Tampereen ammattikorkeakoulu

Reaaliaikainen full stack äänes-tyssovellus

Jennina Färm

OPINNÄYTETYÖ
Huhtikuu 2022

Tietojenkäsittelyn tradenomi
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn tradenomi
Ohjelmistotuotanto

FÄRM, JENNINA:
Reaaliaikainen full stack äänestyssovellus

Opinnäytetyö 51 sivua, joista liitteitä 0 sivua
Huhtikuu 2022

Sosiaalisen paritanssin kilpailujen tuomaroiminen on aina ollut hyvin haastava ja työläs prosessi. Jo yhdeksän tuomarin tulosten käsittely vie aikaa. Mitä jos tuomareina toimisi koko yleisö äänestäen omaa järjestystään finaalien sijoituksista. Opinnäytetyön tavoitteena oli selvittää, kuinka luodaan reaaliaikainen äänestyslomake ja digitaalinen ratkaisu sijoitusten käsittelyyn ja antamiseen.

Opinnäytetyön tarkoituksena oli kehittää sovellus, joka tarjoaa mahdollisuuden luoda äänestyslomake reaaliaikaisesti. Sovelluksen tuli mahdollistaa äänestyksen kohteen lisääminen siten, että se tulee näkyviin äänestäjälle heti, kun se luodaan. Tällöin äänestettävät kohteet lisääntyvät äänestäjälle äänestyksen edetessä. Lisäksi sovelluksen tuli automatisoida tulosten laskenta ja prosessoida laskennan konfliktit.

Opinnäytetyön tuloksena kehitettiin full stack -verkkosovellus, jolla mahdollistettiin käyttö eri alustoilla ilman ylimääräisen sovelluksen asentamista. Palvelinsovellus tarjoaa REST-rajapinnan vuorovaikutukseen. Kehityksessä käytettiin React ja Node.js -teknologioita sovelluksen rakentamiseksi, ja se toimii reaaliajassa WebSocket -protokollaa hyödyntämällä. Opinnäytetyön raportissa esitellään pilvipalveluita, käytettyjä teknologioita, ja käsitellään teoreettisesti reaaliaikaisuuden vaihtoehtoja verkkoalustalla.

Tuloksena nähdään, että nykypäiväisessä verkkosovelluksen reaaliaikaisuudessa on hyvä suosia push-tekniikoita pull-tekniikoiden sijaan. Pull-tekniikat joko käyttävät kaistaleveyttä turhaan tai kuormittavat huomattavasti palvelinta käsitellessään ruuhka-ajan käyttäjämääriä. Kaikilla protokollilla on kuitenkin omat käyttötarkoituksensa. Toistaiseksi WebSocket-protokolla tarjoaa monipuolisimman kommunikointitavan kehitetyn sovelluksen tarpeisiin.

Asiasanat: reaaliaikaisuus, web-tekniikat, verkkosovellus

ABSTRACT

Tampere University of Applied Sciences
BBA of Information Business
Software Production

FÄRM, JENNINA:
Real-time full stack voting application

Bachelor's thesis 51 pages, appendices 0 pages
April 2021

The purpose of the thesis was to develop an application that offers possibility to create a voting form in real-time. The application should allow the topics of a poll to be added so that it appears to the voter as soon as it is created. The number of items to be voted on increases for the voter as the creating the form progresses. The application also had to automate the calculation of results.

As a result of the thesis, a full stack web application was developed, which enabled use on different platforms without installing an application. The server application provides a REST interface for interaction. The development used React and Node.js technologies to build the application and it works in real time using the WebSocket protocol. The thesis report presents cloud services, technologies used and theoretically discusses real-time options on a web platform.

As a result, in creating a contemporary real-time web application, it is good to favor push technologies over pull technologies. Pull technologies either use bandwidth unnecessarily or put a significant strain on the server when handling peak time usage volumes. So far, the WebSocket protocol provides the most versatile way to communicate the needs of the developed application.

Key words: real-time connection, web technologies, web application

SISÄLLYS

1 JOHDANTO

Internet on muodostunut olennaiseksi osaksi jokapäiväistä elämää. Ostokset, pankkitoiminta, viestintä ja viihde ovat suurimmaksi osaksi riippuvaisia internetistä, joten ihmiset omaksuneet sen suureksi osaksi elämäänsä. Niin sanotun Internet of Things (IoT) kasvun myötä yhä useammat laitteet ovat yhteydessä verkkoon, jossa niitä voi käyttää etänä. Tämän yhteyden mahdollistavat useat tekniikat. (Pollard 2019.)

Internet ja Word Wide Web monesti sekoitetaan keskenään. Internet on kokoelma julkisia tietokoneita, jotka on linkitetty Internet-protokollan kautta viestien reitittämiseen. Se koostuu monista palveluista, joista Word Wide Web on vain yksi. Sähköposti, tiedostojen jakaminen ja Internet-puhelut ovat toisia tunnettuja internetin palveluita. (Pollard 2019.)

Opinnäytetyö lähti liikkeelle tarpeesta luoda verkkosovellus, joka mahdollistaa reaaliaikaisen äänestyksen luomisen ja toteuttamisen. Täten opinnäytetyöni perustuu verkkosovelluksen vaatimiin teknologioihin ja siihen, mitkä mahdollistavat reaaliaikaisuuden sovelluksessa.

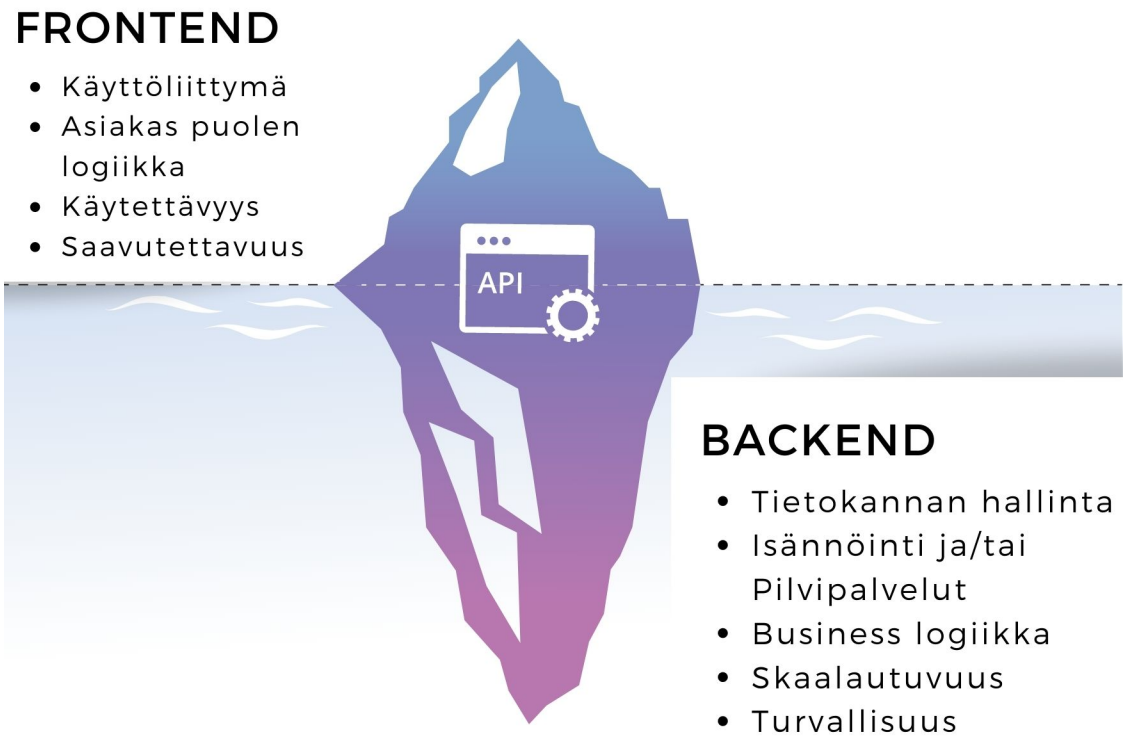
Verkkoselaimesta on tullut yksi suurimmista väylistä sisällölle ja palveluille. Selain on käytettävissä melkein kaikissa laitteissa, jotka ovat kytketty internettiin käyttötarkoituksesta riippumatta. Ihmiset käyttävät selaimia uutisten lukemiseen, yhteyden pitoon, pankkitilin hallintaan, palveluiden ajanvaraukseen, elokuvien ja videoiden katselemiseen sekä kaupan käyntiin vain nimetäkseen muutamia. Se, että selain on yleisesti saatavana, intuitiivinen käyttää ja yhteneväinen ovat ehdottomia etuja suoritettaviin ohjelmiin verrattuna. Lisäksi selain on laitteistosta ja käyttöjärjestelmästä riippumaton alusta sovellukselle. (Karla & Tarnawski. 2019. 1.)

Sovelluksia on miljoonittain ja harvoja puhelimelle tai tietokoneelle ladattuja sovelluksia tullaan käyttämään päivittäin. Sen vuoksi verkkosovellus on myös hyvin yksinkertainen ja looginen valinta, sillä tällöin loppukäyttäjän ei tarvitse asentaa uutta sovellusta viemään tilaa omalta laitteeltaan.

2 TEKNINEN TAUSTA

Sovelluskehityksessä kehittäminen monesti jaetaan kahteen osaan, frontendiin ja backendiin. Riippuen kehitystiimin koosta työtä jaetaan samalla tavoin. Kehittäjä osallistuu, joka backend tai frontend kehitykseen. Kuitenkin on myös tapauksia, jossa tiimi tai projekti on niin pieni, että siinä ei kyetä jaottelemaan. Tällaisissa tilanteissa puhutaan full-stack kehittämisestä. (Filipova & Vilão 2018.) Sillä tämän projektin kehittäjä tiimiin kuuluu vain yksi henkilö, on tämäkin projekti full-stack kehittämistä. Kuitenkin olen jakanut työstämisen selkeästi näihin kahteen osa-alueeseen projektihallinnan selkeyttämiseksi.

Kuvio 1 havainnollistaa frontendin ja backendin suhdetta keskenään, ja mitkä ovat kummankin kehittämisen vastuualueet. Frontend ja backend kehityksen tuotokset kommunikoivat keskenään API:n välityksellä. Tulevissa alaluvuissa käsitellään kutakin kehittämispuolta hieman tarkemmin.



KUVIO 1. Frontend ja backend kehityksen vastuualueet.

2.1 Backend kehitys

Backend kattaa sovelluksen taustajärjestelmän. Taustajärjestelmä luo ohjelmiston osuuden, joka on vastuussa käyttöliittymäsovellusten pyyntöjen vastaanottamisesta ja niiden käsittelystä. Backend toimii niille omistetuilla palvelimilla, joita isännöidään tyypillisesti pilvipalveluissa tai palvelintarjoajilla. Pilvipalveluiden ja palvelintarjoajia ovat esimerkiksi Amazon-verkkopalvelut, Google Cloud -alusta sekä Microsoft Azure Cloud -palvelu. (Filipova & Vilão 2018.) Kehitettävä sovellus on isännöity Herokun tuottamilla palveluilla.

On olemassa useita erilaisia backendin verkkopalveluita. Näitä ovat esimerkiksi RESTful, WSDL ja SOAP. (Filipova & Vilão 2018.) Tässä opinnäytetyössä keskityn RESTful tekniikan käyttöön.

Backend-kehitys keskittyy toimintojen mahdollistamiseen. Taustajärjestelmä on siis vastuussa esimerkiksi tiedon suodattamisesta ja luetteloimisesta (haku), uuden tiedon tallentamisesta sekä sen muokkaamisesta ja poistamisesta käyttöliittymäsovellusten pyyntöjen perusteella. (Filipova & Vilão 2018.)

Backend-kehityksen suurimpia haasteita on suorituskyvyn parantaminen eli optimointi. Taustajärjestelmän tulee käsitellä monesti useita pyyntöjä samanaikaisesti, sillä käyttäjiä harvemmin on vain yksi. Optimointiin kuuluu esimerkiksi, miten tietoja noudetaan, miten tietoa järjestetään ja tietojärjestelmän suunnitteleminen niin, että se selviää tulevista muutoksista sekä miten tietoa integroidaan ulkoisiin palveluihin ja tietokantoihin. (Filipova & Vilão 2018.)

Toinen suuri kokonaisuus taustajärjestelmälle on todennus ja valtuutus. On määriteltävä tarkasti esimerkiksi julkiset ja yksityiset päätepisteet, käyttäjärooli ja käyttöoikeuksien peruuttaminen. (Filipova & Vilão 2018.)

2.2 Frontend kehitys

Frontend kehitys on vastuussa sovelluksen osasta, joka näkyy käyttäjälle. Yleisesti ottaen käyttöliittymällä tarkoitetaan selaimessa toimivaa verkkosovellusta.

Kuitenkin mikä tahansa sovellus, jolla on graafinen- tai komentorivikäyttöliittymä, voidaan määritellä käyttöliittymäsovellukseksi. Toisin sanoen käyttöliittymäsovellus on ohjelmisto, jota käyttäjä pystyy käyttämään, eikä se ole sidonnainen selaimeen, mobiiliin tai vaikka tv-sovellukseen. (Filipova & Vilão 2018.)

Frontend-kehityksen voi jakaa kahteen päähaaraan: esitykseen ja logiikkaan. Esityksellä tarkoitetaan sitä, mitä käyttäjä näkee eli käyttöliittymän. Käyttöliittymään kuuluu, kuinka elementit renderöidään ja kuinka niiden kanssa ollaan vuorovaikutuksessa. Logiikka puolestaan tekee ohjelmistosta sovelluksen. Logiikka esimerkiksi hakee tietoa ja muokkaa sen esitettävään muotoon, käsittelee pyyntöjä ja tiloja sekä validoi syötettyä tietoa. (Filipova & Vilão 2018.)

3 SUUNNITTELUN LÄHTÖTILANNE

3.1 Groove Connection Oy

Groove Connection Oy on kohtalaisen nuori osakeyhtiö. Se perustettiin vuonna 2020 alkukeväästä. Groove Connectionin tavoite on edistää Suomen west coast swing osaamista sekä tuoda sitä näkyviin sekä kotimaassa että kansainvälisesti. Yhtiö pyrkii tuottamaan laadukkaita ja ajankohtaisia tapahtumia ja palveluita tuoden aallon harjalla olevia valmentajia Suomeen, järjestämällä niin virallisia kuin epävirallisia kilpailuita sekä tuottamaan työtä kotimaisille tanssinopettajille. Yhtiö omistaa oikeudet Suomen toiseen viralliseen pistekilpailutapahtumaan, Finnfestiin, joka kattaa parhaimmillaan lähes 300 osallistujaa vuosittain.

Keväällä 2022 Groove Connection on järjestämässä Swinghearts tapahtuman, joka sisältää epäviralliset west coast swing kilpailut. Yhtiö on antanut toimeksiannon sovellukseen, joka mahdollistaa yleisöäänestyksen kilpailun finaaleissa.

3.2 Sosiaalisen paritanssin kilpailujärjestelmä

Sosiaalisen paritanssi kilpailuita kutsutaan monesti Jack and Jill kilpailuiksi, jonka muotoa pääasiassa swing tanssin eri tyylisuunnat käyttävät (Street smart swing 2022). Jack Carey on nimitetty Jack and Jill termin kehittämisestä 1950-luvulla. Perusperiaatteena on kilpailumuoto, jossa tanssijat eivät tiedä sekä musiikkia tai tanssipariaan etukäteen. (WSDC 2022.) Tällä tavalla pyritään testaamaan mahdollisimman aidosti sosiaalisen tanssin taitoja, eikä pelkästään koreografisia taitoja ja esiintyjyyttä (Street smart swing 2022). Nykypäivänä on myös noussut termi Mix and Match, mahdollistaen sukupuolettoman erottelemattomuuden.

Finaaleissa tuomarointiin käytetään sijoituspistejärjestelmää (Relative Placement Scoring System). Tässä jokaisen tuomarin antamilla pisteiden määrällä ei ole merkitystä. Sen sijaan pisteiden perusteella annetaan sijoitus ensimmäisestä sijasta parien maksimimäärään. Tällä pyritään tasa-arvoisuuteen siinä mielessä, että sijoitus on sijoitus huolimatta onko se kolme pistettä vai yhdeksän pistettä

enemmän kuin toisella. Jokaisella tuomarilla on siis erikseen sijoitusrivinsä ja samaa sijoitusta ei yksi tuomari voi antaa kahdelle eri parille. (WSDC 2021.)

Kun tuomarit ovat antaneet sijoituksensa, ruvetaan etsimään sijoituksista enemmistöjä. Kuten monissa muissakin äänestyksessä, kun yli 50 prosentilla on sama sijoitus, enemmistö saavutetaan. Koska parillinen määrä jakaantuu helposti täysin tasan 50 prosenttiin, on parempi, jos tuomareita on pariton määrä. Jos on siis 7 tuomaria, enemmistö on 4. Ensimmäiseen sijaan tarvitaan siis vähintään 4 ensimmäistä sijaa. Jos ei ole, katsotaan sekä ensimmäiset että toiset sijoitukset. Eli kenellä on vähintään 4 tai eniten 1.–2. sijoituksia. Tätä jatketaan kunnes sijoitukset ovat jaettu. (WSDC 2021.)

Jos enemmistö saavutetaan toisen parin kanssa samaan aikaan ja heillä on saman verran kyseisen tason sijoituksia, katsotaan sijoitusten laatua. Laatu voidaan ratkaista matemaattisesti laskemalla enemmistöön vaikuttavat sijoitukset yhteen. Se kenen parin summa on pienin sijoittuu ylemmäksi. Jos myös summat ovat kuitenkin tasan, analysoidaan seuraavan tason sijoitukset. Jos tasatilanne tulee sekä enemmistössä, että laadussa, analysoidaan laatua vielä tarkemmin. Tällöin verrataan vain näitä kahta paria ja etsitään kumpi on saanut enemmän parempia tuloksia tuomarikohtaisesti. (WSDC 2021.)

4 PILVIPALVELU JA SOVELLUKSEN ISÄNNÖINTI

Sovelluksen saamiseksi julkiseen verkkoon tulisi perinteisesti ostaa fyysisiä palvelimia ja suoritettava sovellukset niissä. Tällöin tulisi pitää huolta koko palvelin rakenteesta aloittaen käyttöjärjestelmän ja järjestelmäkirjastojen ajan tasalla pitämisestä päätyen varmistamaan, että vialliset laitteet vaihdetaan. Lisäksi tulisi ottaa huomioon myös palvelimien ylläpito sähkökatkoksen aikana. Ilman suurta budjettia ja tiimiä, on vaikea keskittyä itse sovelluksen kehittämiseen. (Lyu 2021.)

Apuun tulee siis kolmannen osapuolen pilvipalvelut. He hallitsevat palvelimia puolestasi ja tarjoavat erilaisia abstraktiotasoja, jotta kehittäjä voi keskittyä itse sovellukseen. Niin sanottu palvelimeton tietojenkäsittely ajaa tämän idean äärimmilleen. Tällöin tulee huolehtia vain metodeista, jotka hoitavat liiketoimintalogiikkaa. Palveluntarjoaja puolestaan huolehtii laitteiston, käyttöjärjestelmän ja kielen ajoajan. Siihen voi lisäksi yhdistää hallinnoidun tietokannan, viestijonot ja tiedostojen tallennustilan, joita palveluntarjoaja hallitsee täysin. (Lyu 2021.)

Pilvipalvelulle on tullut ajan saatossa monenlaisia määritelmiä. Tässä opinnäytetyössä pilvipalvelun määritelmänä käytetään National Institute of Standards and Technologyn (NIST) määritelmää. Määritelmän mukaan pilvipalvelu on malli, joka mahdollistaa kaikkialla läsnä olevan, kätevän, tarvittaessa saatavilla olevan verkkoyhteyden jaettuun joukkoon konfiguroitaviin laskentaresursseihin. Resurssit voivat olla esimerkiksi verkkoja, palvelimia, tallennustilaa, sovelluksia tai palveluita, jotka voidaan varustaa ja vapauttaa nopeasti minimaalisella hallintatyöllä tai palveluntarjoajan vuorovaikutuksella. Tämä pilvimalli koostuu viidestä olennaisesta piirteestä, kolmesta palvelumallista ja neljästä käyttöönotto tavasta. (Meil & Grance 2011, 6.)

Pilvi-infrastruktuurilla tarkoitetaan kokoelmaa laitteistoja ja ohjelmistoja, jotka mahdollistavat pilven viisi olennaisinta ominaisuutta. Infrastruktuurin voidaan käsittää sisältävän sekä fyysisen- että abstraktin kerroksen. Fyysinen kerros koostuu laitteistoresursseista, jotka ovat tarpeen tarjottavien pilvipalveluiden tukemiseksi. Laitteistoresurssit tyypillisesti sisältävät palvelin-, tallennus- ja verkkokomponentit. Abstrakti kerros puolestaan koostuu ohjelmistosta, joka on asennettu fyysiseen kerrokseen, joka ilmaisee pilven keskeiset ominaisuudet.

Käsitteellisesti abstrakti kerros istuu fyysisen kerroksen päällä. (Meil & Grance 2011, 6.)

4.1 Pilvipalvelun rakenne

Pilvipalvelulla on kolme suosittua palvelumallia, infrastruktuuri palveluna (IaaS), alusta palveluna (PaaS) ja ohjelmisto palveluna (SaaS). (El Kafhali & Salah 2018, 1.)

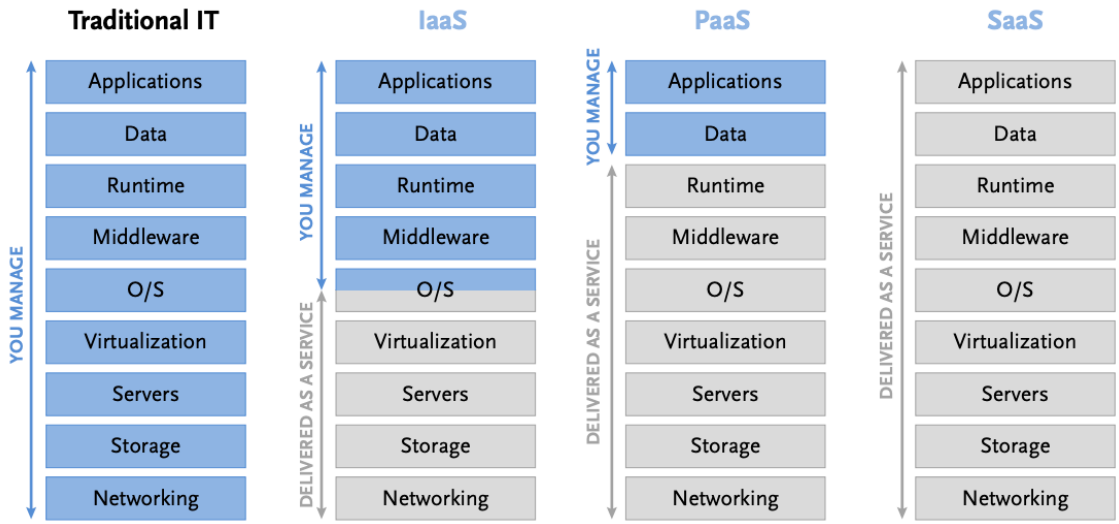
TAULUKKO 1. Pilvipalveluiden kolme mallia. Muokattu. (Meil & Grance 2011, 6-7.)

Lyhenne	Nimi	Kuvaus
IaaS	Infrastructure as a service	Kuluttajalla on oikeus hallita tallennustilaa, verkkoa ja muita perustavanlaatuisia laskentaresursseja, joita kuluttuja voi käyttöönottaa ja ylläpitää ohjelmistojaan
PaaS	Platform as a service	Alusta, johon on esiasennettu käyttöjärjestelmä. Kuluttajat voivat luoda ja ottaa käyttöön sovelluksia pilvialustalle.
SaaS	Software as a service	Mahdollistaa kuluttajalle pilvessä toimivien ohjelmistojen käytön. Ohjelma on käytettävissä ohuen asiakasliittymän, kuten selaimen, kautta. Kuluttaja ei hallitse tai valvo taustalla olevaa pilviinfrastruktuuria, kuten verkkoa, palvelimia tai tallennustilaa.

Esimerkki infrastruktuurista palveluna on Amazonin EC2, joka tarjoaa virtualisoitua laitteistoa, tallennustilaa ja fyysisiä laitteita internetissä. Esimerkki ohjelmistosta palveluna on Customer Relationship Management (CRM), joka tarjoaa ohjelmistoja ja isännöityjä sovelluksia internetin kautta. Esimerkki alustasta palveluna toimii Google App Engine (GAE), joka tarjoaa kykyä ottaa käyttöön sovelluksia, jotka ovat luotu pilvipalveluntarjoajan tukemilla ohjelmistokielillä, kirjastoilla, palveluilla ja työkaluilla. Tällöin palvelun asiakas ei hallinnoi infrastruktuuria, vaan hallinnoi ainoastaan käytössä olevia sovelluksia. (El Kafhali & Salah 2018, 1.)

Jos mietitään vielä palvelumallien tuomia hyötyjä ja käyttötarkoituksia, on hyvä tarkastella alla olevaa kuviota 2. Siitä voidaan helposti huomioida, että SaaS tarjoaa käytännössä täysin valmiin sovelluksen asiakkaalle. PaaS puolestaan

tarjoaa alustan, jolle itse kehitetty sovellus ja sen data on kehittäjän vastuulla. Käyttämällä IaaS tyyppistä palvelua, kehittäjällä ei vielä tarvitse olla omia laitteistoja sovelluksen julkaisemista varten, mutta on vastuussa suuremmasta kokonaisuudesta. Perinteisessä sovelluskehityksessä palvelun tuottamiseksi, kehittäjällä tulee olla sovelluksen ja datan käsittelyn lisäksi kaikki tietoverkoista, muistista ja palvelimista lähtien.



KUVIO 2. Pilvipalvelumallien ja perinteisen kehityksen hallintamäärä. Havainnollistaa eri pilvipalvelumalleihin vaadittua hallinta määrää sekä ottaa huomioon myös perinteisen ohjelmistokehityksen. (Harms & Yamartino 2010, 11.)

4.2 Käyttöönotto (deployment) tavat

NIST määrittelee neljä olemassa olevaa käyttöönottomallia. Yksi niistä on yksityinen pilvi. Pilvi-infrastruktuuri on tarkoitettu tällöin täysin yhden yrityksen tai organisaation käyttöön, joka koostuu useasta kuluttajasta, kuten liiketoimintayksiköstä. Pilvi-infrastruktuuri voi olla omistettu, hallinnoitu sekä liikennöity itse organisaation, kolmannen osapuolen tai näiden kahden yhdistelmän toimesta. Lisäksi se voi joko sijaita organisaation tiloissa tai sen ulkopuolella. (Meil & Grance 2011, 7.)

Yhteisön pilvi-infrastruktuuri on puolestaan suunniteltu yksinomaiseen käyttöön tietylle kuluttajayhteisölle, joka koostuu monesta yrityksestä ja/tai organisaatiosta. Heitä yhdistää yhteiset huolenaiheet, kuten turvallisuusvaatimukset, politiikka tai vaatimustenmukaisuus. Se voi olla omistuksessa, jota hallinnoi yksi tai

useampi yhteisön organisaatio, kolmas osapuoli tai jokin niiden yhdistelmä. (Meil & Grance, 2011 7.)

Julkisen pilven infrastruktuuri on tarkoitettu suuren yleisön avoimeen käyttöön. Se voi olla yrityksen, akatemian tai valtion organisaation omistama, hallinnoima ja ylläpitämä, tai jokin niiden yhdistelmä. Se on olemassa pilvipalveluntarjoajan tiloissa. (Meil & Grance, 2011 7.)

Hybridipilvi on yhdistelmä kahdesta tai useammasta erillisestä pilvi-infrastruktuurista, jotka säilyvät ainutlaatuisina kokonaisuuksina. Infrastruktuurit ovat kuitenkin sidottuja toisiinsa standardoidulla tai patentoidulla tekniikalla, joka mahdollistaa datan ja sovellusten siirrettävyyden. (Meil & Grance, 2011 7.)

4.3 Pilvipalvelun piirteet

On monta syytä miksi palvelut vaihtavat käyttämään pilvipalveluita perinteisesti isännöityjen alustojen sijaan. Isoimmat syyt valita pilvipalvelu on raha, helppokäyttöisyys ja joustavuus.

Helppokäyttöisyyteen liittyen yksi olennaisimmista ominaisuuksista on on-demand itsepalvelu. Kuluttujan on helppo yksipuolisesti varustaa laskentaominaisuuksia, kuten palvelinaikaa ja verkkotallennustilaa. Kaiken pystyy tekemään oman tarpeen mukaan automaattisesti ilman vuorovaikutusta jokaisen palveluntarjoajan edustajan kanssa. (Meil & Grance, 2011 6.)

Toinen helppokäyttöisyyteen liittyvistä ominaisuuksista on laaja verkkoyhteys. Ominaisuudet ovat saatavilla verkon kautta ja niitä käytetään hyödyntäen vakioituja mekanismeja, jotka edistävät heterogeenisten asiakasalustojen, kuten matkapuhelimien, tablettien, kannettavien tietokoneiden tai työasemien, käyttöä. (Meil & Grance, 2011 6.)

Pilvipalvelut ovat tunnettuja ja haluttuja nopeasta joustavuudestaan. Ominaisuudet voidaan joissain tapauksissa joustavasti varustaa ja vapauttaa automaattisesti skaalaten nopeasti isompaan ja pienempään kysynnän tarpeiden mukaan.

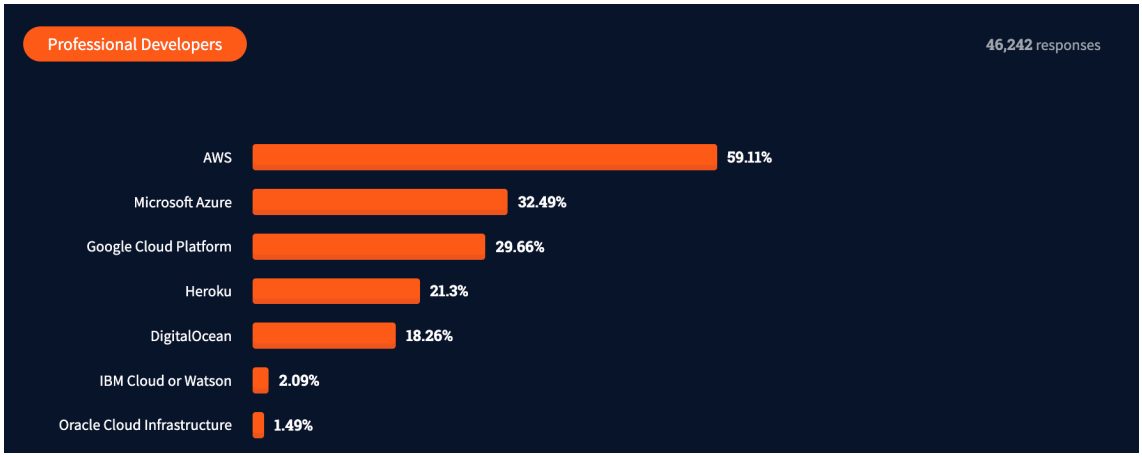
Kuluttajalle varustukseen käytettävissä olevat ominaisuudet näyttävät usein olevan rajoittamattomat ja niitä voidaan ottaa käyttöön minkä määrän hyvänsä ajasta riippumatta. (Meil & Grance, 2011 6.)

Joustavuutta lisätään myös resursseja yhdistämällä. Palveluntarjoajan laskentaresurssit ovat yhdistetty palvelemaan useita kuluttajia käyttämällä usean vuokralaisen mallia eri fyysisten ja virtuaalisten resurssien kanssa. Resurssit ovat dynaamisesti valittuja ja ne siirretään kuluttajien kysynnän mukaan. Kuluttaja on riippumaton fyysisestä sijainnista siinä mielessä, ettei asiakkaalla ole yleensä tarkkaa hallintaa tai tietoa toimitettujen resurssien sijainnista. Sijainti kuitenkin voidaan määritellä abstraktion korkeimmalla tasolla, kuten maa, osavalta tai tietokeskus. Esimerkkejä kyseessä olevista resursseista ovat varastointi, käsittely muisti ja verkon kaistanleveys. (Meil & Grance, 2011 6.)

Rahaa säästetään mitatun palvelun avulla. Pilvijärjestelmät ohjaavat ja optimoivat resurssien käyttöä automaattisesti hyödyntämällä mittausominaisuuksia jollakin palvelutyyppiin sopivalla abstraktio tasolla, kuten tallennus, käsittely, kaistaleveys tai aktiiviset käyttäjätilit. Resurssien käyttöä voidaan seurata, valvoa ja raportoida, mikä tarjoaa avoimuutta ja läpinäkyvyyttä sekä palveluntarjoajalle että käytetyn palvelun kuluttajalle. (Meil & Grance, 2011 6.)

4.4 Alusta palveluna (PaaS)

Tässä luvussa avaan vielä alustaa palveluna tarkemmin, sillä päädyin käyttämään projektissani Herokun tarjoaa pilvipalvelua, joka perustuu PaaS:iin. Lisäksi käyn läpi suurimmat alustapalvelun tuottajat, joka auttaa ymmärtämään valintaani Herokuun liittyen. Heroku kuvailee valintaa alusta- ja infrastruktuuripalvelun välillä asunnon hankkimiseen: Haluatko muuttaa täysin valmiiseen huoneistoon vai rakentaa alusta-alkaen oman kodin (2022).



KUVIO 3. Suosituimmat pilvipalvelut 2021 StackOverFlow:n kyselytutkimuksessa. Havainnollistaa eri pilvipalveluiden suosiota verrattuna toisiinsa. Heroku on ammattikehittäjien keskuudessa neljänneksi suosituin palvelu (Stackoverflow 2021).

PaaS kuvataan parhaiten kehitysympäristöksi, joka isännöin kolmannen osapuolen infrastruktuuria, jonka tarkoituksena on helpottaa uusien sovellusten nopeaa suunnittelua, testausta ja käyttöönottoa. Alustapalvelu ympäristöjä käytetään usein sovellusten ”hiekkalaatikkona”, joissa kehittäjät voivat vapaasti luoda ympäristössä, jossa resurssien kustannukset ovat huomattavasti pienemmät verrattuna oman infrastruktuurin ylläpitämiseen. PaaS palvelumallin käyttäminen eliminoi näin ollen kalliiden infrastruktuurien rakentamisen ja purkamisen tarpeen, joka tyypillisesti näkyy useimmissa yrityskehitysympäristöissä. (Williams 2012.)

Kun otetaan huomioon uusien älypuhelinsovellusten lisääntynyt kysyntä, ei ole ollenkaan yllätys, että PaaS:lla on pilvipalveluiden palvelumalleista suurin kasvuvauhti. (Williams 2012.)

Osa merkittävimmistä alustan tarjoajista ovat AWS Elastic Beanstalk, MS Azure, Heroku, Google App Engine ja AppScale. AWS Elastic Beanstalk on pilvipalvelu nimenomaan verkkosovellusten käyttöönottoon ja skaalaukseen. Se sallii käyttäjän tehdä automaattisesti käyttöönottoja skaalautuvasti, kuitenkin säilyttäen samalla täyden hallinnan AWS resursseihin, joita tarvitaan käyttöönottoon mahdollistamiseksi. (Danielsson, Postema & Munir 2021.)

MS Azure on Microsoftin luoma pilvilaskentapalvelu sovellusten ja palveluiden rakentamiseen, testaamiseen ja käyttöönottoon, jota hallitaan Microsoftin tietokeskuksia käyttäen. (Danielsson ym. 2021.)

Google App Enginen avulla kehittäjät voivat rakentaa skaalautuvia verkko- ja mobiili backendejä millä tahansa ohjelmointikielellä. AppScale puolestaan on avoimen lähdekoodin ohjelmistoinfrastuctuuri, joka tarjoaa automaattisen käyttöönoton, konfiguroinnin ja skaalautuvuuden. (Danielsson ym. 2021.)

Herokun tuottama alustapalvelu, joka perustuu hallittuun konttijärjestelmään integroiduilla datapalveluilla ja tehokkaalla ekosysteemillä. Se on optimoitu nykyaikaisten sovellusten käyttöönottoon ja käyttämiseen. Herokun kehittäjäkokemus on sovelluskeskeinen lähestymistapa ohjelmistotoimitukseen, joka on integroitu nykypäivän suosituimpiin kehittäjätyökaluihin ja työnkulkuun. (Heroku 2022.) Palvelu ajaa sovelluksia virtuaalisäiliössä, jotka suoritetaan luotettavassa ajonaikaisessa ympäristössä. Heroku kutsuu näitä säiliöitä Dynoiksi. Heroku mahdollistaa kehittäjän skaalata sovelluksen välittömästi, joko lisäämällä dynojen määrää tai muuttamalla dynon tyyppiä, jossa sovellus toimii. (Danielsson ym. 2021.)

Heroku lisäksi tukee versionhallintaa, Gitiä. Herokun Git-palvelin käsittelee sallituilta käyttäjiltä tulevat sovelluspäivitykset. (Scott 2021.) Herokun valintaan vaikutti myös se, että se oli tuttu alusta entuudestaan ja hyvin yksinkertainen käyttää, sillä palvelut on rajattu täysin alustapalvelun ympärille. AWS oli minulla toisena vaihtoehtona harkinnassa, mutta lopulta hintapolitiikan epäselkeys ja palveluiden laajuus sai minut valitsemaan yksinkertaiselle ja pienelle sovellukselle yksinkertaisemman ja helpomman vaihtoehdon.

4.5 Ohjelmisto palveluna (SaaS)

Ohjelmisto palveluna on pilvipalvelumalli, jonka useammat ihmiset tuntevat, vaikka he eivät itse kokosi ymmärtävänsä pilvestä paljoa. SaaS yksinkertaisesti sanottuna on kyky käyttää ohjelmistoa, jonkun muun infrastruktuurissa. (Williams 2012.) Sillä kehittämäni sovellus isännöidään pilvipalvelussa, käsittää se

ohjelmiston palveluna määritelmän. Sukelletaan nyt hieman syvemmälle, mitä ohjelmisto palveluna oikeasti pitää sisällään.

Ohjelmisto palveluna (Saas) on suosituin pilvipalvelumalli, jota käytetään nykypäivänä. SaaS koneistus allokoidaan tehokkaasti vastaamaan tarjottua työmäärää. Tämä edellyttää SaaS-palveluiden mallintamista ennustamaan tarvittavaa suorituskykyä ja järjestelmän kokonaiskustannuksia. Lisäksi käyttötarkoitukseen tarvittava koneistuksen määrä ja niiden vastaava kapasiteetti tulee arvioida ennen varsinaista käyttöönottoa. (El Kafhali & Salah 2018, 1.)

Tunnettuihin SaaS-sovelluksiin ja -palveluihin kuuluu muun muassa sähköposti, myynninhallinta, asiakassuhteiden hallinta, verkkopelaaminen, verkkoveropalvelut, taloushallinto, henkilöstöhallinto, laskutus ja yritysyhteistyö. Käyttäjäsovellukset toimitetaan palveluina asiakkaille netin välityksellä. Tämän tyyppisissä pilvipalveluissa kuluttajat ottavat sovelluksensa käyttöön pilviympäristössä, jonka avulla sovelluksen käyttäjät pääsevät käsiksi tarjottuun sovellukseen monista eri liittymistä, kuten selaimista. (El Kafhali & Salah 2018, 1.)

Mahdollisuus käyttää ohjelmistoa, kuten sähköpostia, ilman kalliiden laitteistojen ja monimutkaisten ohjelmistojen käyttöönottoa kotona tai yrityksen tiloissa, tarjoaa suurta joustavuutta ja kustannussäästöjä niin yksityishenkilölle kuin yritykselleki n. SaaS-palvelumalli antaa siis asiakkaalle mahdollisuuden käyttää toimittajien ohjelmistoja, kuten sähköpostia, ilman alustan käyttöön vaadittavia laitteistoja sekä niiden hallintaa ja ylläpitoa. (Williams 2012.)

Pilven asiakkaat eivät pääse käsiksi infrastruktuuriin, kun he käyttävät SaaS-palvelumallia. SaaS-infrastruktuuri on rakennettu tällöin valmiiksi ja usein hyödyntää usean vuokralaisen arkkitehtuurimallia, jossa useampi sovellus on organisoitu yhteen loogiseen ympäristöön. Tällä pyritään saavuttamaan sovelluksien tehokas skaalautuvuus ja taloudellinen optimointi, jolla mahdollistetaan nopeus, turvallisuus, saavutettavuus, katastrofipalautus ja ylläpito. (El Kafhali & Salah 2018, 1.)

5 TEKNOLOGIAT

Tiedonsiirtoon verkkoyhteyksien välityksellä liittyy monia teknologioita. Suosituimmat ja yleispätevimmät teknologiat käyttävät IP, TCP ja UDP protokollia. Vuosien varrella useita muita korkeamman tason protokollia on kehitetty erikoistumaan nimenomaan tiedonsiirtoon. (Karla & Tarnawski 2019. 1.) Näistä korkeamman tason tiedonsiirron menetelmistä ja käyttämistäni teknologioista tulen kertomaan tässä luvussa.

5.1 HTML

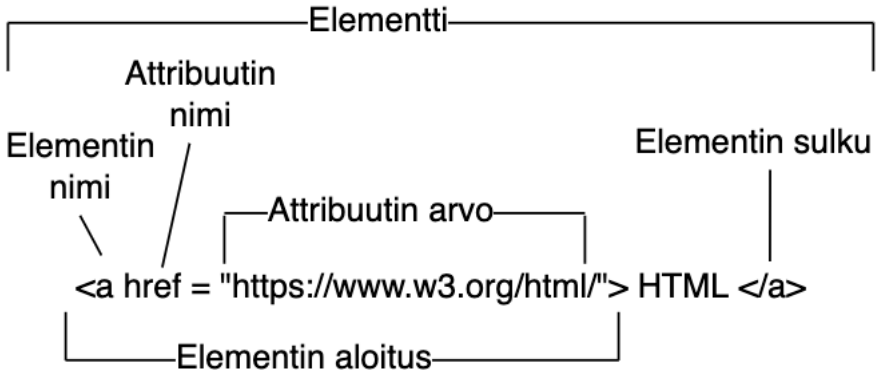
HTML tulee sanoista Hypertext Markup Language. Sen voisi siis toisin sanoen ilmaista yksityiskohtaisiksi tyyliohjeiksi, jotka on kirjoitettu tekstiasiakirjaan ja, joka julkaistaan Word Wide Webissä. (McFedries 2019.) HTMLää käytetään kuvaamaan verkkosivun rakennetta. Kun selain siirtyy verkkosivulle, se vastaanottaa ja jäsentää HTML-dokumentin, joka voi sisältää tekstiä, kuvia, linkkejä ja muuta mediaa, kuten ääntä ja videoita. (Coulson, Jephson & Larsen 2019.)

HTML-dokumentti sisältää tuen ja pääsyn erilaisiin asioihin, kuten vuorovaikutukseen käyttäjien kanssa. Vuorovaikutusta tuetaan esimerkiksi tapahtumankäsittelijöillä, tarkentamisella, kopioimisella ja liittämisellä (Elrom 2021.), joita nykypäivänä pidetään lähes, jos ei kokonaan, itsestään selvyyksinä.

Lyhenteessä lymyilevä Language-sana antaa ymmärtää, että HTML on ohjelmointikieli. HTMLllä ei ole kuitenkaan mitään tekemistä tietokoneohjelmoinnin kanssa. Se on pikemminkin kieli vain siinä mielessä, että siinä on pieni kokoelma sanoja (elementtejä), joilla määritellään, miten haluat tekstin näkyvän selaimessa. Esimerkiksi onko kyseessä otsikko vai numeroitu luettelo. (McFedries 2019.)

HTML sisältää paljon niin sanottuja tageja, toiselta nimeltään elementtejä, joiden avulla voidaan määritellä yksityiskohtia, kuten tekstin jakamista kappaleiksi, luetteloiden luomista, kuvien lisäämistä ja linkkien määrittelyä. Kun elementtejä on kirjoitettu sopivassa järjestyksessä tavalliseen tekstiasiakirjaan, selain hoitaa niin

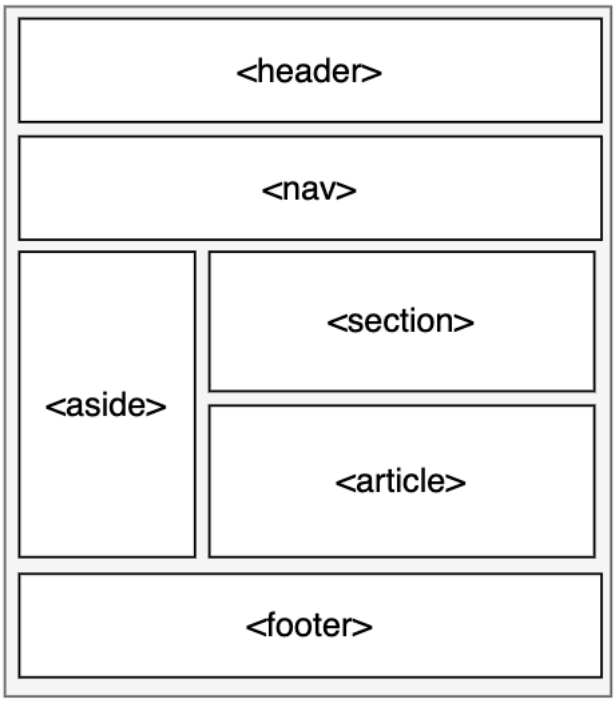
sanotun likaisen työn eli elementtien kääntämisen tai renderöinnin. (McFedries 2019.) Alla olevassa kuviossa 4 on esimerkki linkkielementin rakenteesta.



KUVIO 4. HTML elementin osat visualisoituna sisältäen attribuutin. Muokattu. (Coulson ym. 2019.)

HTML sisältää niin sanottuja semanttisia elementtejä. Toisin sanoen elementillä saattaa oli jokin tietty rooli tai tarkoitus, jolloin esimerkiksi useamman kuin yhden samanlaisen elementin käyttäminen ei ole järkevää. Esimerkiksi ”main” elementti kuvaa HTML dokumentin pääsisältöä. Verkkosivustolla ei tulisi näkyä koskaan enempää kuin yksi pääelementti kerrallaan. (Coulson ym. 2019.) HTML5 mukana tuli paljon juurikin semanttisia saavutettavuutta helpottavia elementtejä, esimerkiksi article, aside, footer, header ja section (Powell, 2010). Kyseiset elementit ovat nykypäivänä hyvin käytettyjä ja tuovat mielestäni myös paljon selkoa, mitä elementit pitävät sisällään.

Moderni ajattelu on, että koitetaan välttää ”div merta” käyttämällä semanttisia elementtejä sen sijaan. Koen osittain epäonnistuneeni semanttisten elementtien käytössä, sillä henkilökohtainen yksityiskohtien tietämys on vielä hyvin vähäistä lyhyen urani ja opintojen tiiviyden vuoksi.



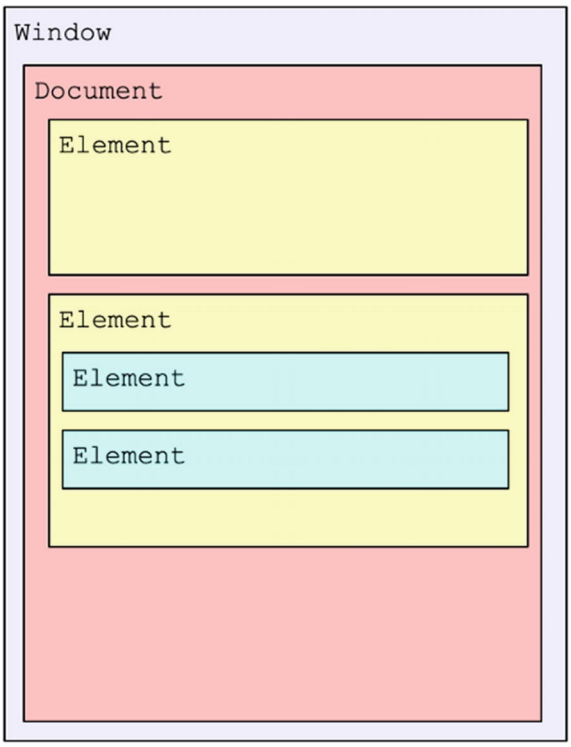
KUVIO 5. HTML5 tuomia semanttisia elementtejä esitettynä sivuston luurankona.

Kuvio 5 havainnollistaa hyvin eri elementtien käyttötarkoituksia suhteessa toisiinsa. Section kuvastaa geneeristä osiota, joka voi sisältää oman ylä- ja alatunnisteensa. Header kuvastaa kyseessä olevan osion tai sivun ylätunnistetta ja se sisältää sisältöä kuvailevaa tietoa. Footer puolestaan sisältää osion täydentävää tietoa toimien sisällön alatunnisteena. Nav kokoaa ryhmän linkkejä yhteen toimien dokumentin tai sivuston navigaationa. Article toimii itsenäisenä osiona, kuten blogijulkaisuna, artikkelina tai itseään jatkavana tietoyksikkönä. Aside sisältää tangentiaalisessa suhteessa olevan sisällön toiseen elementtiin. (Powell 2010.)

5.2 DOM

DOM tulee sanoista Document Object Model. DOMilla tarkoitetaan HTMLn yhteydessä puurakenteista muistissa esiintyvää tapaa esittää HTML-dokumentti. (Elrom 2021.) Puu esitetään joukkona solmuja ja niiden yhteyksiä muihin solmuihin. Solmu edustaa kyseisen elementin sijaintia dokumenttipuurakenteessa tallentamalla viitaukset sen pää- ja alisolmuihin. (Coulson ym. 2019.)

DOMia voi ajatella HTML-dokumenttien APIna, kuten esimerkiksi svg on kuville. Tällöin API kuvaa dokumenttia sisältäen rajapinnat, jotka määrittävät HTML-elementtien toimintaperiaatteita. Lisäksi se ottaa huomioon kaikki rajapinnat ja tyypit, joihin elementit perustuvat. (Elrom 2021.) DOM edustaa HTML-dokumentteja tavalla, joka mahdollistaa puurakenteen läpi kulkemisen ja tarvittavien muutosten tekemisen ohjelmallisesti. Ohjelmalliset muutokset tehdään elementteihin muuttamalla solmujen ominaisuuksia. (Coulson ym. 2019.)



KUVIO 6. DOM hierarkia rakenne. Selaimen ikkuna, jossa kuvataan HTML-dokumentin hierarkiaa DOMin tavoin.

DOM koostuu siis hierarkkisesta rakenteesta erilaisia solmuja ja solmujen rajapinta mahdollistaa itse dokumentin käsittelyn lisäksi jokaisen solmun käsittelyn. (Elrom 2021.) Kuviossa 6 nähdään yksinkertainen DOM-hierarkia puurakenne. Siitä voidaan huomata, kuinka elementit dokumentin sisällä rakentuvat vierekkäisissä ja sisäkkäisissä suhteissa toisiinsa. Ikkuna (window) on selaimen sen hetkinen näkymä.

DOM-manipulaatio on nykyaikaisen interaktiivisten verkkosivujen ydin. Manipulaatiolla mahdollistetaan verkkosivun sisällön muuttaminen dynaamisesti. DOM paljastaa tarvittavat APIt teknologioille, joilla pystytään muuttamaan koodia ajon

aikana. Tämän ansiosta nykyaikaiset verkkosivut voivat olla paljon enemmän kuin staattisia asiakirjoja. (Coulson ym. 2019.)

5.3 HTTP

HTTP on yksi yleisimmistä tiedonsiirron protokollista, joka on kehitetty verkkosivujen käsittelemiseen selaimen tasolla (Karla & Tarnawski 2019. 1). Lyhenne HTTP tulee sanoista Hypertext Transfer Protocol. Se on keskeinen tapa pyytää pääsyä verkkosovelluksiin ja -resursseihin. (Pollard 2019.) Perinteinen HTTP-protokolla toimii siis asiakassovelluksen ja palvelimen välillä, jossa palvelin vastaa ainoastaan asiakkaan pyyntöihin (Karla & Tarnawski 2019. 1). Se on yksi kolmesta tärkeimmistä tekniikoista, jotka Tim Berners-Lee määritteli verkkoa keksiessään. Kun tarkastellaan HTTPtä ollaan ensisijaisesti tekemisissä Word Wide Webin kanssa. (Pollard 2019.)

Kuitenkin nykypäivänä voidaan rakentaa sovelluksia HTTPn päälle ilman perinteistä käyttöliittymää, jolloin Word Wide Webin määrittelemisestä tulee yhä hankalampaa. Toisin sanoen HTTPtä hyödyntävät palvelut, kuten REST ja SOAP, mahdollistavat mobiilisovellusten ja IoT-laitteiden verkkoon yhdistämisen. Tämä mahdollistaa HTTPn avulla viestin lähettämisen esimerkiksi lampulle, sen sammuttamiseksi tai päälle panemiseksi puhelinsovelluksesta. (Pollard 2019.)

HTTP-protokolla on yleisesti helppo toteuttaa ja ymmärtää, sillä se ei sisällä monimutkaisia kerroksia verrattuna WSDL- ja SOAP-tekniikoihin. (Filipova & Vilão 2018.) HTTP on alun perin tarkoitettu siirtämään hypertekstiasiakirjoja. Hypertekstiasiakirjalla tarkoitetaan asiakirjoja, jotka sisältävät linkkejä muihin asiakirjoihin. Ensimmäinen versio HTTPstä ei tukenut muuta kuin näitä asiakirjoja. Nopeasti kuitenkin ymmärrettiin, että HTTP-protokollaa voidaan käyttää muiden tiedostotyyppien, kuten kuvien, siirtämiseen. Tämän vuoksi lyhenteen Hypertext ei ole nykypäivänä relevantti, mutta on liian myöhäistä nimetä se uudelleen, sillä HTTP on liian yleisesti käytetty. (Pollard 2019.)

HTTP on riippuvainen luotettavasta verkkoyhteydestä, joka on rakennettu jonkin tyyppiselle fyysiselle yhteydelle, kuten ethernet tai Wi-Fi. Koska

viestintäprotokollat ovat jaettu kerroksiin, jokainen kerros voi keskittyä siihen, mitä se tekee hyvin. HTTP ei koske alemman tason yksityiskohtia verkkoyhteyden muodostamisesta. Vaikka HTTP-sovellusten tulisi olla tietoisia verkkovikojen tai yhteyden katkeamisen käsittelystä, protokolla ei itse ota huomioon näitä tehtäviä. (Pollard 2019.)

5.4 REST

Nykypäivän frontend-teknologiat kuten React, Angular ja Vue, mahdollistavat sivuston renderöimisen frontendin puolella JavaScriptin avulla. Tällöin asiakassovellus lähettää palvelimelle pyyntöjä hakien vain informaatiota tai lähettääkseen lomakkeelle annettuja tietoja. Käyttäjä pystyy olemaan vuorovaikutuksessa sivuston kanssa saman aikaisesti, kun pyyntöjä käsitellään. JavaScriptin HTTP asiakkaiden asynkronisen luonteen ansiosta vuorovaikutus on mahdollista. Jos puolestaan renderöinti tapahtuu palvelimen puolella, sivustosta tulee hidas, sillä navigoinnin tapahtuessa selaimen tulee hakea kokonaiset sivut palvelimelta pelkän informaation sijaan. Siksi hyödynnetäänkin enenemissä määrin frontend puolella tapahtuvaa renderöintiä ja palvelimille rakennetaan HTTP RESTful API. (Lyu 2021.)

RESTful-verkkopalvelut ovat yksi suosituimmista palveluista, sillä se perustuu tyypillisesti HTTP-protokollaan. (Filipova & Vilão 2018.) REST nousi SOAPin muutaman vuoden dominoinnin jälkeen ja esitteli yksinkertaisemman menetelmän tietojen jamiseen. REST-sovellusliittymiä ei ole sidottu tiettyyn muotoon, ne sijoittuvat tyypillisesti välimuistissa sekä ne toimitetaan HTTP- ja HTTPS-protokollan avulla. Suurin tuulahdus on HTTP-verbien noudattaminen samalla kunnioittamalla verbien alkuperäistä tarkoitusta. Esimerkiksi verbit, kuten DELETE ja PATCH, jäivät käyttämättä, vaikka niiden tarkoitus oli hyvin selkeä. REST on ollut ensisijainen sysäys oikean menetelmän ja käyttötarkoituksen yhdistämiseen. (Kozyra 2016.)

REST noudattaa CRUD (Create-Read-Update-Delete) -tyyppistä lähestymistapaa tietojen hakemiseen ja muokkaamiseen. Verbeistä POSTia käytetään pääasiassa luomiseen, PUT ja PATCH ovat päivittämiseen, GETtiä lukemiseen ja DELETEä poistamiseen. (Kozyra 2016.)

Kozyra myös huomauttaa, kuinka tärkeää on, että RESTful API:n tulisi olla tilaton. Jokaisen pyynnön tulee siis pystyä olla olemassa omanaan, jolloin palvelimella ei ole mitään tietoa aikaisemmista tai tulevista pyynnöistä. (2016.)

RESTfull APIn luoma sovellusarkkitehtuuri antaa myös vapauksia kehitykseen. Nyt backend- ja frontend-tiimi pystyy työskentelemään paremmin itsenäisesti toisistaan riippumatta. Frontend ei myöskään tarvitse enää itse palvelimen sovellusta. Sen sijaan palvelinsovellus voidaan ottaa käyttöön täysin erillisellä palvelimella tai palvelussa maksimoiden suorituskykyä. Tiimien tulee kuitenkin sopia niin sanottu API-sopimus. (Lyu 2021.) API-sopimuksella tarkoitetaan pääasiassa APIn dokumentaatiota. Dokumentaatiossa määritellään, kuinka API tulee käyttäytymään. Se sisältää muun muassa pääte urlit, päätteiden toiminnat, argumentit ja esimerkki vastaukset. (Ibanez 2020.)

5.5 JSON

JavaScript Object Notation eli JSON on nykyään suosituin tapa koodata dataa eri järjestelmien välillä liikkumista varten. JSON koodatulla tiedostolla on useita etuja aikaisempiin ponnisteluihin verrattuna. JSONin ehdottomat etuudet ovat, että se on erittäin helposti ihmisen luettavissa, sitä on helppo jäsentää ohjelmistojen kanssa sekä se ei ole liian monimutkaista. Koska jaetun tiedon määrä kasvaa joka vuosi, on välttämätöntä, että tiedot jaetaan helpossa muodossa. Tiedon muodon helppous mahdollistaa ohjelmoinnin helppouden, oikolukemisen ja virheidenkorjauksen sekä alhaiset kustannukset. (Stokes 2018.)

JSON on tekstipohjainen, kielestä riippumaton tiedonvaihtomuoto tietojen sarjoittamista varten. Se on johdettu JavaScriptin objektiliteraaleista kuten ne on määritelty ECMAScript Language Specificationin kolmannessa painoksessa. JSONille on kuitenkin kaksi standardia: Internet Engineering Task Force- (IETF) ja European Computer Manufacturers Association (ESMA) standardi. IETFn dokumentaatio on noin kuusitoista sivua pitkä ja ECMAn on viisi. Nämä kaksi standardia ovat kuitenkin kohtuullisen yksiselitteisiä. (Stokes 2018.)

JSONin rakenteen voisi helposti ajatella, että JSON-tiedot on strukturoitu objekteiksi, joissa on nimi/arvo-pareja, tai järjestetyiksi arvoluetteloiksi, joita kutsutaan myös taulukoiksi. Useimmat ohjelmointikielet ja niiden ohjelmoijat käyttävät objekteja ja/tai taulukoita säännöllisesti. JSONin yksinkertainen rakenne mahdollistaa sen, että se on riippumaton tietojen luomiseen tai lukemiseen käytetystä tietokonekielestä. (Stokes 2018.)

5.6 Node.js

Node.js on jännittävä uusi alusta verkkosovellusten, sovelluspalvelimien, verkkopalvelimien sekä yleiskäyttöisen ohjelmoinnin kehittämiseen. Se on suunniteltu äärimmäiseen skaalautumiseen verkkosovelluksissa, joka on mahdollistettu yhdistämällä nerokkaan palvelinpuolen JavaScriptin ja asynkronisen ohjelmoinnin keskenään. (Herron 2020.)

Viimeisen vuosikymmenen aikana Node.js on muuttunut uudesta innovaatiosta uusien sovellusten todelliseksi alustaksi (Hunter, 2021), ja näin muovannut itselleen merkittävän roolin nykypäivän ohjelmistokehityksessä. Alustaa käytetään laajasti sekä suurissa, että pienissä projekteissa. Esimerkiksi PayPal on muuntanut monia palveluitaan Javasta Node.js:ksi. (Herron 2020.)

JavaScript kielenä antaa kehittäjille mahdollisuuden luoda melkein ajatuksen nopeudella. Koodi, jota kirjoitetaan, on useimmiten riittävän yksinkertaista, joten sen kirjoittaminen käsin on tehokkaampaa kuin sen generoiminen. Tämä JavaScriptin kaunis yksinkertaisuus sopii täydellisesti Node.js:n kanssa. Node.js:n kehittäjä, Ryan Dahl loi alustan rakentamaan sovelluspalvelimen, joka oli suuruusluokkaansa nähden huomattavasti helpompaa ja nopeampaa kuin mihin kukaan oli tottunut. Tulokset ovat ylittäneet villeimmätkin unelmat. Node.js:n helppouden ja yksinkertaisuuden ansiosta voidaan luoda, validoida ja innovoida tavoilla. Jotka eivät yksinkertaisesti olleet mahdollisia kymmenen vuotta sitten. (Hunter 2021.)

Node.js-arkkitehtuuri poikkeaa muiden sovellusalustojen tyypillisestä valinnasta. Normaalisti säikeitä käytetään sovelluksen skaalautumiseen, kuitenkin Node.js välttelee säikeisyyttä niiden luontaisen monimutkaisuuden vuoksi. Väitetään, että yksisäikeisissä tapahtumakeskeisissä arkkitehtuureissa muistin jalanjälki on

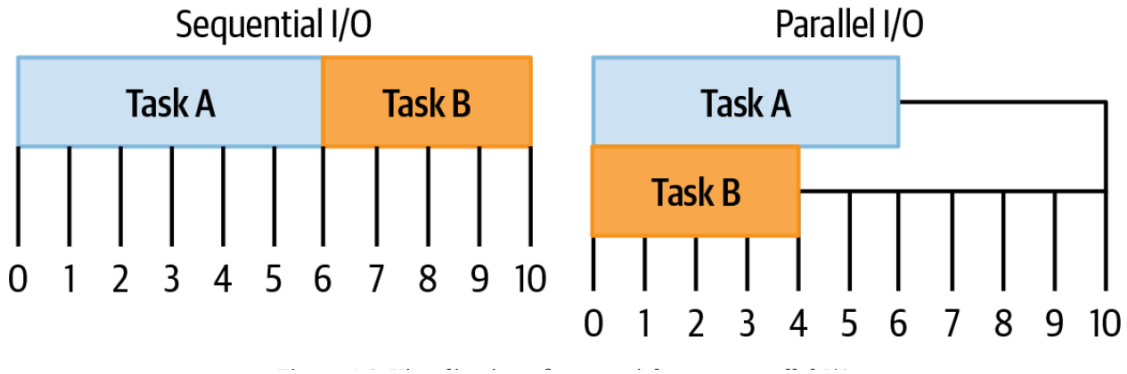
pieni, suorituskyky suuri, viive kuormituksessa on lyhyempi ja ohjelmointi itsessään yksinkertaisempaa. (Herron 2020.)

Pohjimmiltaan Node.js on JavaScriptin erillinen laajennettu moottori. Se soveltuu yleiskäyttöiseen ohjelmointiin ja keskittyy selkeästi palvelinpuolen kehitykseen. (Herron 2020.)

Tärkein arkkitehtuurinen valinta on Node.js:n tapahtumalähtöisyys monisäikeisyyden sijaan. Arkkitehtuuri siis perustuu takaisinkutsufunktioiden lähettämiseen yksisäikeiseen tapahtumasilmukkaan, jolloin tulokset saapuvat kutsujalle tapahtumana, joka laukaisee tapahtumankäsittelijän takaisinkutsufunktion. (Herron 2020.)

Node.js kattaa läpikotaisin Continuation-Passing Style (CPS) -mallin. Se näkyy moduuleiden takaisinkutsuissa. Takaisinkutsu on funktio, jotka välitetään eteenpäin. Tapahtumasilmukka kutsuu välitettyä takaisinkutsua, kun alkuperäinen tehtävä on suoritettu. Näiden takaisinkutsufunktioiden nähdään toimivan asynkronisesti. Kuitenkin, jos ajatellaan tarkemmin, asia on päinvastoin. Kun yksi funktio kutsuu toista funktiota samassa pinossa, koodin sanotaan toimivan synkronisesti. (Hunter, 2021.) Monissa tapauksissa tapahtuma kuitenkin halutaan tehdä asynkronisesti (Herron 2020).

Pitkäkestoiset tehtävät ovat tyypillisesti sisään tai ulos kulkevia tehtäviä, niin sanottuja I/O tehtäviä (input/output). Esimerkiksi, jos sovelluksesi tarkoitus on suorittaa kaksi tehtävää. Tehtävä A on lukea tiedosta ja tehtävä B on lähettää http pyyntö kolmannen osapuolen palveluun. Jos seuraava prosessi on riippuvainen molemmista tehtävistä, tehtävät tulisi kyetä suorittamaan rinnakkaisesti kuvion 7 mukaisesti. Jos suorittaminen tapahtuisi peräkkäisinä toimintoina http-pyyntöön vastaamiseen kuluva kokonaisaika olisi pidempi, jolloin seuraavan prosessin aloittaminen myöhästyy. (Hunter 2021.)



KUVIO 7. Peräkkäisten ja rinnakkaisten tehtävien visualisointi. Havainnollistaa kahden tehtävän suorittamisen ajankäyttöä, joko peräkkäin tai rinnakkain. (Hunter 2021.)

Rinnakkaistoiminta näyttää nopeasti rikkovan JavaScriptin yksisäikeistä luonnetta. Tästä kuitenkin päästään mielenkiintoiseen osioon. Node.js itsessään on monisäikeinen. Node.js:n alemmat tasot ovat kirjoitettu C++:lla, kuten alla olevasta taulukosta voidaan huomata (taulukko 2). Se siis sisältää kolmannen osapuolen työkalut, kuten libuv, joka käsittelee käyttöjärjestelmän abstraktioita ja I/O:ta kuin myös JavaScriptin moottoria, V8:aa. Vain Node.js:n ylemmät kerrokset on kirjoitettu JavaScriptillä, jotka käsittelevät suoraan kehittäjän tekemiä operaatioita. (Hunter 2021.)

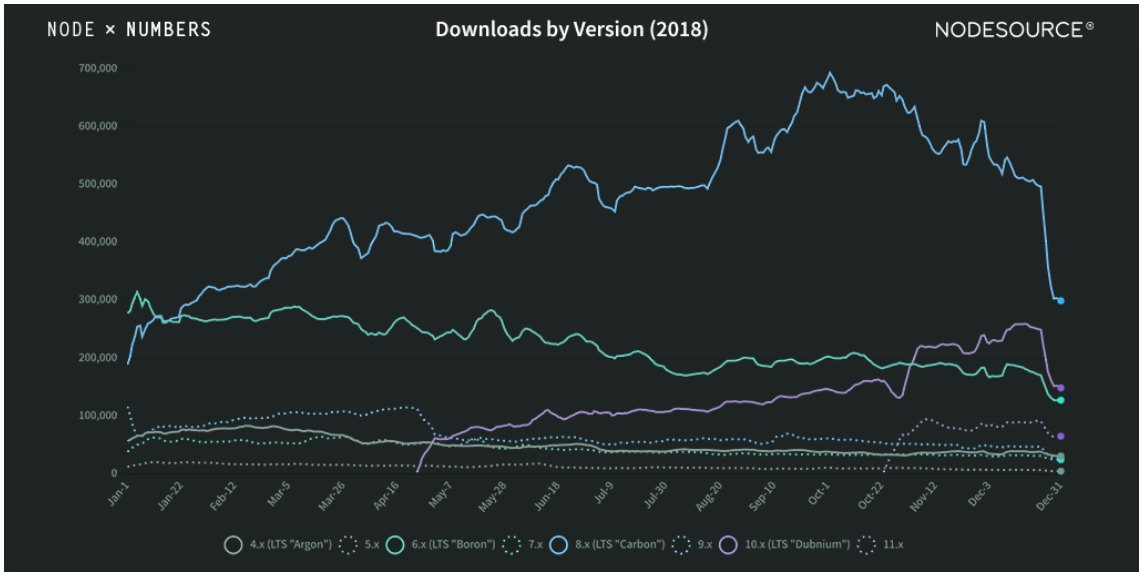
TAULUKKO 2. Node.js:n rakennetasojen väliset suhteet. Havainnollistaa Node.js:n tasojen suhteita keskenään. Kuinka C++:aa ja JavaSciptia on käytetty eri osaan alustan kommunikoinnissa. Muokattu. (Hunter 2021.)

<table>
<tr><td rowspan="2">JavaScript</td><td colspan="4">Kehittäjä: sovelluksen lähdekoodi ja npm moduulit</td></tr>
<tr><td colspan="4">Core Node.js APIt</td></tr>
<tr><td rowspan="2">C++</td><td colspan="4">Node.js sidokset</td></tr>
<tr><td>OpenSSL</td><td>V8</td><td>libuv</td><td>jne.</td></tr>
<tr><td></td><td colspan="4">Käyttöjärjestelmä</td></tr>
</table>

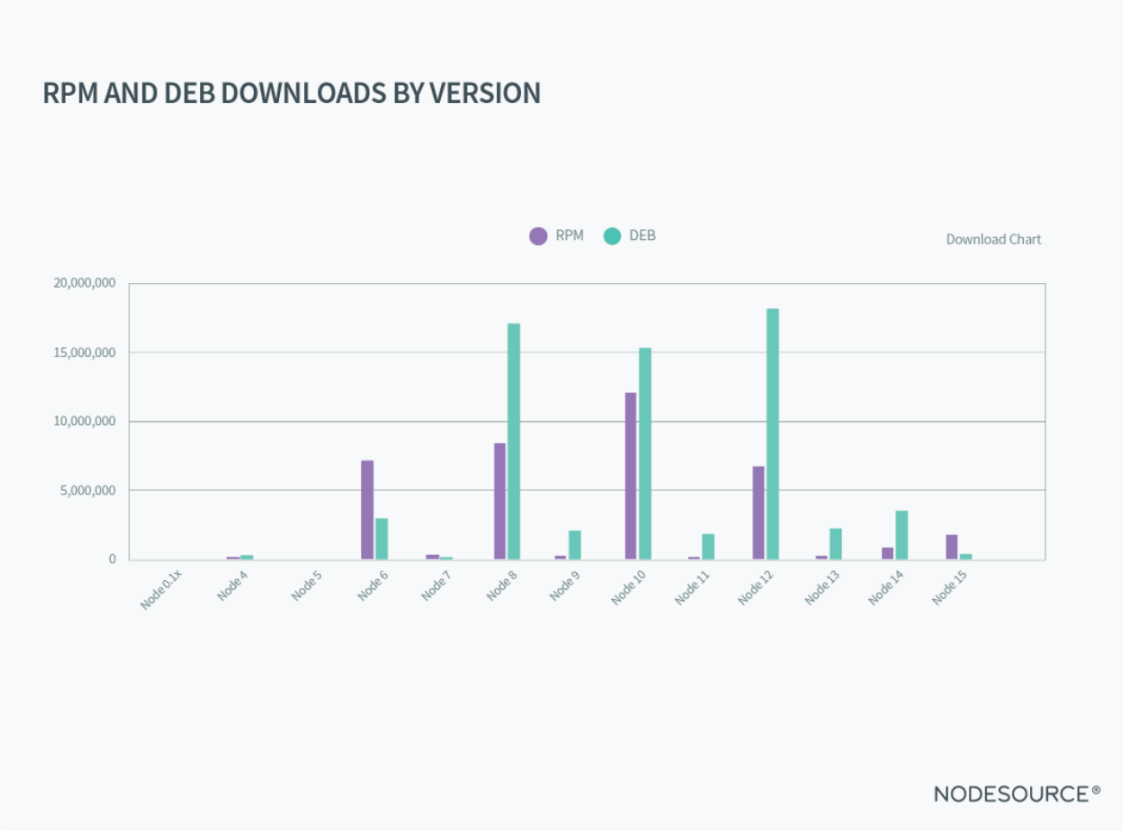
5.7 Miksi valita Node.js?

Yksi suurista syöstä valita Node.js on sen suosio. Node.js:stä on nopeasti kasvanut suosittu kehitysalusta organisaation tai projektin koosta riippumatta. Suuria Node.js:n käyttäjiä ovat esimerkiksi PayPal, LinkedIn ja eBay. (Herron 2020.)

NodeSourcen mukaan Node.js:n kasvu on ollut hyvin nopeaa. Jos verrataan vuoden 2018 ja 2020 taulukoita alustan latausmääristä (kuvio 8 ja 9 x) (NodeSource n.d.). Niistä huomataan, että kahden vuoden aikana vuoden 2018 suosituin version 8.x latauskertojen määrä on kasvanut melkein 700 tuhannesta latauksesta noin 17 miljoonaan latauskertaan puhumattakaan versiosta 12.x, jota ei ollut julkaistu vielä 2018. Yhteensä latauksia vuonna 2020 oli 98.9 miljoonaa. (Parody & Villa 2021.)

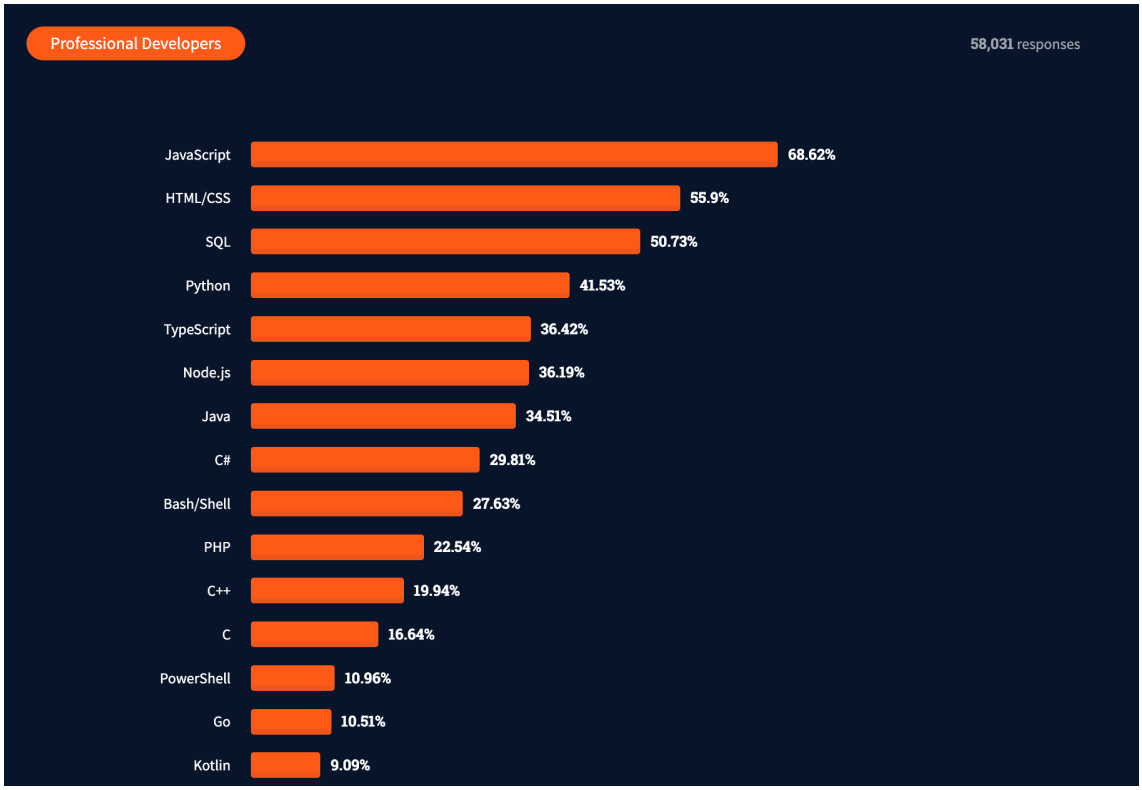


KUVIO 8. Node.js lataukset versionnumeroittain 2018. Havainnollistaa Node.js latauksien määrää vuonna 2018 eritellen Node.js versionumerot (NodeSource n.d.).



KUVIO 9. Node.js lataukset versionnumeroittain 2020. Havainnollistaa Node.js latauksien määrää vuonna 2020 eritellen Node.js versionumerot (Parody & Villa 2021).

Suosiosta myös kertoo StackOverFlow vuotuinen kyselytutkimus, jonka mukaan Node.js nousi kuudenneksi suosituimmaksi kieleksi vuonna 2021 (kuvio 10). Kyselyyn viittaava kuvio 10 ottaa huomioon vain ammatikseen tekevät ohjelmoijat. Suosio ei kuitenkaan laske, kun otetaan mukaan myös harrastelijat. (StackOverFlow 2021.)



KUVIO 10. Suosituimmat ohjelmointikielet 2021 StackOverFlow kyselytutkimuksessa. Havainnollistaa Node.js suosiota verrattuna muihin ohjelmointikieliin. Node.js on ammattikehittäjien keskuudessa kuudenneksi suosituin kieli (Stackoverflow 2021).

Suosio ei kuitenkaan kerro kaikkea, sillä jokainen väittää, että heidän ohjelmistonsa tekee hienoja asioita. On tärkeämpää ottaa huomioon ohjelmiston tekninen ansio. (Herron 2020.)

Toinen merkittävä etuus Node.js:n käytössä on se, että näin mahdollistetaan JavaScriptin käytön sekä asiakas- että palvelinsovelluksessa. Unelma mahdollisuudesta koodata vain yhdellä kielellä juontaa juurensa Javan alkuaikoihin, jolloin selaimessa olevat Java-sovelmat tuli olla käyttöliittymä palvelinsovelluksille, jotka oli kirjoitettu Javalla. Tällöin JavaScript kehitettiin juuri näiden sovelmien kevyeksi ohjelmointikieleksi. Java ei koskaan saavuttanut suurta suosiota frontend-puolen ohjelmointikielenä ja Java-sovelma ilmauksena on haihtunut hämärään muistiin frontendin sovellusmallista. (Herron 2020.) Se on hyvin totta, sillä en itsekkään ollut kuullut Java-sovelmasta, ennen tätä hetkeä.

JavaScript on siis syrjäyttänyt Javan ensisijaisena selaimen sisäisenä frontend-kielinä. Yleensä frontend-kehittäjät siis elivät omassa JavaScript-

kielimaailmassaan, kun taas backend-kehittäjät koodasivat tunnetuimmin joko PHP:tä, Javaa, Rybya tai Pythonia.

Ajan myötä selainten sisäisistä JavaScript-alustoista tuli uskomattoman tehokkaita, mikä antoi mahdollisuuden kirjoittaa yhä monimutkaisempia selainpuolen sovelluksia. Node.js:n avulla voidaan vihdoin toteuttaa sovelluksia samalla ohjelmointikielellä sekä frontend- että backend-puolella. (Herron 2020.)

Yhteinen kieli mahdollistaa esimerkiksi sen, että sama ohjelmistohenkilöstö voi työskennellä molemmissa päissä ilman toisen kielen opettelua. Koodi pystytään siirtämään palvelimen ja asiakkaan välillä myös helpommin sekä tiedon välityksessä on käytössä yhdenmukaiset tietotyypit. Lisäksi yhteinen kieli tarjoaa mahdollisuuden käyttää yhteisiä ohjelmisto-, testaus- ja laadunraportointityökaluja ongelmitta. (Herron 2020.)

Node.js:n arkkitehtuurinen valinta suosia säikeettömyyttä ja rakentaa sovellus tapahtumasuuntautuneesti asynkronisella ohjelmointimallilla tuo ehdottomasti hyviä etuuksia. Tämä arkkitehtuuri yhdistettynä nopeaan JavaScript-moottoriin, väitetään tuottavan vähemmän yleiskustannuksia kuin säiepohjaisella arkkitehtuurilla. Muissa järjestelmissä, jotka käyttävät säikeiden mahdollistamaa samanaikaista työskentelyä, havaitaan monesti muistivuotoja ja turhaa monimutkaisuutta. (Herron 2020.)

Nykypäivänä on noussut idea mikropalveluista. Sen sijaan, että halutaan suuria kokonaisuuksia, verkkosovellukset pyritään jakamaan pieniksi ja hyvin kohdistetuiksi palveluiksi, joita pienet tiimit voivat helposti kehittää. Vaikka mikropalvelut eivät ole täysin uusi idea itsessään, mikropalvelumalli sopii paremmin moderneiden ketterien projektihallintatekniikoiden kanssa. Lisäksi se mahdollistaa yksityiskohtaisempien sovelluksien käyttöönotan. Node.js on erinomainen alusta mikropalveluiden toteutukseen yksisäikeisyyden ja tapahtumasuuntautuneisuuden ansiosta. (Herron 2020.)

Vuosina 2014 ja 2015 Node.js yhteisö kohtasi suuren erimielisyyden käytännön, suuntautumisen ja hallinnan suhteen. Projekti nimeltä io.js oli vihamielinen haara, jota ohjasi ryhmä, jonka tavoitteena oli sulauttaa useita ominaisuuksia ja vaihtaa

henkilöitä, jotka olivat mukana päätöksentekoprosessissa. (Herron 2020.) Lopputuloksena Node.js:n ja io.js:n arkistot yhdistettiin ja syntyi itsenäinen Node.js säätiö, jolla on vastuu yhteisöstä. Yhteisö työskentelee yhdessä edetäkseen yhteiseen suuntaan. (Williams 2015.)

Konkreettinen tulos projektien erimielisyydestä on uusien ECMAScript kielen ominaisuuksien nopea käyttöönotto. Tämä taas käytännössä tarkoittaa sitä, että uusimmat päivitetyt ominaisuudet, kuten promise ja asynkroniset toiminnot, ovat nopeasti käytössä Node.js ohjelmoijilla. Node.js yhteisö ei siis pelkästään selviytynyt riitatilanteesta, vaan yhteisö ja sen tukema alusta vahvistuivat sen seurauksena. (Herron 2020.)

Lopulta voidaan päätellä, että Node.js ei ole pelkästään suosittu alusta, vaan sillä on hyvin vahva yhteisö sen takana. Lisäksi alustan valintaa tukee useat ja vakavat tekniset syyt, joista Node.js:n arkkitehtuuriset valinnat ovat yksi suurimmista. Itse koen sen myös selkenä ja helppolukuisena kielenä, jota on myös helppo pilkkoa pienemmiksi palasiksi, mikropalveluiksi.

5.8 React

JavaScript on jokaisen frontend-kehittäjän sormien ulottuvilla, mikä tekee siitä erittäin suositun ohjelmointikielen. Sen jopa ajatellaan olevan web-sivujen asiakaspuolen koodi. (Herron 2020.)

React, joka tunnetaan myös nimellä ReactJS, on Facebookin, nykyisen Metan, kehittämä JavaScript-kirjasto. Se on kehitetty nimenomaan käyttöliittymäkehitykseen verkkoympäristössä. Reactin kehittäjänä on Jordan Walke, joka työskenteli tuolloin Facebookilla. Muita vastaavia kirjastoja ovat esimerkiksi jQuery, Angular ja Vue.js. (Elrom 2021.)

React käsittelee koodin taustalla ennen kuin päivittää muutokset asiakkaan selaimeen. Tämä nopeuttaa käyttäjän valitseman verkkosivun latautumista. Reactilla pystytään siis määrittämään, mitä muutoksia DOMiin halutaan tehdä. Muutokset määräytyvät React-elementtien perusteella ja ne toteutetaan DOMin

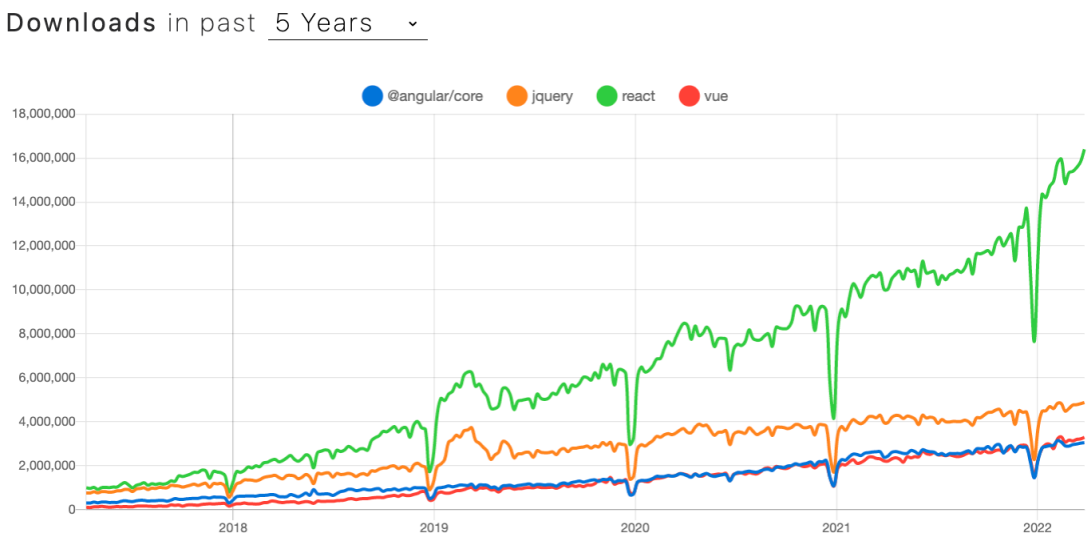
sisällä ennen varsinaista esittämistä käyttäjälle. Reactia voisikin kuvailla virtuaalisen DOMin simulaationa. Kun taustaprosessit ovat valmiita, muutokset toteutetaan DOMiin ja tulokset näkyvät selaimessa. (Elrom 2021.)

Virtuaalinen DOM (VDOM) on ohjelmointikonsepti, jossa käyttöliittymän ihanteellinen esitys säilytetään muistissa, josta se synkronoidaan varsinaiseen DOMiin manipulointia käsittelevän kirjaston avulla, kuten ReactDOM. Reactin päätavoite on nopeuttaa kyseistä prosessia. Muutosten tekeminen virtuaalisessa DOMissa auttaa välttämään koko HTML DOMin päivittämistä, jolloin prosessi nopeutuu. (Elrom 2021.)

Koko virtuaalisen DOMin päivittäminen on nopeampaa kuin varsinainen DOMin. Kun React-komponentti renderöidään, jokainen virtuaalisen DOMin elementti päivitetään. Virtuaalisen DOMin päivittämisen jälkeen alkaa täsmäytysprosessi, jolloin React selvittää, mitkä elementit muuttuivat. Täsmäytysprosessin lopussa vain muuttuneet elementit päivitetään varsinaiseen DOMiin koko DOM-puun sijaan. (Elrom 2021.)

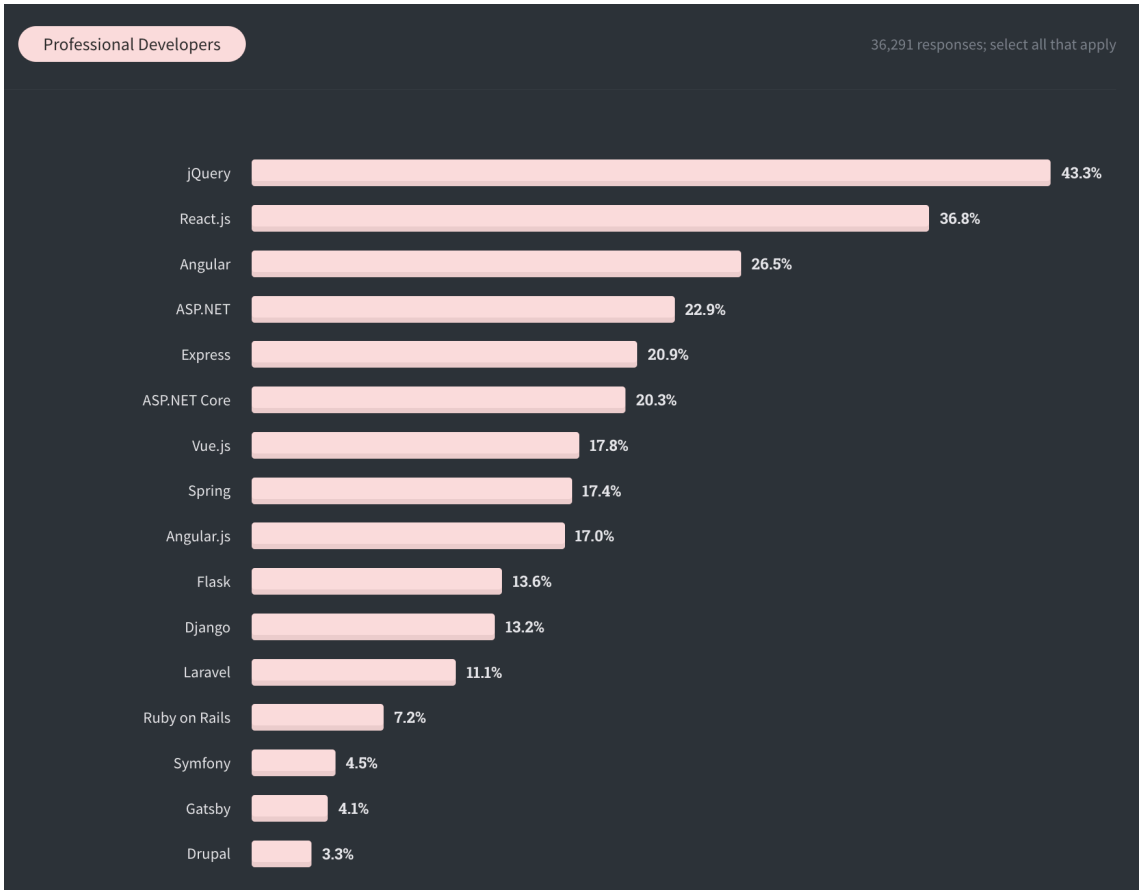
5.9 Miksi valita React?

Kuten Node.js myös React on suosittu ohjelmoijien keskuudessa.

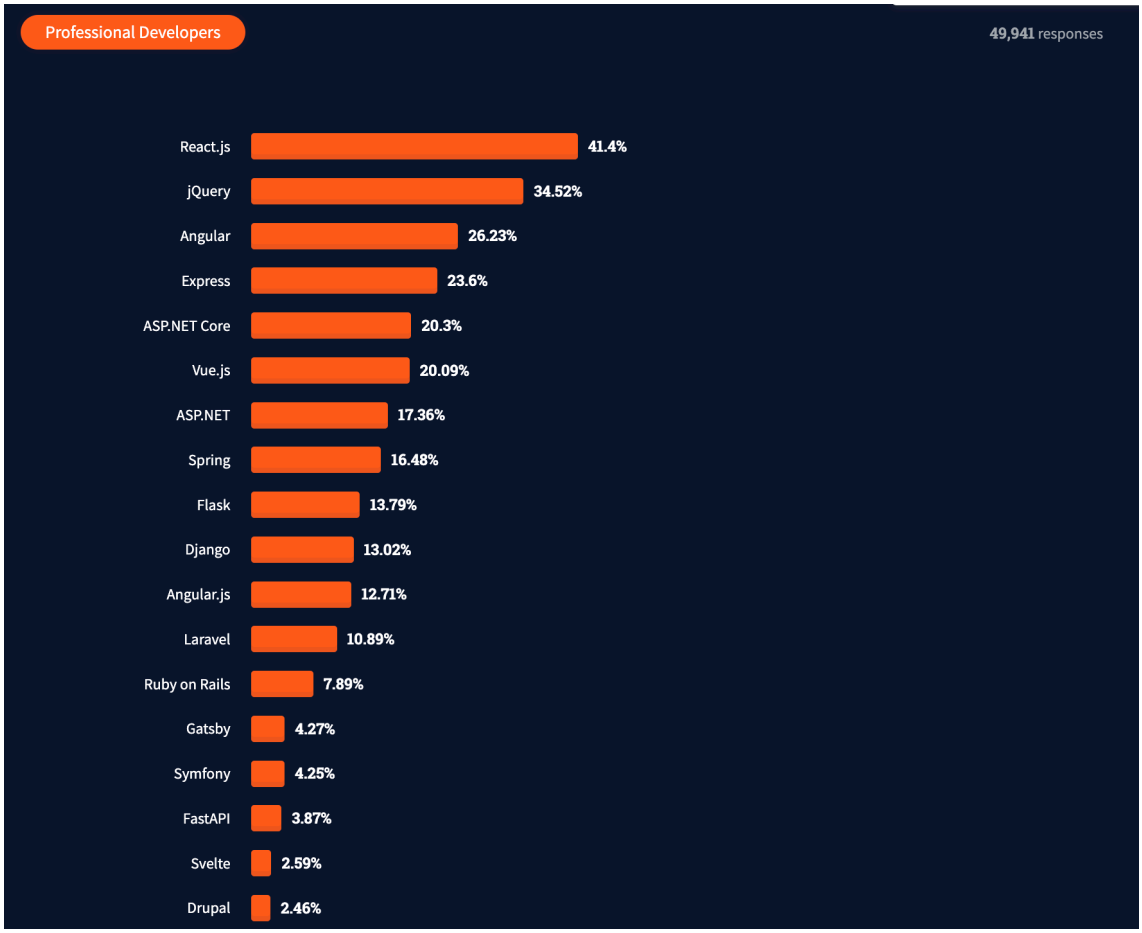


KUVIO 11. Käyttöliittymäkirjastojen latausmäärä 2018-2022. Havainnollistaa käyttöliittymäkirjastojen npm latauksien määrää suhteessa toisiinsa vuosilta 2018-2022 (npm trends 2022).

Vaikka kuviosta 11 voisi päätellä, että React on selkeästi suosituin kirjasto latausmäärien perusteella, kuvioista 12 ja 13 tarkastellessa huomataan, kuinka React syrjäytti vasta viime vuonna suurimman kilpailijansa jQueryn. Latausmäärän suuruutta selittänee se, että Reactilla kehittäminen on tehty hyvin helpoksi päästä alkuun (Elrom 2021), eikä kuviossa 11 huomioida latausten koulutuksellista, harrastelija tai ammatillista tarkoitusperää. Hauska sivu huomio on myös, kuinka kuviosta 11 voidaan huomata, miten paljon joulun ja vuoden vaihteen aikana latausmäärät laskevat.



KUVIO 12. Suosituimmat verkkoviitekehykset 2020 StackOverFlow kyselytutkimuksessa. Havainnollistaa React.js suosiota verrattuna muihin käyttöliittymän toteutuksiin käytettyihin viitekehyksiin vuonna 2020. React.js on ammattikehittäjien keskuudessa toiseksi suosituin viitekehys (Stackoverflow 2020).



KUVIO 13. Suosituimmat verkkoviitekehykset 2021 StackOverFlow kyselytutkimuksessa. Havainnollistaa React.js suosiota verrattuna muihin käyttöliittymän toteutuksiin käytettyihin viitekehyksiin vuonna 2021. React.js on ammattikehittäjien keskuudessa suosituin viitekehys (Stackoverflow 2020.)

React on kaiken lisäksi kevyt verrattuna yhteen suurimpaan kilpailijaansa, Angulariin (Restuta, Github 2019). Taulukkoa 3 tarkastellessa huomataan, kuinka React, React DOM ja redux kirjastot vievät noin 100 kilobittiä vähemmän muistia, kuin Angular ja Rx kirjasto.

TAULUKKO 3. Modernien JavaScript viitekehysten koot GZip:in jälkeen. Havainnollistaa React viitekehyksen keveyttä suhteessa muihin viitekehyksiin (Restuta, Github 2019).

Approximations for GZipped versions

Name	Size
Ember 2.2.0	111K
Ember 1.13.8	123K
Angular 2	111K
Angular 2 + Rx	**143K**
Angular 1.4.5	51K

React 0.14.5 + React DOM	40K
React 0.14.5 + React DOM + Redux	42K
React 15.3.0 + React DOM	43K
React 16.2.0 + React DOM	31.8K
Vue 2.4.2	**20.9K**
Inferno 1.2.2	20K
Preact 7.2.0	**4kb**
Aurelia 1.0.2	63K

Nykypäivänä on hyvin ratkaisevaa, että DOMin puurakenteiden manipulointi on mahdollisimman nopeaa (Elrom, 2021). Otetaan esille esimerkki, jossa verkkokaupan käyttäjä katsoo ostokorinsa sisältöä ja haluaa muokata yhden tuotteen lukumäärää. Useammat JavaScript kehykset päivittäisivät koko ostoskorin luettelon vain yhden tuotteen päivittämiseksi. Tämä voi tulla raskaaksi prosessiksi, kun tuotteiden määrä kasvaa esimerkiksi satoihin tuotteisiin. Elrom toteaa, kuinka nykyaikaiset verkkosivut sisältävät helposti suuria DOM-rakenteita, ja esimerkin kaltainen käyttäytyminen rasittaa turhaan käyttäjää sillä HTML-sivu latautuu huomattavasti hitaammin. Reactin hyödyntämä virtuaalisen DOMin päivittäminen siis mahdollistaa sen, ettei koko verkkosivua tarvitse päivittää, kun sivun informaatio muuttuu. (Elrom 2021.)

React-sovellus on erittäin hyvä alusta luoda niin sanottu yhden sivun sovellus (single-page application, SPA). SPA mahdollistaa sen, ettei sivua päivitetä ja kokemus tuntuu kuin käyttäjä olisi mobiilisovelluksessa. Sivut on tarkoitus renderöidä asiakasselaimessa, joka toimii erityisesti pienille ja keskisuurille projekteille (Elrom 2021.), kuin minun äänestyssovellukselleni.

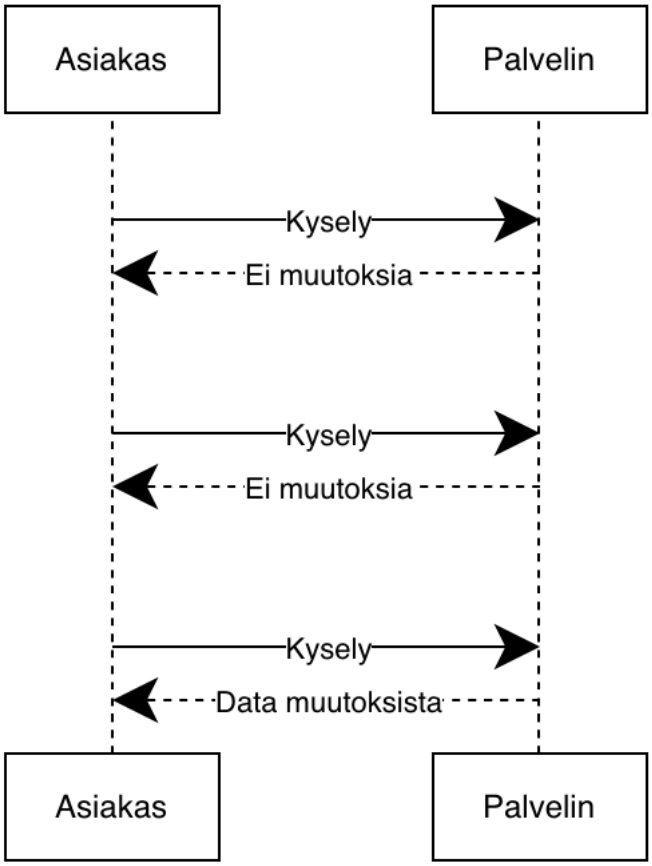
6 REAALIAIKAISUUS

Perinteisessä HTTP-mallissa ainoastaan asiakkaan tulee lähettää pyyntöjä ja palvelimen tulee prosessoida pyynnöt ja lähettää vastaus. Tämä sopii hyvin perinteisille verkkosivuille, jotka ovat käytännössä joukko dokumentteja. Kun kyse on kuitenkin verkkosovelluksesta, ne kehittyvät koko ajan yhä interaktiivisemmiksi, jolloin käyttäjälle tulee tarve lähettää tietoa palvelimelta asiakkaalle. (Lyu 2021.)

Kyky lähettää tietoa reaaliajassa verkon välityksellä käyttäen selainta avaa monia mahdollisuuksia liittyen simulaatioihin ja hallintaan. Näitä mahdollisuuksia voisivat hyödyntää niin teollisuus ja tutkimukset kuin opetukselliset yksiköt. Käyttämällä olemassa olevia laitteistorakenteita kuten verkkoselainta, on suhteellisen helppo laajentaa niiden toimintoja yhdistäen rakenteet simulaatioihin ja hallintasilmukoihin. (Karla & Tarnawski 2019. 1.)

6.1 Short Polling

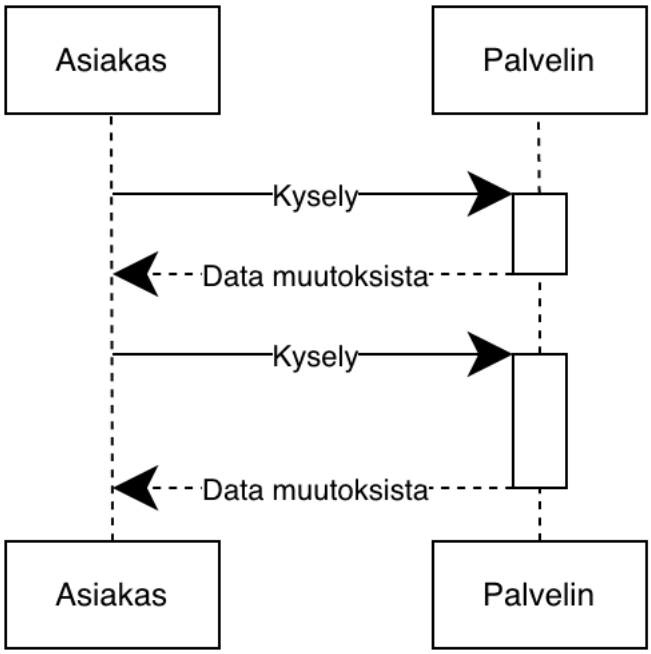
Short polling on yksi yksinkertaisimmista tavoista toteuttaa reaaliaikaisuutta verkkosovelluksissa. Tällöin asiakas lähettää aika ajoin kyselyn palvelimelle saadakseen tietää, onko palvelimella tapahtunut muutoksia. (Lyu 2021.)



KUVIO 14. Short polling. Havainnollistaa short polling protokollan viestintätapaa asiakkaan ja palvelimen välillä.

Huonoja puolia short pollingissa on se, että suurimmaksi osaksi palvelimella ei ole tapahtunut muutoksia. Tämä lähestymistapa siis lähinnä kuluttaa kaistaleveyttä lähettämällä monta kyselyä ja saamalla hyvin vähän käytettävää tietoa. Se voi myös aiheuttaa turhaa stressiä palvelimelle. (Lyu 2021.)

6.2 Long Polling



KUVIO 15. Long polling. Havainnollistaa long polling protokollan viestintätapaa asiakkaan ja palvelimen välillä.

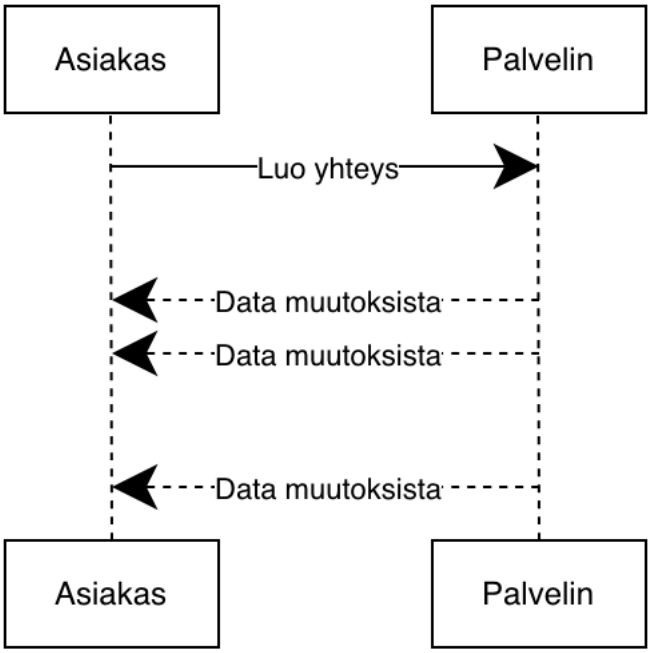
Long polling on prosessi, joka pitää yhteyden palvelimeen elossa ilman, että tiedot lähetetään välittömästi takaisin asiakkaalle (Lombardi 2015). Se siis rajoittaa palvelimelle lähetetyt ylimääräiset kyselyt. Kun asiakas lähettää HTTP-kyselyn palvelimelle, palvelin pidättäytyy vastaamasta siihen asti, kunnes sillä on dataa, jota lähettää takaisin. Tämä eroaa perinteisestä short polling -tekniikasta, missä vastaus lähetetään välittömästi takaisin, jos dataa ei ole saatavilla. (Lyu 2021.)

Kun asiakas saa vastauksen päivityksestä, asiakas lähettää välittömästi toisen pyynnön, jotta uuden päivityksen tullessa asiakas saa tiedon heti (Lyu 2021). Uuden kyselyn lähettäminen heti perään mahdollistaa jatkuvan yhteyden luomisen palvelimeen, jotta tietoja pystytään lähettämään edestakaisin (Lombardi 2015).

Long polling vähentää paljon kyselyssä tuotettua resurssien käyttöä, mutta tuottaa palvelimelle lisävastuuta useiden avoimien kyselyiden seuraamisesta. Menetelmä ei siis toimi hyvin sivuston ruuhkautuessa. Jotta long polling toimisi, ruuhkan tasaamiseksi on yleensä käytettävä ”sticky session”- strategiaa, jota on vaikeampi hallita kuin toisia ruuhkantasaus strategioita. (Lyu 2021.)

Lombardi toteaa, että long polling on yleisin tapa toteuttaa reaaliaikaisia sovelluksia verkossa (2015). Kuitenkin seitsemän vuoden aikana on tapahtunut paljon ja syntynyt uusia tapoja ja protokollia.

6.3 Server-Sent Events – SSE



KUVIO 16. Server-Sent Events. Havainnollistaa Server-Sent Events protokollan viestintätapaa asiakkaan ja palvelimen välillä.

Polling prosessien lisäksi HTML5 spesifikaatiossa on myös tekniikka, jota kutsutaan Server-Sent Events (SSE). Tällöin asiakas muodostaa yhteyden palvelimeen EventSource Web APIn avulla. Kun yhteys on muodostettu, palvelin voi lähettää tapahtumia asiakkaalle, joita asiakas voi käsitellä DOM tapahtumien tavoin. Koska tälle tekniikalle on hyvin määritelty standardi ja sille on monia kirjastoja, koodi on yleensä paljon yksinkertaisempi ja luettavampi kuin long polling -menetelmä. Tietoja voidaan kuitenkin lähettää vain yksisuuntaisesti palvelimelta asiakkaalle, mikä rajoittaa paljon käyttötapauksia. (Lyu 2021.)

Kehittäjät ovat luottaneet long polling-, short polling- ja SSE-tekniikoihin palvelimissa ja asiakkaissa, jotta yhteydet pysyvät auki pidempään ja luodaan pitkäkestoisen yhteyden illuusio. Vaikka nämä tavat toimivat teknisesti, ne aiheuttavat

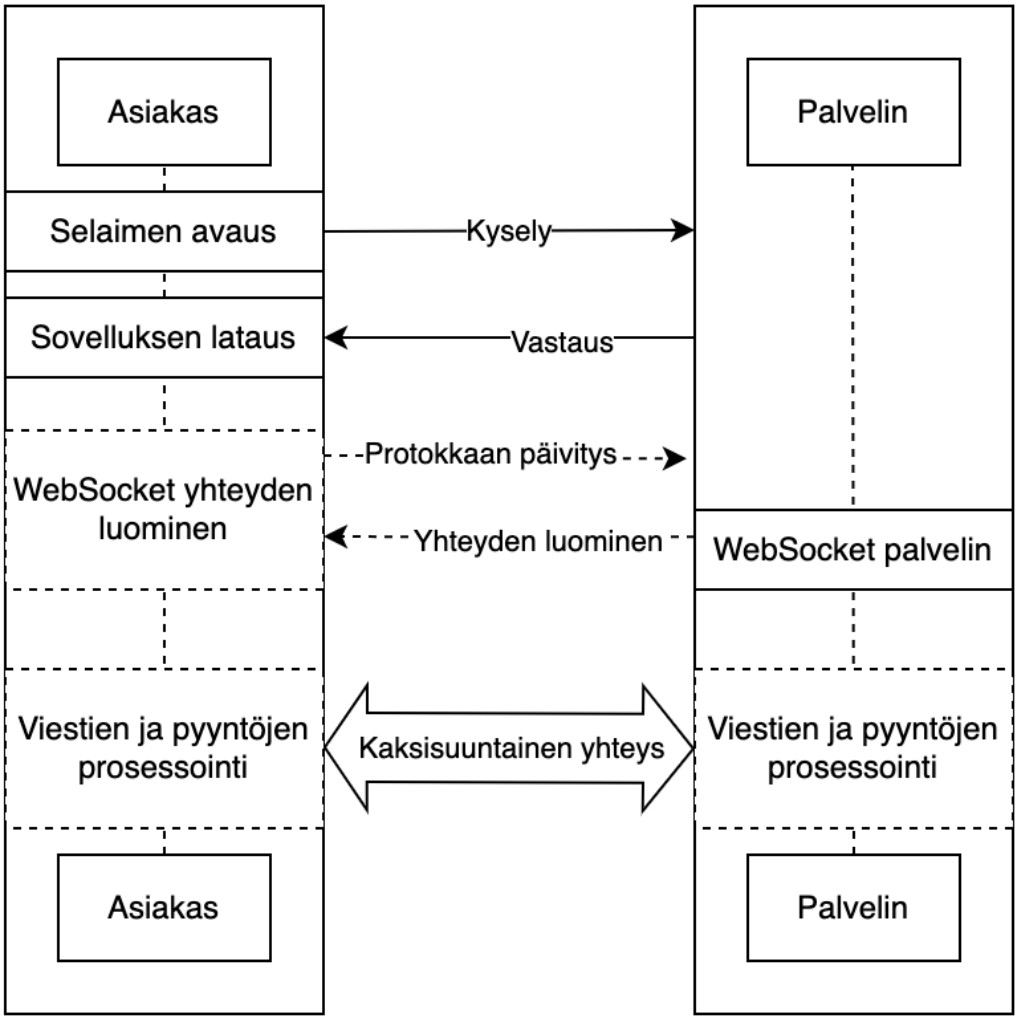
resurssien allokointiongelmia palvelimille. Olemassa olevilla tekniikoilla havaittu latenssi loppukäyttäjälle voi olla alhainen, mutta tehokkuus taustalla jättää paljon toivomisen varaan. Pitkät kyselyt tekevät tarpeettomia pyyntöjä ja ylläpitää jatkuvaa virtaa avautuvista ja sulkeutuvista yhteyksistä palvelimella. (Lombardi 2015.)

6.4 WebSocket

Verkkoprotokollien kehittyessä mahdollistettiin reaaliaikainen tiedon lähettäminen. Tällä hetkellä on käytössä kaksi johtavaa protokollaa, jotka ottavat huomioon ajankäytön, WebSocket ja WebRTC. (Karla & Tarnawski 2019. 1.) Tässä luvussa keskitytään erityisesti WebSocket protokollaan.

Aikaisemmin läpikäydyt HTTP:n yli ajetut tekniikat lähettävät pyyntöjä tasaisin väliajoin riippumatta siitä, saadaanko minkäänlaista vastausta, koska palvelimen tai asiakkaan tilasta ei ole tietoa. WebSocket-protokolla on kuitenkin tässä mielessä erilainen. Palvelimella ja asiakkaalla välillä on avoin yhteys, jonka avulla molemmat voivat lähettää viestejä edestakaisin. (Lombardi 2015.)

WebSocket-protokolla mahdollistaa täysin kaksisuuntaisen kommunikoinnin verkkopalvelimen ja asiakkaan välillä standardoidun yhteyden välityksellä. Yhteys siis luo vapauden sekä asiakkaalle että palvelimelle lähettää tietoa milloin tahansa. (Cameron 2021, 223.) Asiakas muodostaa yhteyden palvelimeen sopivan HTTP-pyynnön avulla, jolla yhteys päivitetään WebSocket-protokollaan. Protokollan päivityksen jälkeen yksi pysyvä yhteys palvelimeen on muodostettu, jonka avulla samanaikainen tiedonsiirto asiakkaan ja palvelimen välillä mahdollistuu. (Karla & Tarnawski 2019. 2.)



KUVIO 17. WebSocket. Havainnollistaa WebSocket-protokollan viestintätapaa asiakkaan ja palvelimen välillä.

Yksi WebSocket-protokollan monista eduista on, että se aloittaa yhteyden palvelimeen yksinkertaisella HTTP-pyynnöllä. WebSocketia tukevat selaimet ja asiakkaat lähettävät palvelimelle pyynnön tietyillä otsikoilla, joissa pyydetään yhteyden päivittämistä WebSocket-protokollaan. (Lombardi 2015.)

WebSocket antaa mahdollisuuden käyttää siis päivitettyä HTTP-pyyntöä ja lähettää tietoja viestipohjaisella tavalla ja TCP:n luetettavuudella. Tämä tarkoittaa yhtä yhteyttä ja kykyä lähettää dataa edestakaisin asiakkaan ja palvelimen välillä ilman mitätöntä resurssien käyttöä. WebSocketin päälle voi myös kerrostaa toisen protokollan, jolla voi tarjota sen turvallisesti Transport Layer Securityn avulla. (Lombardi 2015.)

WebSocket on yksi moderneista tiedonsiirto teknologioista, joka täyttää kriteerit tietojen reaaliaikaisesta siirrosta samalla ylläpitäen tiedon johdonmukaisuuden, ettei tietoa katoa yhteyden aikana. Koska WebSocket on hyvin standardoitu, sitä tuetaan lähes kaikilla verkkoselaimilla. (Karla & Tarnawski 2019. 2.)

Asiakkaan ja palvelimen välillä vaihdettu tieto määritellään sovelluksessa tarkoin. Jokainen siirretty tieto aktivoi myös siihen liittyvät metodit ja toiminnot vastaanottavalla puolella. Tämän kaltainen kommunikointi ja sen käsittely on helppoa kehittää ja toteuttaa. (Karla & Tarnawski 2019. 2.) Vaikka WebSocket käyttää HTTP:tä alkuperäisenä siirtomekanismina, viestintä ei pääty sen jälkeen, kun asiakas on vastaanottanut vastauksen. WebSocket-protokollan avulla voi siis vapautua tyypillisen HTTP-pyyntö-vastaussyklin rajoituksista. (Lombardi 2015.)

WebSoket-protokollalla on monia hyötyjä. Suurin ja merkittävin hyöty on reaaliaikainen ja kaksisuuntainen kommunikaatio (Lyu 2021). WebSocket-protokolla vähentää myös huomattavasti asiakkaan ja erityisesti palvelimen resurssien tarvetta (Lombardi 2015), koska se käyttää kaistaleveyttä huomattavasti vähemmän. WebSocket toimii porteissa 80 ja 443, jotka ovat oletusarvoisia HTTP- ja HTTPS-portteja. Sen vuoksi tällä protokollalla on myös vähemmän ongelmia palomuurien ja ruuhkantasaajien kanssa. (Lyu 2021.) Turvallisuuden näkökulmasta WebSocket toimii myös Transport Layer Securityn (TLS) tai Secure Sockets Layerin (SSL) kautta (Lombardi 2015).

7 POHDINTA

Tämän opinnäytetyön päätavoitteena oli toteuttaa reaaliaikainen äänestyssovellus. Sovelluksen tuli olla verkkosovellus, jotta sovelluksen asennukselta vältyttäisiin. Tuloksena syntyi DanceVote, joka koostuu Reactilla toteutetusta käyttöliittymästä ja Node.js- ja Express-teknologioilla luodusta palvelimesta.

Sovellus isännöitiin Heroku-alustapalvelulla. Heroku on itselle tuttu ympäristö, johon teki valinnasta luontevan. Kuitenkin tietokannan sisällyttäminen palveluun herättää kysymyksen sovelluksen siirrettävyydestä toiselle alustalle.

Reaaliaikaisuus mahdollistettiin sovellukseen WebSocket-protokollaa hyödyntäen. Skaalautuvuutensa ja kaksisuuntaisuutensa ansiosta WebSocket vaikutti sopivimmalta vaihtoehdolta. Mikään muu läpikäydyistä reaaliaikaisuuden vaihtoehdoista ei mahdollistanut kaksisuuntaista kommunikointia käyttöliittymän ja palvelimen välille. WebSocketin kaksisuuntainen kommunikaatio tuotti myös tehokkaimman ja nopeimman datan siirto nopeuden, jolloin viive äänestyksen päivityksestä äänestyslomakkeelle ei ollut häiritsevä.

WebSocket-protokolla tarjoaa helposti lähestyttävän, reaaliaikaisen ja yksinkertaisen suorituskyvylle optimoidun ohjelmointirajapinnan verkkosovelluksille. Socket.io tyyppisten WebSocket-protokollaan perustuvien kirjastojen avulla yhteyden luotettavuus ja turvallisuus paranee sekä jopa lisää käytettävyyttä ja koodin luettavuutta.

Käyttäen palvelin- ja asiakassovelluksessa samaa kieltä helpotetaan ja optimoidaan tiedonsiirtoon koskevaa käsittelyä. Tekstimuodossa kulkeva tieto JSON objektina on erittäin käytännönläheinen ja luettava JavaScript tietorakenne. Kuitenkin JavaScriptin tyypittömyys tuo omia haasteitaan erityisesti virheiden hallinnassa. Tälle olisi kuitenkin selkeä ja käytetty ratkaisu, TypeScript. TypeScriptillä voidaan kirjoittaa nimenomaan tyypitettyä JavaScript koodia. Se on vankka, turvallinen ja helppo korjata. Lisäksi TypeScript käännetään ennen kuin se muutetaan JavaScriptiksi, joka mahdollistaa virheiden havainnoinnit ennen koodin varsinaista suorittamista. (Monteiro 2018.) TypeScriptin käyttämiselle on siis useampi perustelu tulevaisuuteen,

Palvelin puolelle on kuitenkin noussut kohtuullisen uusi Google kehittämä ohjelmointikieli, Go, jonka ensimmäinen vakaa versio julkaistiin 2021. Go on kieli, joka on rakennettu nimenomaan helpottamaan backend-kehityksen koko työnkulkua. Se parantaan niin suoritusnopeutta, käännösnopeutta, riippuvuuksien hallintaa, samanaikaisuutta ja optimointia. Kääntäminen tapahtuu suoraan binäärimuotoon, jolloin tekstin generoinnilta ja sen kääntämiseltä vältytään. Kääntäjällä on siis vähemmän työtä monimutkaisen ohjausvirran ja logiikan analysoinnissa, joka saa Go:n koodista toisinaan ytimekkään. (Kamal & Bakshi 2021.)

Valintani Node.js:n käyttämiseen liittyi sen tapahtumakeskeisyys säikeisyyden sijaan. Projektin ollessa huomattavasti isompi kehittäjän olisi hyvä arvioida Go:n käyttämistä, sillä Go ei myöskään perustu perinteiseen raskaaseen säikeisyyteen. Säikeisyyden sijaan Go esittelee täysin uuden Goroutines-konseptin. Goroutine on erittäin kevyt säikeen kaltainen suoritusympäristö, joka toimii Go:n ajoaikana käyttöjärjestelmän sijaan. Lyhyesti sanottuna Goroutinet kartoitetaan käyttöjärjestelmäsäikeisiin, josta ne kartoitetaan syvemmälle prosessisäikeisiin, mikä mahdollistaa miljooninen Goroutine-säikeiden suorittamisen suorituskyvystä tinkimättä. (Kamal & Bakshi 2021.) Go on ehdottomasti kieli, jota tulee pitää mielessä tulevaisuutta silmällä pitäen.

WebSocket-protokollan hyödyntäminen mahdollistaa lisäksi sovelluksen helpon jatkokehityksen skaalautuvuutensa ansiosta. Tällä hetkellä reaaliaikaisuutta hyödynnetään ainoastaan äänestyslomakkeen päivittämiseen. Sovelluksella olisi kuitenkin mahdollisuuksia esimerkiksi laajentaa reaaliaikaiseen tulosten näyttämiseen. Lisäksi sovellusta voisi kehittää nimettömän äänestyksen lisäksi epäviralliseen ja viralliseen tuomarointi toimintaan ja lisätä mahdollisuus tuomaroimiselle myös kilpailujen karsintoihin.

DanceVoten ensimmäisessä virallisessa käytössä Swinghearts tapahtumassa äänestäjiä kertyi 40, joka on suhteellisen pieni määrä. Olisi mielenkiintoista nähdä, miten paljon datan lähettämisnopeus vaihtelisi asiakkaiden lisääntyessä. Lisäksi olisi hyvä päästä testaamaan palvelimen kuormitusta laajemmassa mittakaavassa ei pelkästään suhteessa reaaliaikaisuuteen. Mielenkiintoisia tilanteita

olisi erimerkiksi suuret äänestysmäärät ja useamman äänestyksen yhtäaikainen toteuttaminen.

Opinnäytetyössä ei varsinaisesti testattu teknologioiden ja protokollien eroavaisuuksia. Päätökset käytetyistä teknologioista tehtiin olemassa olevan näytön, teorian ja tutkimustulosten perusteella omaa kokemusta sekä osaamista unohtamatta. Uusiin ja erilaisiin projekteihin tulevaisuudessa olisi mielenkiintoista päästä toteuttamaan ratkaisuja samaan asiaan eri teknologialla. Henkilökohtaisella tasolla haluan myös syventyä WebSocketin mahdollisuuksiin ja sen optimointiin vielä paljon, sillä tällä projektilla sain vasta pienen pintaraapaisun aikaiseksi todellisen osaamisen polulla.

Toinen reaaliaikaisuuden mahdollistava moderni teknologia, jonka jätin käsittelemättä tässä opinnäytetyössä, on WebRTC. Se tulee sanoistaa Web RealTime Communication, joka on ilmainen avoimen lähdekoodin hanke. Sen päätarkoitus on tarjota reaaliaikaista viestintää selainten, mobiililaitteiden tai IoT-laitteiden välille käyttäen yksinkertaista APIa. Se on tuettu tärkeimmillä verkkoselaimilla, joko oletuksena tai valinnaisena ominaisuutena ja monet muut ilmoittavat tukevansa sitä tulevaisuudessa. (Karla & Tarnawski 2019. 2.) Tämä teknologia tuli vastaan viime metreillä tätä opinnäytetyötä, ja onkin mielenkiintoinen vaihtoehto tutkia ja testata tulevaisuudessa reaaliaikaisuuden toteuttamiseen. Lisäksi Karlan ja Tarnawskin toteamasta on jo 3 vuotta aikaa, jolloin WebRTC:n tukeminen ja yleisyys on kasvanut, jos se on saanut jalan sijaa ohjelmointikehityksen maailmassa.

Näin lopuksi voidaan todeta, että kehitettäessä kommunikoivaa websovellusta, kehittäjän on hyvä suosia push-tekniikoita perinteisten pull-tekniikoiden sijasta. Pull-tekniikat kuormittavat palvelinta helposti turhaan, jonka vuoksi palvelimen skaalautuvuus heikkenee silmin nähden. Kommunikointi niin palvelimen kuin asiakkaan puolelta nopeuttaa datan siirtoa ja vähentää kuormitusta molempiin suuntiin. Lisäksi pull-tekniikoilla tuotettu reaaliaikaisuus on haastavampaa, sillä kyselyitä tehdään joko paljon lyhyellä aikavälillä tai palvelimen tulee pystyä vastaanottamaan ja käsittelemään monta kyselyä samanaikaisesti.

Coulson ym. toteavat, kuinka verkkoteknologian ja verkkoalustan muutosten mukana pysyminen on yksi vaikeimmista haasteista, joita verkkokehittäjä tai -

suunnittelija kohtaa. (2019.) Suunnittelutrendit muuttuvat ja teknologiat kehittyvät koko ajan. Kaikki selainvalmistajat parantavat jatkuvasti selaimiaan pöytäkoneissa ja mobiililaitteissa ja esiin tulee jatkuvasti uusia ominaisuuksia ja työkaluja parantaen sekä käyttäjä- että kehittäjäkokemusta, jotka tulee ottaa huomioon jo kehitettyjen verkkosovellusten ja -sivujen kanssa.

Onneksi kuitenkin monet selaimet ovat avoimia kehittäjille jo selainten omien kehitysjaksojensa aikana, joka auttaa kehittäjiä ennakoimaan uusien ominaisuuksien testatusta ja käyttöönottoa. Tällöin Coulsonin ym. mielestä nähdään, mitä vaikutuksia muutoksilla voi olla kehitetyille verkkosivuille, jolloin kehittäjät eivät joudu reagoimaan odottamattomasti selaimen päivittyessä (2019). Tämä tarkoittaa sitä, että tuotteita ja palveluita on nykypäivänä vaikea jättää ilman jatkokehitystä ja ylläpitoa. Vuosia kestävän tuotteen luominen modernin teknologioin on siis yksi suurimmista haasteista.

LÄHTEET

Cameron, Neil. 2021. Electronics Projects with the ESP8266 and ESP32. Building Web pages, Applications and WiFi Enabled Devices. New York: Springer Science+Business Media New York. Viitattu 14.3.2022. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/electronics-projects-with/9781484263365/html/499197_1_En_9_Chapter.xhtml

Carey, Scott. 2021. The decline of Heroku. Viitattu 10.4.2022. Vaatii käyttöoikeuden. https://go-gale-com.libproxy.tuni.fi/ps/i.do?p=ITOF&u=tampere&id=GALE|A657650529&v=2.1&it=r

Coulson, Lewis. Jephson, Brett & Larsen, Rob. 2019. The HTML and CSS workshop: a new, interactive approach to learning HTML and CSS. Birmingham, England : Packt. Viitattu 19.4.2022. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/the-html-and/9781838824532/?sso_link=yes&sso_link_from=tampere-university

Danielsson, Patrik & Postema, Tom & Munir, Hussan. 2021. Deployment in Product Development at Axis. Pdf-dokumentti. Viitattu 10.4.2022. https://ieeexplore-ieee-org.libproxy.tuni.fi/stamp/stamp.jsp?tp=&arnumber=9317772

El Kafhali, Said & Salah, Khaled. 2018. Performance analysis of multi-core VMs hosting cloud SaaS applications. Pdf-dokumentti. Viitattu 8.4.2022. Vaatii käyttöoikeuden. https://www-sciencedirect-com.libproxy.tuni.fi/science/article/pii/S092054891730048X

Elrom, Elad. 2021. React and Libraries: Your Complete Guide to React Ecosystem. New York: Springer Science+Business Media New York. Viitattu 9.4.2022. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/react-and-libraries/9781484266960

Filipova, Olga & Vilão, Rui. (toim.) 2018. Software Development From A to Z: A Deep Dive into all the Roles Involved in the Creation of Software. Uud. painos. New York: Springer Science+Business Media New York.
Viitattu 23.2.2022. Vaatii käyttöoikeuden.
https://learning.oreilly.com/library/view/software-development-from/9781484239452

Harms, Rolf & Yamartino, Michael. 2010. The Economics of The Cloud. Viitattu 10.4.2022. https://news.microsoft.com/download/archived/presskits/cloud/docs/The-Economics-of-the-Cloud.pdf

Heroku. 2022. Heroku. Viitattu 10.4.2022. https://www.heroku.com/home, https://www.heroku.com/platform

Hunter, Thomas. 2021. Distributed systems with Node.js: building enterprise-ready backend services. O'Reilly Media: Beijing. Vaatii käyttöoikeudet. https://learning.oreilly.com/library/view/distributed-systems-with/9781492077282/?sso_link=yes&sso_link_from=tampere-university

Ibanez, Lazaro. 2020. Development: What is an API Contract? Lazaro Ibanez-blogi 1.6.2020. Viitattu 14.3.2022. https://lazaroibanez.com/development-what-is-an-api-contract-683ced58e06f#:~:text=An%20API%20Contract%20is%20the,is%20interesting%20to%20be%20documented.

Kamal, Baheer & Bakshi, Tanmay. 2021. The Ideal Language for Backend Developers. Mc GrawHill: New York. Viitattu 11.4.2022. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/tanmay-teaches-go/9781264258154/cover.xhtml

Karla, Tomasz & Tarnawski, Jaroslaw. 2019. Soft Real-Time Communication with WebSocket and WebRTC Protocols Performance Analysis for Web-based Control Loops. Pdf-dokumentti. Viitattu 8.4.2022. https://ieeexplore-ieee-org.libproxy.tuni.fi/stamp/stamp.jsp?tp=&arnumber=8864680

Kozyra, Nathan. 2016. Learning Go web development: build frontend-to-backend web applications using the best practices of a powerful, fast, and easy-to-deploy server language. Birmingham: PACKT Publishing. Viitattu 19.4.2022. Vaatii käyttöoikeuden. https://web-p-ebscohost-com.libproxy.tuni.fi/ehost/detail/detail?vid=0&sid=6d2454ce-0c3f-4106-b001-b14ecec3e68f%40redis&bdata=JkF1dGhUeX-BlPWNvb2tpZSxpcCx1aWQmc2l0ZT1laG9zdC1saXZlJnNjb3BlPXNpdGU%3d

Lombardi, Andrew. 2015. WebSocket. Sebastopol, CA: O'Reilly. Viitattu 25.4.2022. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/websocket/9781449369262/

Lyu, Shing. 2021. Practical Rust Web Projects. Building Cloud And Web-Based Applications. New York: Springer Science+Business Media New York. Viitattu 14.3.2022. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/practical-rust-web/9781484265895/html/Cover.xhtml

McFedries, Paul. 2019. Web design playfround: HTML + CSS the interactive way. Shelter Island, New York : Manning. Viitattu 19.4.2022. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/web-design-playground/9781617294402/?sso_link=yes&sso_link_from=tampere-university

Mell, Peter & Grance, Timothy. 2011. NIST Special Publication 800-145. The NIST Definition of Cloud Computing. Pdf-dokumentti. Viitattu 13.3.2022. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

Monteiro, Fernando. 2018. Hands-On Full Stack Web Development with Angular 6 and Laravel 5: Become Fluent in Both Frontend and Backend Web Development with Docker, Angular and Laravel. Birmingham: Packt Publishing, Limited. Viitattu 19.4.2022. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/hands-on-full-stack/9781788833912/?sso_link=yes&sso_link_from=tampere-university

NodeSource. 2018 edition. Node x Numbers. Viitattu 9.4.2022. https://nodesource.com/node-by-numbers

Npm Trends. 2022. @angular/core vs jquery vs react vs vue. Viitattu 9.4.2022. https://www.npmtrends.com/@angular/core-vs-react-vs-vue-vs-jquery

Parody, Liz & Villa, Marian. 2021. Node By Numbers 2020. Viitattu 9.4.2022. https://nodesource.com/blog/node-by-numbers-2020#:~:text=js%20Versions%20Downloads%20in%202020,js%20Binary%20Downloads%20in%202020.

Powell, Thomas. 2010. HTML & CSS : the complete reference.
New York : McGraw-Hill. 5. painos. Viitattu 20.4.2022. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/html-css/9780071496292/?sso_link=yes&sso_link_from=tampere-university

StackOverFlow. 2021. 2021 Developer Survey. Viitattu 9.4.2022. https://insights.stackoverflow.com/survey/2021

StackOverFlow. 2020. 2020 Developer Survey. Viitattu 9.4.2022. https://insights.stackoverflow.com/survey/2020

Street Smart Swing, 2022, What is a jack and jill swing dance competition? Viitattu 23.2.2022. https://jaminjackson.com/what-is-a-jack-and-jill-competition/

William, Owen. 2015. TNW Latest News. Node.js and io.js are settling their differences, merging back together. Viitattu 9.4.2022. https://thenextweb.com/news/node-js-and-io-js-are-settling-their-differences-merging-back-together

Williams, Bill. 2012. The Economics of Cloud Computing. Indianapolis: Cisco Press. Viitattu 10.4.2022. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/the-economics-of/9780132904186/copyright.html

Word Swing Dance Council (WSDC), 2022, Jack Carey | 1991. Viitattu 23.2.2022. https://www.worldsdc.com/about/hall-of-fame/jack-carey-1991/

Word Swing Dance Council (WSDC), 2021, Unravelling the Mystery of the Relative Placement Scoring System. Pdf-dokumentti. Viitattu 23.2.2022. https://www.worldsdc.com/wp-content/uploads/2021/04/Relative-Placement-12-20-20-Rev-5-1-10-1.pdf