan tuomat mahdollisuudet kasvavat mobiililaitteiden ja muiden kevyiden laitteiden yleistymisen ansiosta, sillä dataa voidaan käyttää eri tilanteissa ja paikasta riippumatta. Tilaisuus käyttää opinnoista saatua taitoa tuli kohdalleni kuin tilauksesta, kun kuulin koulutussihteerin jatkuvasta kamppailusta tietojen manuaalisesta siirtämisestä, mikä oli kaukana digiloikasta: tehokkaat laitteet eivät riitä, vaan toimintatavat ja kestävät kehitykset ovat tärkeitä. Ohjelmoijan tehtävä on tuoda asiakkailleen toimivia sovelluksia ja palveluja, mutta myös turvallisia ympäristöjä. Sen tähden tietosuojan ja tietoturvan eri osa-alueet kannattaa ottaa huomioon jo suunnitelman alkuvaiheessa – määrityksien mukaisesti.

Tavoitteeni oli sekä kasvattaa taitojani ohjelmointirajapintojen parissa että lisätä tietomäärää ja osaamista ko. osa-alueesta. Toivon sen avulla pääseväni mukaan kasvavan tietotekniikan sovelluskehityksen eri osa-alueisiin. Ammattilaiseksi tuleminen vaatii tietojen hakemista tietotekniikan kirjallisuudesta sekä tekijänoikeuksien ja käyttöoikeuksien tietämystä. Ohjelmointikielten vastavuoroisuuden tunteminen edesauttaa oikeiden työkalujen ja työvälineiden valitsemista. Lisäksi sain suunnattomasti iloa mukanaolosta Pyramus-sovelluksen

kehittämisessä ja ympäristön kehityksessä toivoen työpanokseni tuovan myös myönteisiä vaikutuksia Ohjelmistokehitystiimille sekä liikelaitoksen toimintaan.

Kiitokseni Otavian johtaja-rehtorille, vararehtorille sekä Eventillan teknologiajohtajalle (CTO) ja muille henkilöille, jotka ovat auttaneet minua työssäni. Kiitän Xamkin opettajia ja erityisesti opettaja-ohjaajaani Janne Turusta arvokkaista neuvoista ja ohjeista opinnäyteraportin laatimiseen. Lopuksi haluan kiitää puolisoani ja ystäviäni kaikesta tuesta.

LÄHTEET

6Aika. 2017a. CKAN 2.7 TESTIDATA. Tilastokeskus / Vantaan kaupungin tietopalveluyksikkö. WWW-dokumentti. Saatavissa: https://generic-qa.dataportaali.com/data/sv/dataset/ckan-2-7-testidata [viitattu 15.2.2019].

6Aika. 2017b. GeoJSON testidata – Avoimen datan portaali. WWW-dokumentti. Saatavissa: https://generic-qa.dataportaali.com/data/fi/dataset/geojson-testidata [viitattu 15.3.2019].

Bootstrap. 2019. The most popular HTML, CSS, and JS library in the world. WWW-dokumentti. Päivitetty 18.3.2019. Saatavissa: https://getbootstrap.com/ [viitattu 27.4.2019].

Cenno. 2014. Mikä on API ja miksi se on SaaS ohjelmistossa niin tärkeä? Päivitetty 18.8.2017. Blogi. Saatavissa: https://www.cenno-app.com/blog/2014/05/mika-on-api-ja-miksi-se-on-saas-ohjelmistossa-niin-tarkea/ [viitattu 4.1.2019].

Coss. s.a. Avoin rajapinta. WWW-dokumentti. Saatavissa: https://coss.fi/avoimuus/avoin-rajapinta/ [viitattu 11.1.2019].

D'mello, B. J., Satheesh, M. & Krol, J. 2017. Web Development with MongoDB and Node. Build fast web applications for handling any kind of data. Third edition. Birmingham: Packt Publishing Ltd.

Digitaalinen Helsinki. s.a. Alfaversio (prototyyppi). Blogi. Saatavissa: https://digi.hel.fi/kehmet/menetelmalaari/alfaversio-prototyyppi/ [viitattu 3.5.2019].

EJS. s.a. Embedded JavaScript templating. WWW-dokumentti. Saatavissa: https://ejs.co/ [viitattu 26.4.2019].

Ekonoja, A., Lahtonen, T. & Mäntylä, J. 2005. HTTP-protokolla ja Apache. WWW-dokumentti. Päivitetty 14.1.2005. Saatavissa: http://appro.mit.jyu.fi/2004/syksy/www/luennot/luento9/#TOC2 [viitattu 6.1.2019].

EU:n tietosuoja-asetus (2016/679). 2016. EUROOPAN PARLAMENTIN JA NEUVOSTON ASETUS (EU) 2016/679. WWW-dokumentti. Saatavissa: https://eur-lex.europa.eu/legal-content/FI/TXT/HTML/?uri=CELEX:32016R0679&from=FI [viitattu 27.4.2019].

Express performance Best Practices Using Express in Production. s.a. Production best practices: performance and reliability. WWW-dokumentti. Saatavissa: https://expressjs.com/en/advanced/best-practice-performance.html#dont-use-synchronous-functions [viitattu 27.4.2019].

Express routing. s.a. Express routing. WWW-dokumentti. Saatavissa: https://expressjs.com/en/guide/routing.html [viitattu 25.4.2019].

Fielding, E. T. 2000. Representational State Transfer (REST). WWW-dokumentti. Saatavissa: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm [viitattu 6.1.2019].

Fredrich, T. s.a. What is REST Anyway. WWW-dokumentti. Saatavissa: https://www.restapitutorial.com/lessons/whatisrest.html [viitattu 6.1.2019].

GitHub. 2019. Otavanopisto / pyramus. WWW-dokumentti. Saatavissa: https://github.com/otavanopisto/pyramus [viitattu 15.6.2019].

Google Cloud. 2019. Cross-origin resource sharing (CORS). WWW-dokumentti. Päivitetty 18.4.2019. Saatavissa: https://cloud.google.com/storage/docs/cross-origin [viitattu 27.4.2019].

Kaario, K. 2002. TCP/IP-verkot. E-Kirja. Jyväskylä: Docendo Finland oy. Saatavissa: https://kaakkuri.finna.fi [viitattu 24.4.2019].

Keig, A. 2013. Advanced Express Web Application Development. Your guide to building professional real-world web applications with Express. E-Kirja. Birmingham: Packt Publishing Ltd. Saatavissa: https://kaakkuri.finna.fi/ [viitattu 25.4.2019].

Kotkanen, H. 2016. Avoimen datan hyödyntäminen –koulutus. PDF-dokumentti. Saatavissa: https://www.hel.fi/hel2/tietokeskus/data/dokumentit/koulutusmateriaali/Hyodyntamiskoulutus_Henri_Kotkanen.pdf [viitattu 4.1.2019].

Microsoft. 2019. Evästeiden poistaminen ja hallinta. WWW-dokumentti. Päivitetty 29.1.2019. Saatavissa: https://support.microsoft.com/fi-fi/help/17442/windows-internet-explorer-delete-manage-cookies [viitattu 27.4.2019].

Microsoft. s.a. Csv-tiedostojen luominen tai muokkaaminen Outlookiin tuontia varten. WWW-dokumentti. Saatavissa: https://support.office.com/fi-fi/article/-csv-tiedostojen-luominen-tai-muokkaaminen-outlookiin-tuontia-varten-4518d70d-8fe9-46ad-94fa-1494247193c7 [viitattu 3.5.2019].

MDN web docs. 2019. Working with JSON. WWW-dokumentti. Päivitetty 4.4.2019. Saatavissa: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON [viitattu 26.4.2019].

MyHelsinki Open API. s.a. Terms of Use for the API. WWW-dokumentti. Saatavissa: http://open-api.myhelsinki.fi/doc [viitattu 2.1.2019].

Node.Js. 2019. About Node.js®. WWW-dokumentti. Päivitetty 25.4.2019. Saatavissa: https://nodejs.org/en/about/ [viitattu 25.4.2019].

Node.Js. s.a. Node.js v12.0.0 Documentation. WWW-dokumentti. Saatavissa: https://nodejs.org/api/crypto.html#crypto_class_hmac [viitattu 27.4.2019].

Npm. 2019. Express-session. WWW-dokumentti. Päivitetty 11.4.2019. Saatavissa: https://www.npmjs.com/package/express-session [viitattu 27.4.2019].

Pitkänen, H. 2016. Rajapintojen käyttö tiedonsiirrossa. Pro gradu -kokoelma. Itä-Suomen yliopisto. Saatavissa: http://epublications.uef.fi/pub/urn_nbn_fi_uef-20170032/urn_nbn_fi_uef-20170032.pdf#page=22&zoom=100,0,353 [viitattu 5.1.2019].

Redhat. 2017. What are APIs? WWW-dokumentti. Päivitetty 11.1.2019. Saatavissa: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces [viitattu 11.1.2019].

Sanastokeskus. s.a. Rajapinta. WWW-sanakirja. Saatavissa: http://www.tsk.fi/tepa/fi/haku/rajapinta [viitattu 1.1.2019].

Seravo. 2013. COSSin webinaari: liiketoimintaa Pyramus-järjestelmällä. WWW-dokumentti. Saatavissa: https://coss.fi/tapahtumat/webinaari-liiketoimintaa-pyramus-jarjestelmalla/ [viitattu 15.6.2019].

W3C. 2010. Same Origin Policy. WWW-dokumentti. Päivitetty 6.1.2010. Saatavissa: https://www.w3.org/Security/wiki/Same_Origin_Policy [viitattu 27.4.2019].

ZEF. 2014. Eventilla: Tapahtumat sisältömarkkinoinnin kärkenä. Blogi. Saatavissa: https://blog.zef.fi/eventilla-tapahtumat-sisaltomarkkinoinnin-karkena [viitattu 27.1.2019].

Zumeran, R. 2017. The History of APIs and How They Impact Your Future. Blogi. Saatavissa: https://www.openlegacy.com/blog/the-history-of-apis-and-how-they-impact-your-future [viitattu 3.1.2019].

KUVALUETTELO