Valtteri Lilja

Biljardin turnausjärjestelmä AC Lunalle

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

26.9.2018

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Valtteri Lilja Biljardin turnausjärjestelmä AC Lunalle 37 sivua 26.9.2018
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Vesa Ollikainen

Insinöörityön tavoitteena oli kehittää biljardin turnausjärjestelmä AC Lunalle. Samalla haluttiin perehtyä Full Stack JavaScript kehittämiseen ja verkkosovelluksen julkaisemiseen pilvipalvelussa. Mahdollisimman hyvä lopputuloksen vuoksi tuli työssä ensin perehtyä verkkosovelluksiin ja niiden historiaan.

Työn tarve nousi esille AC Lunassa pelattujen biljarditurnausten ja -liigojen nykyisestä kirjanpidosta, joka oli sekalainen määrä erilaisia Facebook ryhmiä ja Google Docs dokumentteja. Kirjanpidon ylläpito vaati järjestävältä taholta kohtuuttoman määrän manuaalista työtä, jonka vuoksi oli perusteltua korvata kirjanpito tehokkaammalla vaihtoehdolla. Korvaavan järjestelmän päätavoitteina oli siirtää kaikki kirjanpito yhteen paikkaan ja vähentää manuaalisen työn määrää.

Työn ensimmäisessä vaiheessa selvitettiin liigojen ja turnausten asettamia vaatimuksia ja tutkittiin olisiko markkinoilla valmista tuotetta. Vaatimukset kuitenkin osoittautuivat niin spesifeiksi, että sopivaa järjestelmää ei löytynyt. Näin päädyttiin toteuttamaan oma kustomoitu järjestelmä.

Järjestelmän toteutustavaksi valittiin verkkosovellus. Taustatutkimuksessa perehdyttiin verkkosovelluksiin yleisesti ja niiden nousua yhdeksi nykyhetken yleisimmistä sovellusmalleista. Tutkimustavoitteiden vuoksi työ tehtiin kokonaan JavaScriptillä.

Sovellus toteutettiin modernina SPA-verkkosovelluksena. Sovellus koostui verkkopalvelimella toimivasta REST API:sta ja tietokannasta sekä selainpohjaisesta käyttöliittymästä. Verkkopalvelin toteutettiin Node.js:llä ja Expressillä. Käyttöliittymä tehtiin käyttäen Reactia ja Reduxia. Tietokannaksi valikoitui MariaDB, koska relaatiomalli soveltui hyvin toteutettavaan järjestelmään.

Insinöörityön tekeminen syvensi osaamista JavaScriptin parissa ja yleistä tietoutta verkkosovelluksista. Järjestelmän toteutus onnistui kiitettävästi ja loppukäyttäjät ovat ottaneet sovelluksen hyvin vastaan.

Avainsanat	Verkkosovellus, React, Redux, JavaScript

Author(s) Title Number of Pages Date	Valtteri Lilja Billiard tournament management system for AC Luna 37 pages 26 September 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Vesa Ollikainen

The goal of this project was to create a tournament management system for AC Luna billiards club. From investigative point of view the goal was to learn more about Full Stack JavaScript development and how to release a working application via cloud services.

Need for this system rose from the current state of scorekeeping in the league and tournaments held in AC Luna. Before the system the score was kept in multiple Facebook groups and Google Docs documents. This required a lot of effort from the persons managing the leagues and tournaments. Goals of the new system were to decrease the amount of manual work and move all of the scorekeeping to one place.

First it was necessary to see if there were any readymade systems on the market. Unfortunately, none of the systems matched the requirements of the league because of the specific tournament format of the league. Given the conclusion, it was justified to create own system for the tournament management and score keeping.

Easiest option to create the system was a web application. To create a web application for the task in hand, it was necessary to research web applications and their history. To meet the research goals of the project the web application was created fully with JavaScript.

The best possible architecture for the web application was a SPA web application. The server side of the application consisted of web server that handled the REST API and a MariaDB relational database. REST API was created with Node.js and Express. Client side was created with React and Redux.

By completing this projected the skills increased vastly both in JavaScript and in general web application development. Concrete result was the working tournament management system that the end users have been very happy with.

Keywords	Web application, React, Redux, JavaScript

Sisällys

Lyhenteet

1 Johdanto 1

2 Biljardin turnausjärjestelmä 1

2.1 Biljardi lajina 2
2.2 Turnausjärjestelmältä vaadittavat ominaisuudet 2
2.3 Tarjolla olevat järjestelmät 4

3 Web-sovellus 5

3.1 Historia ja verkkosovellusten kehittyminen 5
3.2 Verkkosovelluksen rakennustarvikkeet 6
3.2.1 HTML ja CSS 7
3.2.2 JavaScript 7
3.2.3 Palvelimet 8
3.2.4 REST API 10
3.2.5 Tietokannat 10
3.3 Klassisesta moderniin 12
3.4 Tietoturva 15
3.5 Pilvipalvelut 16

4 Sovelluskehykset ja kirjastot 18

4.1 React Reduxilla höystettynä 19
4.2 Node.js ja Express 22
4.3 MariaDB 25

5 Sovelluksen toteuttaminen ja toteutettu sovellus 25

5.1 Jatkokehitys 32

6 Yhteenveto ja pohdinta 32

Lähteet 35

Lyhenteet

JS	JS eli JavaScript on ohjelmointikieli. Sen nykymuoto on dynaamisesti tyypitetty, tulkattava oliopohjainen komentosarjakieli, jonka syntaksi perustuu löyhästi C-ohjelmointikieleen. Käytössä selaimissa ja palvelinpuolella.
PHP	PHP (Hypertext Preprocessor) on Perlin kaltainen ohjelmointikieli, jota käytetään erityisesti web-palvelinympäristöissä dynaamisten web-sivujen luonnissa.
Java	Java on Sun Microsystemsin kehittämä teknologiaperhe ja ohjelmistoalusta, johon kuuluu muun muassa laitteistoriippumaton oliopohjainen ohjelmointikieli sekä ajoaikainen ympäristö virtuaalikoneineen ja luokkakirjastoineen.
CSS	Cascading Style Sheets (CSS) on verkkosivujen tyyliasun luomiseen käytettävät dokumentti tai dokumentit, jotka muodostavat säännöt verkkkosivun tyyliasulle.
SQL	Structured Query Language (SQL) on IBM:n kehittämä standardoitu kyselykieli, jolla relaatiotietokantaan voi tehdä erilaisia hakuja, muutoksia ja lisäyksiä.
NoSQL	NoSQL (Not only SQL) on käsite, jolla kuvataan perinteisestä relaatiomallista (kirjaimellisesti sen Structured Query Language -tietokantakieltä eli SQL:aa käyttävistä toteutuksista) poikkeavia tietokantoja.
ACID	Tietokantajärjestelmien eheyden periaate, jonka kirjaimet ovat akronyymi sanoista Atomicity, Consistency, Isolation, Durability.
HTML	Hypertext Markup Language on verkkosivujen rakenteen määrittämiseen ja kuvaamiseen luotu merkintäkieli.
REST API	Representational State Trasfer Application Programming Interface eli REST API tai RESTful API on rajapinta, joka perustuu HTTP protokollaan.

1 Johdanto

Tässä työssä toteutetaan biljardi turnausjärjestelmä AC Lunalle, joka on Vantaan Myyrmäessä toimiva Suomen biljardiliiton jäsenseura. Seuran kotipaikkana toimii Myyrmäenraitilla sijaitseva Bar Cafe Luna. AC Lunan jäsenistä suurin osa kilpailee sekä Suomen Biljardiliiton alaisissa kilpailuissa, että seuran itse järjestämissä yksittäisissä kilpailuissa ja pidemmällä aikavälillä pelattavissa liigoissa. Liigojen tulosseuranta ja kirjanpito on ennen tehty sekalaisella Facebook ja Excel kombinaatiolla, jotka tässä työssä tehty turnausjärjestelmä korvaa kokonaisuudessaan. Turnausjärjestelmän päätarkoitus on vähentää järjestävän tahon manuaalisesti tekemän työn määrää liigan hallinnassa ja järjestämisessä.

Turnausjärjestelmä toteutettiin selaimessa toimivana modernina HTML5- ja JavaScript-pohjaisena verkkosovelluksena. Järjestelmän asettamat vaatimukset katetaan täysin tällä teknologialla, eikä tällainen sovellus aseta minkäänlaisia rajoitteita esimerkiksi käyttöjärjestelmän tai käytettävän laitteen suhteen. Alussa esitellään biljardi lajina ja toteutettavan turnausjärjestelmän asettamat vaatimukset. Tämän jälkeen perehdytään verkkosovelluksiin yleisesti ja niiden matkaan yhdeksi yleisimmistä nykypäivän sovellustyypeistä. Verkkosovelluksen rakennuspalikoita tarkastellaan syvällisemmin ja paneudutaan erityisesti työssä käytettyihin sovelluskehyksiin ja kirjastoihin. Lopuksi tarkastellaan työssä toteutettua sovellusta ja käydään läpi työn yhteenveto.

Työn tavoitteena oli myös perehtyä verkkosovelluksiin yleisesti ja erityisesti moderniin JavaScript pohjaiseen Fullstack kehittämiseen ja siinä käytettyihin teknologioihin ja kirjastoihin. Toteutettu järjestelmä julkaistiin pilvipalvelussa.

2 Biljardin turnausjärjestelmä

Bar Cafe Lunassa on jo pitkään pelattu AC Lunan toimesta erilaisia biljardikilpailuja läpi biljardikauden. Kilpailut vaihtelevat joka viikko pelattavista viikkokilpailuista aina muutaman kuukauden ajanjaksolla järjestettäviin isompiin turnauksiin. Nämä isot turnaukset ovat olleet osa Lunaliigaa ja niitä on myös itsessään kutsuttu liigoiksi. Tämä järjestelmä tehtiin korvaamaan liigojen kirjanpito, kaikki ottelutulokset ja ranking järjestelmä, mikä on aiemmin hoidettu monilla Facebook ryhmillä ja Google dokumenteilla.

2.1 Biljardi lajina

Biljardi on pallopeli, jota pelataan biljardikepillä verkakankaan peittämällä kivipöydällä, jossa pelialue on rajattu kumivaleilla. Lähes kaikissa biljardilajeissa valleissa on kuusi reikää eli pussia, joihin palloja lyödään. Biljardissa on useita alalajeja ja pelimuotoja, joista tunnetuimmat ovat pool ja snooker. Pool biljardin tunnetuimpia pelejä ovat 8-pallo, 9-pallo, 10-pallo ja straight pool, joka tunnetaan myös nimellä 14-1. Kaikissa näissä mainituissa peleissä tarkoituksena on lyödä biljardikepillä valkoista lyöntipalloa eli kiveä ja pussittaa kohdepallot pöydän kuuteen pussiin. Kohdepallojen määrä ja lyöntijärjestys vaihtelevat pelimuodoittain. Kaikissa muissa mainituissa peleissä paitsi straight poolissa pelin voittaa voittamalla ennalta sovitun määrän eriä. Straight poolissa pelaajaan tulee vastaavasti pussittaa ennalta sovittu määrä palloja pelin voittoon ja peliä pelataan vain yksi erä. Lunassa järjestetyissä liigoissa on pelattu vuorollaan kaikkia näistä peleistä.

2.2 Turnausjärjestelmältä vaadittavat ominaisuudet

Yleisiä turnausmalleja on olemassa monia erilaisia. Biljardissa turnaukset pelataan usein niin sanotulla tuplaeliminaatiolla. Tuplaeliminaatio (kuva 1) on turnausmalli, jossa kaikki pelaajat arvotaan turnauskaavioon, jossa he etenevät voittamalla pelejä. Ensimmäisellä kierroksella pelinsä voittaneet pelaajat siirtyvät niin sanotulle voittajien puolelle ja hävinneet häviäjien puolelle. Jos pelaaja häviää kaksi ottelua, on hän ulkona turnauksesta. Tuplaeliminaatio voidaan pelata loppuun sellaisenaan tai toinen vaihtoehto on tuplaeliminaationa pelattava karsinta, jossa pelaajamäärän koosta ja sovitusta riippuen valittu määrä pelaajia etenee kertaeliminaatioon eli cup-kaavioon. Cup kaaviossa pelaaja on ulkona turnauksesta hävitessään ensimmäisen kerran. Kolmas yleinen turnausmalli on lohkomalli. Lohkomallissa kaikki osallistujat jaetaan tasan eri lohkoihin, jossa kukin pelaaja pelaa kaikkia lohkon jäseniä vastaan. Lohkoista edetään turnauksen seuraavaan vaiheeseen, joka voi olla joko kerta- tai tuplaeliminaatio valitusta turnausmallista riippuen.

Kuva 1. Tyhjä tuplaeliminaatio kaavio [31].

Lunaliiga on elokuusta toukokuun loppuun kestävä Bar Cafe Lunassa pelattava biljardiliiga. Kausi koostuu neljästä isommasta turnauksesta (myös liigaksi kutsuttu) ja kolmesta pienemmästä turnauksesta eli sprintistä. Isompi turnaus kestää noin 3kk ja pienempi turnaus pelataan alusta loppuun yhden päivän aikana. Molemmista saa ranking pisteitä kaudelle. Kauden lopussa pelataan finaali, jossa kahdeksan parhaiten sijoittunutta pelaajaa ottaa mittaa toisistaan cup tyyppisessä turnauksessa. Jokaisessa turnauksessa pelataan eri peliä, jotka ovat aiemmin mainitut 8-pallo, 9-pallo, 10-pallo ja 14-1. Pienemmissä turnauksissa ei pelata straight poolia.

Isommissa turnauksissa pelaajat saavat käydä pelaamassa keskinäiset pelinsä itse valitsemanaan ajankohtana itse valitsemassaan pelipaikassa. Iso turnaus sisältää neljä vaihetta: lohkovaiheen, eliminaation, finaalikarsinnan, ja finaalin. Jokaiselle liigan vaiheelle paitsi finaalille annetaan tietty aikaraja, johon mennessä pelaajien tulee pelata omat pelinsä kyseisessä vaiheessa. Finaali taas suoritetaan perinteisempään tapaan yhden päivän aikana Bar Cafe Lunassa. Isojen turnausten maksimi osallistujamäärä on ollut 32 pelaajaa, jotta turnaus ehditään varmasti pelata annettuna aikana loppuun. Lunaliigan turnaukset pelataan tasoituksellisina kilpailuina, koska osallistujien joukko on kirjava aina kilpapelaajista harrastelijoihin. Osallistumisesta on kerätty nimellinen maksu.

Ison turnauksen alussa jokaiselle pelaajalle määrätään tasoitus, jonka katsotaan heijastavan hänen taitotasoaan. Tasoitukset ovat jo voitettuja eriä paitsi straight poolissa. Straight poolissa pelaajalle annetaan alussa hänen taitotasoaan vastaava määrä jo pussitettuja palloja, jolloin esimerkiksi sataan pisteeseen pelatessa peli voi alkaa tilanteesta 0-50. Pelaajien tasoituksia ei muuteta kesken käynnissä olevan turnauksen, mutta kauden aikana tasoitusta voidaan muuttaa suuntaan tai toiseen jos näin tarpeelliseksi kat-

sotaan. Kun pelaajille on määrätty tasoitukset, heidät arvotaan täysin satunnaisesti neljään eri lohkoon. Lohkon kaikki pelaajat pelaavat toisiaan vastaan. Jos pelaaja pelaa kaikki lohkovaiheen pelinsä, ansaitsee hän 4 bonuspistettä kausirankingiin. Bonus pisteiden avulla pyritään välttää pelaamattomia pelejä, joilla osa pelaajista voisi taktikoida sijoitustaan lohkossa keskinäisten pelien toimiessa ratkaisevan tekijänä tasatilanteessa. Lohkojen voittajat pääsevät suoraan finaalivaiheeseen ja muut pelaajat siirtyvät joko finaalikarsintaan tai eliminaatioon. Finaalikarsintaan siirtyvät pelaajat jotka jäivät lohkon sijoille 2-5 ja pelaajat sijoilta 6-8 siirtyvät eliminaatiovaiheeseen. Eliminaatio pelataan cup tyylisesti kertaeliminaationa ja finaalikarsinta tuplaeliminaationa. Finaali pelataan kertaeliminaationa. Pelaajat saavat pisteitä kausirankingiin saavutetun sijoituksen mukaan. Isommista turnauksista saa enemmän pisteitä kuin pienistä. Pienistä turnauksista ei myöskään saa bonus pisteitä.

2.3 Tarjolla olevat järjestelmät

Ennen turnausjärjestelmän toteutusta kartoitettiin mahdolliset markkinoilla jo olevat mahdollisesti soveltuvat järjestelmät. Ehdolle nousi kaksi eri järjestelmää Challonge (challonge.com) ja Cuescore (cuescore.com). Challonge on yleinen turnausten hallinnointiin luotu järjestelmä, jota käytetään paljon esimerkiksi e-sport turnauksissa [31]. Cuescore taas on maailmalla yleisesti käytetty biljardin turnausjärjestelmä [32]. Kumpikin näistä olisi soveltunut todella helposti yksittäisen turnauksen järjestämiseen, mutta ranking pisteytyksen toteuttaminen halutulla tavalla ei olisi onnistunut kummassakaan. Kumpikaan järjestelmistä ei myöskään tarjoa mahdollisuutta eliminaatiovaiheen toteuttamiseen saman turnauksen alaisuudessa. Eliminaatiovaihe on ollut suosittu pelivaihe pelaajien keskuudessa, koska eliminaatiovaiheen voittaja on saanut ilmaisen osallistumisen seuraavaan isoon turnaukseen. Lunaliigojen toteuttaminen on myös vaatinut paljon manuaalista työtä järjestävältä taholta, minkä vuoksi haluttiin minimoida järjestäjiltä vaadittava panostus turnausten vetämisessä. Yhtenä vaatimuksena oli, että pelaajien tulee voida ilmoittaa omat pelituloksensa itse järjestelmään ja toisena kaikkien turnausvaiheiden välisten siirtymien tulee tapahtua mahdollisimman automaattisesti.

Oman haasteensa toi myös muutama spesifi Lunaliiga kohtainen sääntö. Esimerkiksi kun isommat turnaukset pelataan usean kuukauden aikana ja turnauksen finaali yhtenä päivänä, voi tällöin joku finaaliin karsiutuneista pelaajista olla estynyt pelaamaan liigan finaalipäivänä. Tällöin pelaajan tulee saada finaalista ansaittava pistemäärä, mutta hänet

tulee voida korvata toisella pelaajalla, jotta finaali saadaan pelattua tasaväkisenä. Oman järjestelmän puolesta puhuviksi seikoiksi nousi esiin myös järjestelmän räätälöitävyys: jos formaattia halutaan muuttaa tulevaisuudessa, on se helpompaa itse hallinnoidussa järjestelmässä. Lunaliigaa pelaavien pelaajien joukossa on myös ollut tiettyä vastahakoisuuttaa uusien järjestelmien käyttöönotossa, joten järjestelmä ja sen käyttö haluttiin tehdä pelaajille mahdollisimman helpoksi. Oma järjestelmä myös mahdollistaa paremmin statistiikan keräämisen, jos tällaista halutaan alkaa jatkossa tekemään.

3 Web-sovellus

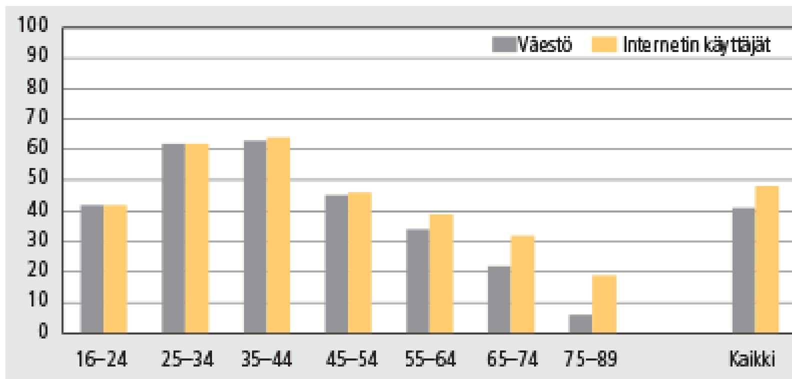
Web-sovellus eli verkkosovellukset ovat sovellusmalli, jossa käyttäjän käyttämä sovellus toimii verkkoselaimessa. Verkkosovelluksen ero verkkosivuun on interaktiivisuus. Verkkosivu on puhtaasti informatiivinen, kun taas verkkosovelluksella voidaan suorittaa jokin toiminto, esimerkiksi tilata lääkäriaika. Verkkosovellus voi koostua yhdestä tai useammasta verkkosivusta, jolla vähintään yhdellä voidaan suorittaa jokin toiminto. Sovellusmalli on usein kaksi- tai kolmijakoinen, mikä koostuu esitystasosta eli käyttöliittymästä selaimessa ja verkkopalvelimella toimivasta sovellustasosta. Usein verkkosovellukseen ja sen sovellustasoon liittyy vielä tietovarastotaso, johon sovelluksen tiedot tallennetaan. [1.]

3.1 Historia ja verkkosovellusten kehittyminen

Verkkosovellusten historia on kulkenut käsi kädessä internetin kehityksen kanssa. Kuten myös aikanaan tietokoneen, myös internetin ja verkkosovellusten suhteen eniten niiden yleistymistä auttaneena seikkana voidaan pitää graafisen käyttöliittymän tuloa selaimiin 90-luvun alussa. Tämä mahdollisti internetin ja verkkosovellusten tien kotitalouskäyttöön. [2.]

Internet otti suuria teknologisia harppauksia eteenpäin 2000-luvun alussa, kun niin sanottu IT-kupla puhkesi. Kuplan puhjettua valtiot alkoivat ottaa suurempaa ja vaikutusvaltaisempaa roolia verkossa ja sen kehityksessä. Samanaikaisesti kuplan puhkeamisesta selvinneet teknologiayritykset alkoivat luomaan normeja digitaaliselle kaupankäynnille ja kulttuurille. Teknologioiden kehittyessä oli mahdollista esittää tietoa mitä moninaisemmin keinoin verkossa. Tämä toi internetiä yhä lähemmäs sitä, millainen tiedon valtaväylä ja yleishyödyke se tänä päivänä on. [2.]

Nykyisin verkkosovelluksia käyttää iso osa ihmisiä pitkin maailma ja ne kykenevät jo mitä moninaisempiin eri tehtäviin. Hyvänä esimerkkinä voidaan pitää sosiaalisen median sovellus Facebookia, jota käyttää kuukausittain 2.13 miljardia ihmistä [3]. Verkossa asiointi on nykyään mahdollista niin lähikaupassa kuin valtion virastossa. Esimerkiksi 41 prosenttia suomalaisista oli jo vuonna 2013 asioinut valtion virastoissa sähköisesti (kuva 2). [4].



Kuva 2. Täytetyn virallisen lomakkeen lähettäminen internetissä viimeisen vuoden aikana vuonna 2013. Osuus väestöstä (16–89-vuotiaat) ja osuus netin käyttäjistä Suomessa. Prosenttia. [4]

Näiden viimeisen kymmenen vuoden aikana on nähty jo suuria muutoksia verkkosovelluksissa ja tulevaisuudessa niitä tullaan näkemään varmasti lisää. Suurin etu verkkosovelluksissa verrattuna tavalliseen työpöytäsovellukseen on niiden monialustaisuus. Koska verkkosovellusta käytetään selaimessa, ei sitä tarvitse luoda tietylle tietokone- tai käyttöjärjestelmätyypille. Selainten suorituskyvyn ja teknologiarajoitusten poistuessa yhä useammat sovellukset tulevat olemaan työpöytäsovellusten sijaan verkkosovelluksia. [1.]

3.2 Verkkosovelluksen rakennustarvikkeet

Verkkosovellusten kirjo on kasvanut ja niiden kehittäminen sekä teknologiat ovat muuttuneet huomattavasti viime vuosikymmenten aikana. Mukaan on tullut lukemattomia määriä erilaisia sovelluskehyksiä ja ratkaisumalleja, mutta moni perustava asia on silti säilynyt samana. Verkkosivut piirretään edelleen käyttäen HTML-merkintäkieltä ja muotoillaan käyttäen CSS:n avulla. Sivujen dynaamisuus saadaan aikaan JavaScriptillä, eikä verkkosovellusta voida toteuttaa ilman jonkinlaista palvelinratkaisua.

3.2.1 HTML ja CSS

HTML ja CSS ovat kaksi pääteknologiaa verkkosivujen piirtämiseen ja muotoiluun. HTML (Hypertext Markup Language) on merkintäkieli, jolla voidaan kuvata verkkosivun rakenne. HTML sisältää monia erilaisia valmiita elementtejä tämän tekemiseen, esimerkiksi paragraph (kappale), list (lista) ja table (taulukko) jne. Tällä hetkellä käytössä on HMTL kielestä versio 5. [6.]

CSS on kieli, jolla voidaan kuvata verkkosivujen ulkoasua erillisillä tyylisivuilla. Tyylisivut voivat määrittää erilaisia asioita kuten mm. värit, fontit ja asettelu. CSS:n avulla ulkoasu voidaan muokata näkymään halutulla tavalla näytön koosta riippuen. Esimerkiksi kännykässä verkkosivu halutaan näyttää erilaisena kuin tietokoneen näytöllä. CSS ei ole sidottu HTML:ään millään tavoin vaan sitä voidaan käyttää minkä tahansa XML pohjaisen merkintäkielen kanssa. HTML:n ja CSS:n eriyttäminen helpottaa verkkosivujen ylläpitoa, tyylisivujen jakamista eri verkkosivujen välillä. Niiden eriyttäminen helpottaa myös sivujen luomista erilaisiin ympäristöihin. [5.]

3.2.2 JavaScript

JavaScript (JS) on kevyt, suoraan tulkittu ohjelmointikieli. Sen on kehittänyt Brendan Eich vuonna 1995 ja siitä tuli ECMA standardi vuonna 1997. JS on alkujaan tullut tunnetuksi verkkosivujen ohjelmointiin käytettynä kielenä selaimissa, mutta nykyään sitä käytetään myös palvelinten puolella. JS:n standardi on nimeltään ECMAScript ja vuodesta 2012 lähtien kaikki modernit selaimet tukevat vähintään ECMAScriptin versiota 5.1. Vuonna 2015 ECMAScriptistä julkaistiin kuudes versio ECMAScript 2015, joka tunnetaan nimellä ES6. Nykyään ECMAScriptistä pyritään julkaisemaan uusi versio joka vuosi ja tällä hetkellä on meneillään ECMAScript 2018. [7.]

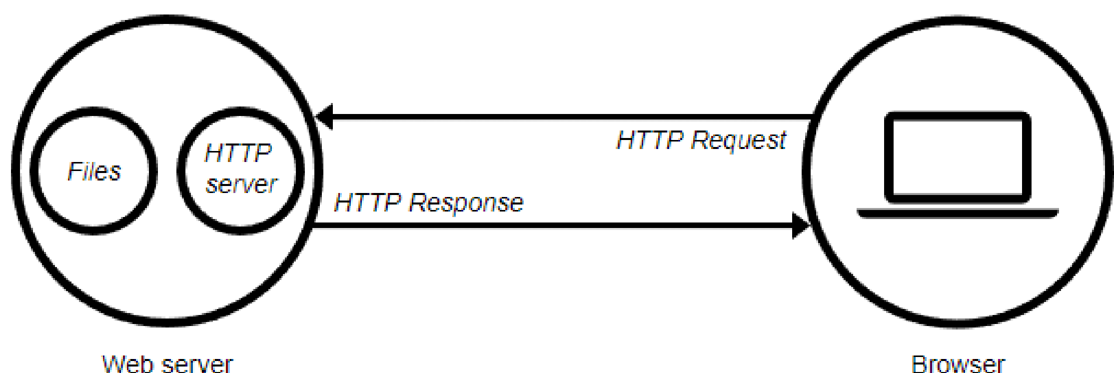
Internetin käytön lisääntyessä, JavaScriptistä on tullut yhä tärkeämpi kieli ohjelmoijille, koska selaimet käyttävät JavaScriptiä. Saman aikaisesti se ei kuitenkaan ole monenkaan ohjelmoijan sydämissä kovin pidetty ohjelmointikieli. Tämä johtuu osittain selaimista ja toisaalta JavaScriptin noususta maailman kartalle hyvin lyhyessä ajassa. JS sisältää kuitenkin myös paljon hyvää. Ohjelmointikielenä se on hyvin dynaaminen ja tehokas. Huvittavinta tässä kaikessa on ehkä se, että sillä voi saada aikaan toimivaa koodia, vaikka itse kielestä ei syvällisesti ymmärtäisi juuri mitään. Kielenä JS on suoraan tulkattu,

dynaaminen ja heikosti tyypitetty. Se pohjautuu löyhästi C-kielen syntaksiin ja vaikka nimestä voisi kuvitella, ei sillä ole mitään tekemistä Java-ohjelmointikielen kanssa. [8.]

3.2.3 Palvelimet

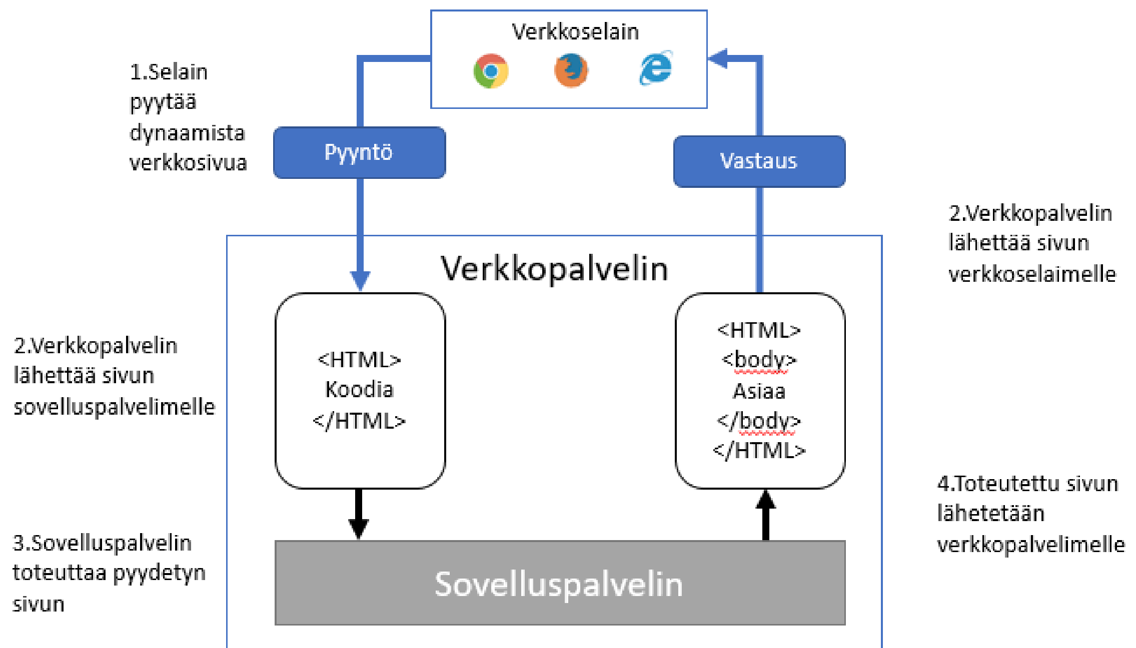
Palvelin on yleisnimitys mille tahansa tietokoneelle, joka välittää tietoa missä tahansa verkossa. Usein kun puhutaan palvelimesta, tarkoitetaan verkkopalvelinta, joka toimii verkkosovellusten ”moottorina” ja vastaa selainten tai muiden järjestelmien lähettämiin pyyntöihin. Muita palvelintyyppejä ovat mm. sähköpostipalvelin, tiedostopalvelin, nimipalvelin ja tulostuspalvelin. Mikä tahansa tietokone voi toimia palvelimena, mutta usein palvelimet ovat erillisiä vain palvelintoimintaan tarkoitettuja tietokoneita. [9.]

Verkkosovelluksen ja verkkosivujen luomiseen tarvitaan verkkopalvelin. Verkkopalvelin voi viitata joko itse tietokoneeseen, vain ohjelmistoon tällä tietokoneella tai näihin molempiin. Verkkopalvelimen tapauksessa ohjelmiston vähimmäisvaatimus on HTTP-palvelin. Yksinkertaisimmillaan verkkopalvelin vastaa selainten lähettämiin HTTP-pyyntöihin (kuva 3). Kun selaimen pyytämä tiedosto löytyy palvelimelta, palvelin vastaa pyyntöön tällä tiedostolla. Jos tiedostoa ei löydy palvelimelta, vastataan pyyntöön HTTP koodilla 404. Palvelimet voivat olla joko dynaamisia tai staattisia. Äsken kuvattu yksinkertainen esimerkki kuvaa staattista verkkopalvelinta. Dynaaminen palvelin eroaa staattisesta palvelimesta siten, että palvelin ei vastaa pyyntöihin suoraan pyydetyillä tiedostoilla, vaan tiedostoja voidaan muotoilla ensin. Usein tämä tarkoittaa sitä, että sivun sisältö halutaan koostaa tiedoista, jotka löytyvät esimerkiksi tietokannasta. Tällöin dynaaminen palvelin käyttää jonkinlaista HTML-sivupohjaa, johon tietokannasta haetut tiedot täydennetään ja selaimen pyyntöön vastataan tällä rikastetulla verkkosivulla. [10.]



Kuva 3. HTTP-pyyntö staattiselle verkkopalvelimelle [10]

Dynaaminen palvelin on monella tapaa joustavampi kuin staattinen verkkopalvelin. Kuvitellaan esimerkiksi Wikipediaa, jossa on miljoonia ja miljoonia artikkeleita. Jos nämä kaikki sivut olisivat staattisia verkkosivuja, olisi ratkaisu hyvin epätehokas ja hankala ylläpitää. Sen sijaan artikkeleiden tiedot haetaan tietokannasta, ja täydennetään HTML-sivupohjaan. Dynaamisen verkkopalvelimen toteutus on toki vaikeampaa kuin staattisen palvelimen, mutta sen edut ovat huomattavat. Dynaamisen palvelimen toteuttamiseen ei siis riitä pelkkä HTTP-palvelin, vaan lisäksi tarvitaan myös muita ohjelmistoja. Usein käytetään sovelluspalvelinta ja tietokantaa. Sovelluspalvelin hoitaa tiedonhaun tietokannasta, muotoilee ja päivittää sen HTML-sivupohjaan ja lähettää muotoillun sivun selaimelle (kuva 4). [10.]



Kuva 4. HTTP-pyyntö dynaamiselle verkkopalvelimelle [11]

Sovelluspalvelimia on olemassa lukuisa erilaisia, lähestulkoon joka ohjelmointikielelle ja käyttötarkoitukselle löytyy omansa ja tärkeintä on valita omaan käyttötarkoitukseen soveltuvin. Erilaisia käyttötarkoituksia voivat olla mm. blogien ylläpito, erilaiset wikit tai verkkokaupat. Lisäksi on olemassa sisällönhallintajärjestelmiä eli CMS:iä (Content management system), jotka ovat geneerisempiä. Sisällönhallintajärjestelmät ovat usein sellaisia, että niiden käyttöönotto ja käyttäminen eivät vaadi niin suurta teknologista osaamista. [10.]

3.2.4 REST API

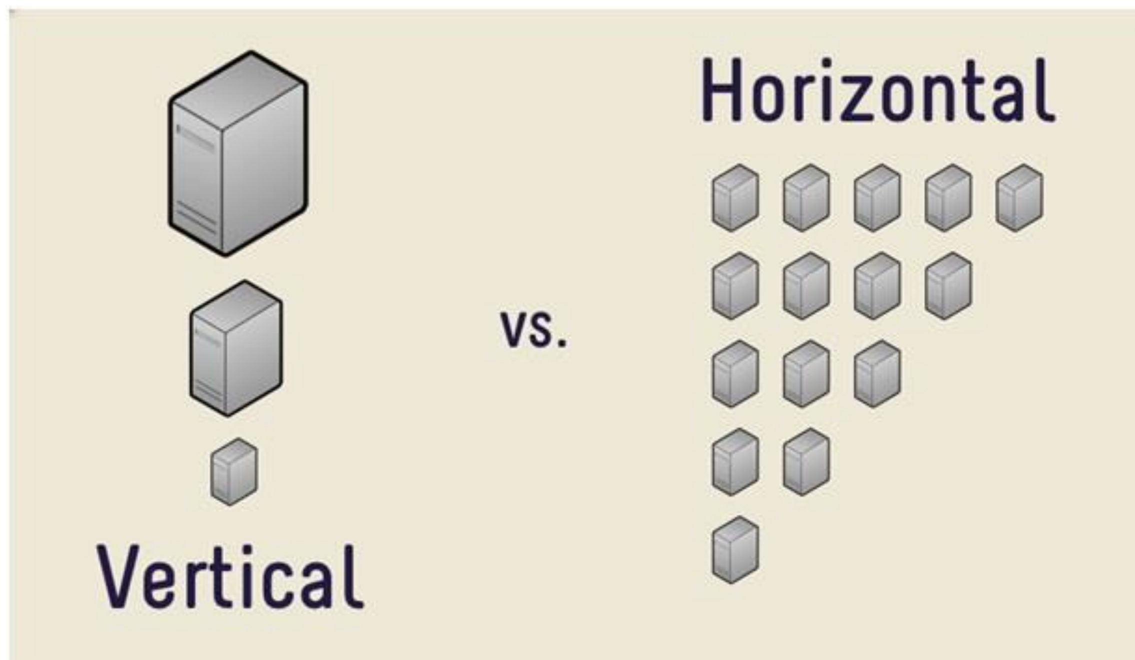
REST API on REST arkkitehtuurimallia noudattava rajapinta, joka on yleisesti käytössä kaikissa HTTP-protokollaa käyttävissä palveluissa. REST API määrittää joukon funktioita, joiden avulla kehittäjät voivat käyttää tehdessään esimerkiksi GET- tai POST-kyselyitä käyttäjän ja rajapinnan välillä. Rajapinnan voidaan sano olevan RESTful siinä vaiheessa, kun se toteuttaa kaikkia REST arkkitehtuurimallin periaatteita. Näistä tärkeimpinä mainittakoon tilattomuus, osapuolten riippumattomuus toisistaan ja mahdollisuus tallentaa tietoa väliaikaisesti. Tilattomuudella tarkoitetaan sitä, että mitään käyttäjän puolen asioita ei tallennetta palvelimelle pyyntöjen välissä. Riippumattomuus on sitä, että kumpikin osapuoli huolehtii vain omista tehtävistään ja voidaan vaihtaa toiseen täysin toisesta osapuolesta riippumatta. Tiedon väliaikaisella tallentamisella pyritään keventämään palvelinten kuormaa ja parantaa käyttäjäpuolen tehokkuutta. Tämän vuoksi jokaisen rajapinnan sanoman tulee ilmaista, voiko tätä tietoa tallentaa väliaikaisesti joko kokonaan tai osittain, jotta käyttäjä ei saa tästä tiedosta riippuen väärää tai vanhentunutta tietoa. [29.]

3.2.5 Tietokannat

Tietokanta on tietorakenne, johon tietoa voidaan tallentaa ja sitten lukea. Useimpien tietokantojen perusyksikkö on taulu, joka sisältää erilaisia kenttiä. Esimerkiksi yrityksen tietokanta voisi pitää sisällään taulut tuotteista ja työntekijöistä. Tuotetaulun kentät voisivat olla esimerkiksi tuotteen nimi, hinta ja tuotteen lukumäärä varastossa, kun taas työntekijätaulu pitäisi sisällään tiedot yrityksen työntekijöiden tiedoista. [12.] Verkkosovelluksen toteuttamiseen ei ole välttämätöntä käyttää tietokantaa, mutta monissa tapauksissa sovelluksen käyttötarkoitus vaatii tietokannan tai jonkin muun tietovaraston käyttämistä.

Tietokannat voidaan karkeasti jakaa kahteen tyyppiin, relaatiotietokantoihin (SQL) ja ei-relaatio tietokantoihin (NoSQL). Näiden kahden tyypin suurin ero on se, että relaatiokanta toimii ennalta määritetyn rakenteen perusteella, kun taas NoSQL-tietokannat eivät vaadi mitään ennalta määrättyä tietomallia, johon tieto tallennetaan. Sen sijaan NoSQL kannoissa tieto tallennettaan erilaisiin kokoelmiin. NoSQL kantojen tyypit ovat dokumenttikanta (Document database), verkkotietokanta (Graph store), arvoparikanta (Key-value store) ja sarakeperhetietokanta (Wide-column store) [13]. Tästä johtuen relaatiokannat skaalautuvat vertikaalisesti NoSQL-tietokantojen skaalautuessa horisontaalisesti. Kaikki

relaatiokannat toteuttavat ACID-periaatetta ja tiedon eheyden säilyttäminen on sisäänrakennettu relaatiomalliin. Relaatiokannan tietoa voidaan myös indeksoida miltei loputtomasti. NoSQL-kannoista vain osa tukee ACID-periaatetta ja tiedon indeksointi on rajallisempaa. Tiedon eheyden säilyttäminen tulee ratkaista muilla tavoin, koska tietokannat eivät itse vaadi sitä. NoSQL kantojen edut tulevat kuitenkin esiin, kun käsitellyt tietomäärät ovat todella suuria tai tiedon rakenne voi muuttua radikaalisti pienellä aikavälillä. Tällöin NoSQL kannat voittavat relaatiokannat lukunopeudessa johtuen niiden horisontaalisesta skaalautuvuudesta (kuva 5). NoSQL kannat myös mahdollistavat tiedon tallentamisen sen jo olemassa olevassa formaatissa. Esimerkiksi JSON muotoinen tieto voidaan tallentaa kantaan sellaisenaan, kun taas relaatiokannoissa kyseinen tieto joudutaan muuntamaan relaatiokannan kenttien mukaisesti. [14.]

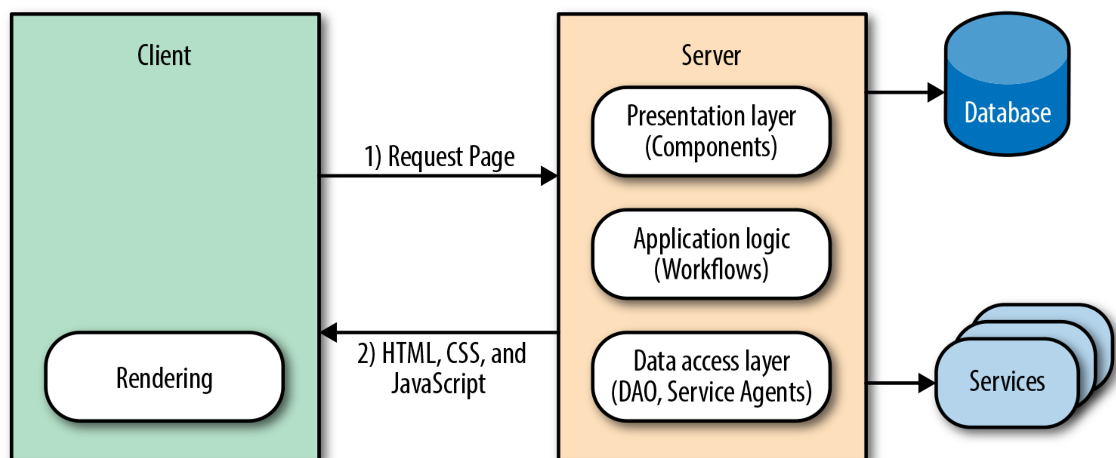


Kuva 5. Horisontaalinen ja vertikaalinen skaalautuvuus

Mitä tahansa sovellusta kehitettäessä tulee miettiä sovelluksen käyttötarkoitusta ja sovelluksen käyttämän tiedon rakennetta. Jos tieto on rakenteellisesti helposti mallinnettavissa relaatiomallilla, eikä tiedon määrä kasvaa suunnattomaksi, on monesti yksinkertaisempaa valita relaatiokanta. Relaatiokannat ovat myös olleet markkinoilla kauemmin, joten niillä on jo olemassa oleva ekosysteemi. [14.]

3.3 Klassisesta moderniin

Yksinkertaisimmillaan klassinen verkkosovellus on vähimmillään yksittäinen verkkosivu, jonka palvelintaso toteuttaa käyttäen jotain palvelinpuolen kieltä, esimerkiksi PHP:tä, Rubya tai Javaa. Palvelimen toteuttama HTML-kuvattu ja CSS-muotoiltu verkkosivu lähetetään vastauksena käyttäjän selaimeen, jossa sovelluksesta riippuen voidaan käyttää vielä JavaScriptiä erilaisten toimintojen suorittamiseen tai tekemään käyttäjäkokemuksesta responsiivisemman (kuva 6). Responsiivisuudella tarkoitetaan sivun muotoutumista käyttäjän tekemien toimintojen mukaan tarkoituksena helpottaa sivun käyttämistä tai lukemista. Tämä voi olla mitä tahansa painikkeesta joka vie takaisin sivun alkuun aina yksittäisen painikkeen värin muutoksiin. [5.]

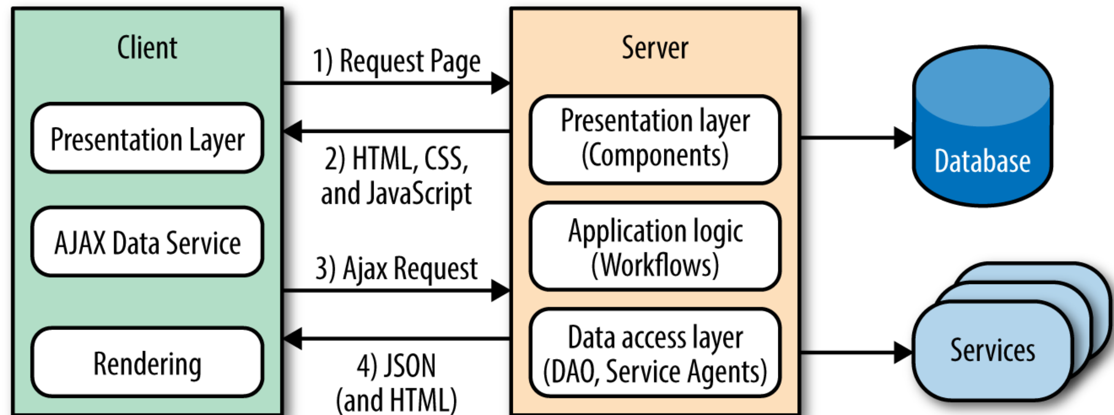


Kuva 6. Klassinen verkkosovellus [5].

Tällainen verkkosovellus noudattaa internetin alkuperäistä ideologiaa. Jokainen selaimesta lähtevä kysely, esimerkiksi navigointi tai tietojen lähettäminen lomakkeella, tuottaa kokonaan uuden verkkosivun. Perinteinen verkkosovellus on sinällään hyvin toimiva ratkaisu, mutta erittäin raskas ja vähemmän responsiivinen verrattuna moderniin verkkosovellukseen. Jokaiseen kyselyyn vastataan kokonaisella verkkosivulla, kun vastaavasti jo olemassa olevalle sivulle voitaisiin päivittää tietoja. [5.]

Seuraava askel verkkosovelluksille oli XMLHttpRequest objektin tulo mukaan kuvioihin. Tämän objektin avulla pystyttiin hakemaan tietoa verkko-osoitteesta ilman koko sivun uudelleen päivittämistä [16]. Tämä myös lisäsi entisestään kiinnostusta verkkosovelluksiin, ja pönkitti niiden asemaa ohjelmistojen maailmankartalla. [5] XMLHttpRequestin

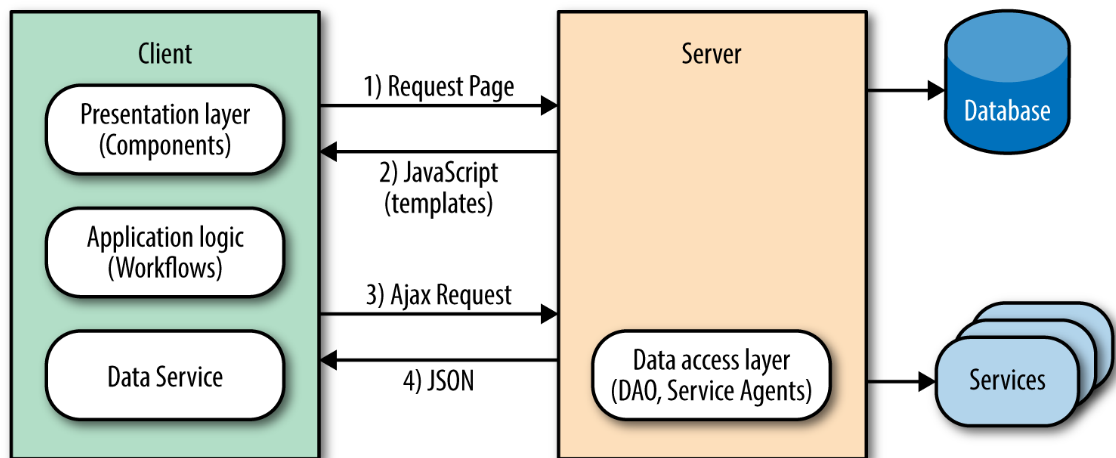
käytöstä ja sitä hyödyntävästä ohjelmoinnista vakiintui vuonna 2005 termi Ajax (Asynchronous JavaScript + XML) [15]. Ajaxin avulla verkkosivuja pystyttiin tekemään responsiivisemmiksi (kuva 7). [5.]



Kuva 7. Klassinen verkkosovellus Ajaxilla [5].

Ajaxin käyttö kuitenkin mutkisti monia asioita verkkosovelluksien tekemisessä. Ajaxia käyttäen voitiin hakea joko palanen HTML:ää tai hakea vain tiedot jotka sivulla haluttiin näyttää. Jälkimmäisen toteuttaminen kuitenkin vaatii erilaisten mallien ja resurssien kahdentamisen käyttäjän puolelle, toisin sanottuna tuplasti koodia ja testattavaa. Yksinkertaisissa ja pienissä sovelluksissa tämä kahdentaminen ei välttämättä haitannut niin paljoa, mutta kun kyseessä on esimerkiksi, suunnattoman suuri verkkokauppa, tulee koodin ylläpidosta huomattavasti hankalampaa. [5.]

Seuraava askel verkkosovelluksille oli SPA (Single-page application) (kuva 8). Näissä koko sovelluksen piirtämiseen liittyvä ja mahdollisesti osa tai jopa koko sovelluslogiikka siirrettiin näyttötasolle käyttäjän selaimeen. Tällöin kaikki piirtämiseen vaadittava koodi voitiin kirjoittaa samalla kielellä eikä sitä tarvinnut kahdentaa kuten klassisessa mallissa. [5.]



Kuva 8. SPA verkkosovellus [5].

Palvelimen kuorma myös väheni huomattavasti, koska sitä käytettiin vain näytön tarvitseman tiedon hakemiseen ja resurssien lähettämiseen selaimelle, kun käyttäjä ensimmäisen kerran siirtyy sivulle. Sivujen tuottaminen kevenee ja nopeutuu huomattavasti, koska jokaista dokumenttia ei tarvitse käsitellä palvelintasolla vaan kaikki piirtäminen tehdään käyttäjän selaimessa näyttötasolla. Vastuujako on siis hyvin selkeä ja toimiva. [5.]

Tämän mallin ongelmat tulevat esiin osittaisena käyttäjäkokemuksen heikentymisenä. Koska kaikki sivut tuotetaan selaimessa, tulee kaikki resurssit lähettää ensin selaimelle ennen kuin sivua voidaan tuottaa. Tämä voi olla hidasta, riippuen käyttäjän verkkoyhteydestä ja sovelluksen koosta. Kun klassiset verkkosovellukset piirtävät sivun suoraan, niin SPA sovelluksissa käyttäjälle voidaan näyttää joko latausikoni, tai vastaavasti ensimmäinen sivu piirretään palvelimen puolella. Jälkimmäisessä vastuujaon rajat alkavat hälvenemään ja koodia joudutaan taas kahdentamaan. Toinen ongelma tulee esiin hakukoneoptimointia ajatellessa. SPA sovellukset eivät ole luonnostaan hakukoneoptimoituja, koska kaikki piirtäminen tapahtuu selaimessa ja sivuston reititykset toimivat # merkin avulla. Esimerkiksi *http://esimerkki.fi/#yksi* ja *http://esimerkki.fi/#kaksi* ovat hakuroboteille sama sivusto. Sivuston tietoja ei myöskään voi näyttää hakukoneella, koska hakurobotille puhtaasta SPA sovelluksesta näkyy vain HTML-runko, johon sivusto piirretään selaimessa. Näiden ongelmien ratkaisemiseksi on löydetty keinot, mutta kaikki ne omalla tavallaan rikkovat SPA arkkitehtuuria ja vaativat ylimääräisiä ponnisteluja. [5.]

Nämä ongelmat kuitenkin ratkaistiin luomalla klassisen ja SPA verkkosovelluksen hybridi, isomorfinen verkkosovellus. Isomorfiset verkkosovellukset ovat JavaScript pohjaisia

verkkosovelluksia, jossa samaa koodia ajetaan sekä selaimessa, että palvelimella. Mitään koodia ei tarvitse kahdentaa, sivuston latausnopeus paranee eikä hakukoneoptimoinnin vuoksi ei tarvitse erikseen ponnistella. Isomorfisen arkkitehtuurin käyttöön otossa kannattaa kuitenkin miettiä kuinka tarpeellista hakukoneoptimointi on ja kuinka kriittistä ensimmäisen sivun latausnopeuden paraneminen on. Esimerkiksi sovelluksissa joihin kirjaudutaan sisään, voi monesti olla yksinkertaisempaa käyttää perinteistä SPA mallia. [5.]

3.4 Tietoturva

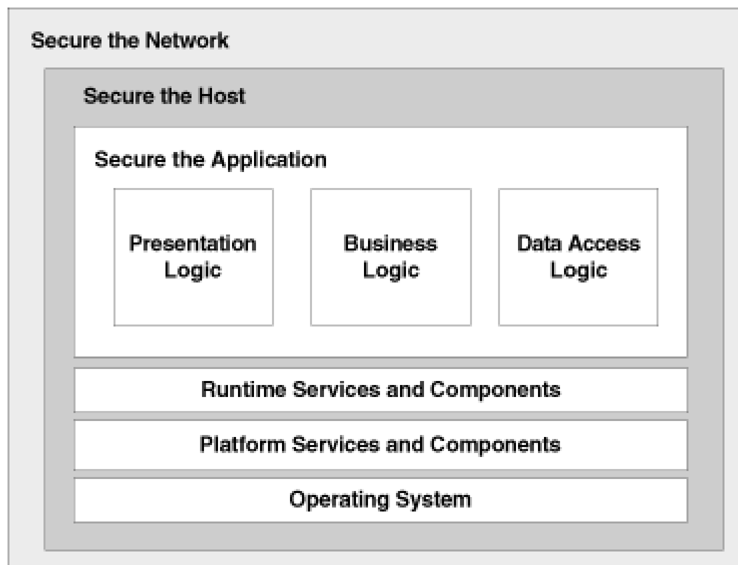
Verkkosovellusten tapauksessa tietoturvan tärkeyttä ei voi painottaa liikaa. Usein ensimmäisenä tulee mieleen ulkoiset uhat, eli virukset, tietomurrot, palvelunestohyökkäykset jne. Näiden kaltaiset iskut usein saavat usein eniten huomiota, mutta usein huomiotta jäävät mahdolliset oman organisaation sisäiset uhat. Usein suurin uhka on ja ongelma on välinpitämättömyys. Verkkosovellusten turvallisuuteen ei kuitenkaan ole yhtä ja oikeaa ratkaisua vaan sen saavuttamista voidaan pitää jatkuvana prosessina. [17.]

Tietoturvan päätarkoitus on resurssien ja tiedon turvaaminen. Se voi olla niin verkkosivujen suojaamista hyökkäyksiltä kuin organisaation maineen suojelemista. Tietoturvan saavuttaminen vaatii jatkuvaa analyysiä mahdollisista uhista. Miten organisaatio voi puolustautua, jos ei tiedä mikä mahdollinen uhka on, tai mistä ne tulevat on? [17.]

Tietoturvan perusteet voi jakaa kuuteen osa-alueeseen:

- **Autentikointi**. Kuka olet? Käyttäjien, palvelinten, palvelujen tms. tunnistamista.
- **Auktorisointi**. Mitä voit tehdä? Kun autentikointi on tehty, pitää tietää mitä autentikoitu taho voi tehdä.
- **Auditointi**. Tapahtumat tulee kirjata, jotta kukaan ei voi kieltää tapahtunutta ja tapahtumia voidaan seurata.
- **Luottamuksellisuus.** Sovelluksen tietojen tulee pysyä vai auktorisoitujen tahojen saatavilla.
- **Eheys.** Tiedon tulee olla turvattua niin vahingossa, kuin tahallisesti aiheutetuilta ei halutuilta muutoksilta.
- **Saatavuus.** Tiedon tulee olla saatavilla auktorisoiduille tahoille.

Verkkosovellusta rakentaessa on arvioitava kokonaisvaltaisesti mahdolliset heikkoudet, niistä aiheutuvat uhat ja potentiaaliset hyökkäykset ja niiden kohteet. Verkkosovelluksen tapauksessa jäsennys on kolmijakoinen. Suojaa verkko, suojaa palvelin ja suojaa sovellus (kuva 9). [17.]



Kuva 9. Verkkosovelluksen turvaaminen [17].

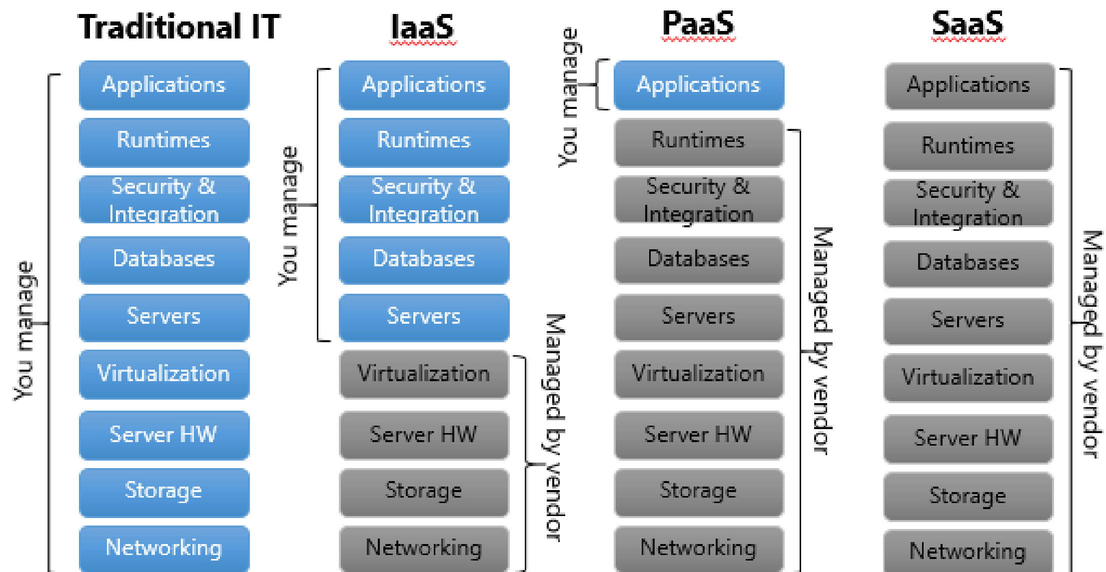
Verkkosovellusten tapauksessa esiin nousee koko ajan uusia uhkia ja kehittäjien tulee pysyä ajan hermolla näiden estämiseksi. Erinomainen paikka uusimman ja pätevimmän tiedon saamiksesi on OWASP (Open Web Application Security Project). OWASP on yleishyödyllinen organisaatio, jonka päätehtävä on sovellusten tietoturvan parantaminen ja helpottaa yksilöitä ja organisaatioita tietoturvallisten sovellusten luomisessa. OWASP esimerkiksi listaa sivuillaan (https://www.owasp.org) erilaisia sovelluksiin kohdistuvia uhkia, julkaisee oppaita ja yleisesti ylläpitää tietoa erilaisista tietoturvaa liittyvistä asioista. [18.]

3.5 Pilvipalvelut

Pilvipalvelut tai pilvilaskenta on yleisnimi IT resursseille, jotka ovat pyydettäessä saatavilla internetin välityksellä. Nämä resurssit voivat olla mitä tahansa palvelimista tietokantoihin tai erilaisiin sovelluksiin. Maksaminen tapahtuu usein joko käyttöajan perusteella tai kiinteillä hinnalla tietylle aikajaksolle. [19.] Markkinoilla on tällä hetkellä useita palveluntarjoajia esimerkiksi Amazon Web Services ja Microsoft Azure.

Pilvipalveluiden käyttämisen suurimmat edut ovat niiden joustavuus ja edullisuus. Yritysten ja organisaatioiden ei tarvitse tehdä suuria ja kalliita laitehankintoja oman IT-infrastruktuurin pystyttämiseen. Sen sijaan kaikki palvelimet ja muut palvelut ovat saatavilla globaalisti ja tarvittavissa määrin. Yrityksen ei myöskään tarvitse huolehtia oman IT-infrastruktuurin ylläpidosta, vaan ainoastaan pitää huolta omista sovelluksista ja keskittyä sen myötä omaan ydinosaamiseen. Omien pilvipalveluiden hallinta sujuu usein verkkosovelluksen kautta ja esimerkiksi uusia palvelimia voi pystyttää yhdellä klikkauksella. Pilvipalvelut ovat mahdollistaneet yrityksille nopeamman, helpomman ja halvemman keinon muokata palveluitaan nykyisessä nopeasti muuttuvassa maailmassa. Ei ole enää tarvetta arvailla palveluiden käyttöastetta ja määrää, vaan kaikki resurssit saadaan helposti joustamaan käytön mukaan. Lisäksi pilvipalvelut ovat saatavilla globaalisti, joten laajentuminen eri markkinoille käy erittäin helposti. [19.]

Pilvipalvelut voidaan jakaa kolmeen eri kategoriaan. Nämä kategoriat ovat IaaS (Infrastructure as a Service), PaaS (Platform as a Service) ja SaaS (Software as a Service) (kuva 10). Alimman tason palvelu IaaS tarjoaa käyttäjälle vain perus resurssit, jotka voivat olla mitä tahansa palvelimista erilaisiin tietovarastoihin. Käyttäjä voi tämän jälkeen muokata näitä resursseja omien tarpeidensa mukaan ja palveluntarjoaja pitää huolta ainoastaan resurssien ylläpidosta, esimerkiksi verkon ylläpidosta ja virtualisoinnista. Seuraava ylempi taso eli PaaS poistaa käyttäjältä tarpeen hallinnoida käyttöjärjestelemiä ja laitetasoa. Palveluntarjoaja antaa käyttäjälle valmiin alustan, jota käyttäjä käyttää, mutta hänen ei tarvitse huolehtia mistään päivityksistä, alustan skaalautumisesta tai mistään mikä vaikuttaa alustan ajamiseen. Ylin taso eli SaaS tarjoaa käyttäjälleen valmiin sovelluksen käytettäväksi. Hyvä esimerkki valmiista SaaS palvelusta on sähköposti. Palveluntarjoaja hoitaa sähköpostipalvelinten, niiden käyttöjärjestelmien ja ohjelmien ylläpidosta ja käyttäjä voi suoraan käyttää sähköpostia huolehtimatta mistään näistä. [20.]



Kuva 10. Pilvipalvelun eri kategoriat versus perinteinen IT [21].

Pilvipalvelut voidaan mieltää joko julkisiksi tai yksityisiksi tai niiden yhdistelmäksi (hybridipilvipalveluksi). Näillä kategorioilla tarkoitetaan pilvipalveluiden hallinnointimallia. Ensimmäisessä koko sovellus ja sen kaikkia liitännäisiä osia ajetaan täysin julkisten palveluiden avulla. Hybridipilvi on muoto, jossa yrityksen tai organisaation jo olemassa olevat resurssit ovat käytössä rinnakkain pilvipalvelun kanssa. Tällainen tilanne voi syntyä, kun halutaan käyttää jo valmiina olevaa IT-infrastruktuuria omana yksityisenä palveluna. Esimerkiksi yritys voi haluta siirtyä käyttämään julkisia pilvipalveluita täysivaltaisesti, mutta siirtymä halutaan tehdä porrastetusti. Private cloud eli yksityispilvi ei varsinaisesti poikkea vanhasta mallista, jossa yrityksillä ja organisaatioilla oli omat konesalinsa. Etuna tässä on se, että yrityksellä on täysi hallinta omiin resursseihinsa, ja he voivat käyttää ja yksilöidä niitä haluamallaan tavalla. Resursseja voidaan myös käyttää mahdollisimman tehokkaasti käyttäen virtualisointia ja sovellushallintaa, kuten pilvipalveluissa muutenkin. [20.]

4 Sovelluskehykset ja kirjastot

Sovelluskehys tai ohjelmistokehys on ohjelmistotuote tietylle kielelle tai ympäristölle, joka muodostaa alustan kehitettävälle sovellukselle. Alusta sisältää jo valmiiksi monia kehitystä helpottavia osia ja komponentteja, joita kehityksessä voi käyttää. Tämä nopeuttaa kehitystä huomattavasti, koska kehittäjän ei tarvitse luoda kaikkea tarvitsemaansa

alusta asti itse. Sovelluskehykset käyttävät usein monia valmiita ohjelmointikirjastoja. Kirjasto eroaa sovelluskehyksestä siten, että kirjasto ei velvoita ohjelmoijaa käyttämään mitään tiettyä kaavaa tai rakennetta ohjelmoidessa, vaan sen sijaan tarjoaa käyttäjälleen apuvälineitä aina luokista yksittäisiin metodeihin. [22.]

Tässä työssä tehtiin käyttämällä JavaScriptillä tehtyjä kirjastoja ja sovelluskehyksiä. Käyttöliittymä tehtiin Reactilla ja Reduxilla. Palvelinpuoli tehtiin käyttäen NodeJS pohjaista ExpressJS sovelluskehystä. Saman ohjelmointikielen käyttäminen molemmilla sovellustasoilla helpottaa koodin kirjoittamista, koska syntaksi on koko ajan sama. JavaScriptin ekosysteemi on myös hyvin runsas ja työkaluja löytyy joka lähtöön.

4.1 React Reduxilla höystettynä

React on Facebookin kehittämä JavaScript pohjainen kirjasto, joka on luotu käyttöliittymien suunnitteluun. Reactilla on hyvin helppo suunnitella kokonaisia käyttöliittymiä tai sitä voi vastaavasti käyttää vain yksittäisten käyttöliittymän palojen kehittämiseen johtuen sen komponenttipohjaisuudesta. Koska React on JavaScript pohjainen, on tiedon esittäminen hyvin yksinkertaista ja sovelluksen tilaa ei tarvitse sotkea verkkosivun DOM:iin. Reactin eduiksi voidaan myös lukea sen valmius tuottaa sivu jo etukäteen palvelimella, joka nopeuttaa sivujen latausnopeutta. Reactista on myös olemassa React Native kirjasto, jolla voi luoda mobiilisovelluksien käyttöliittymiä. [23.]

React on JavaScript pohjainen, mutta sitä on helpompi kirjoittaa käyttämällä XML:ää muistuttavaa JSX-syntaksia. JSX syntaksi kehitettiin Reactia varten, mutta sen käyttäminen on täysin vapaaehtoista. [23.]

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}
```

Esimerkkikoodi 1. Yksinkertainen React komponentti [23].

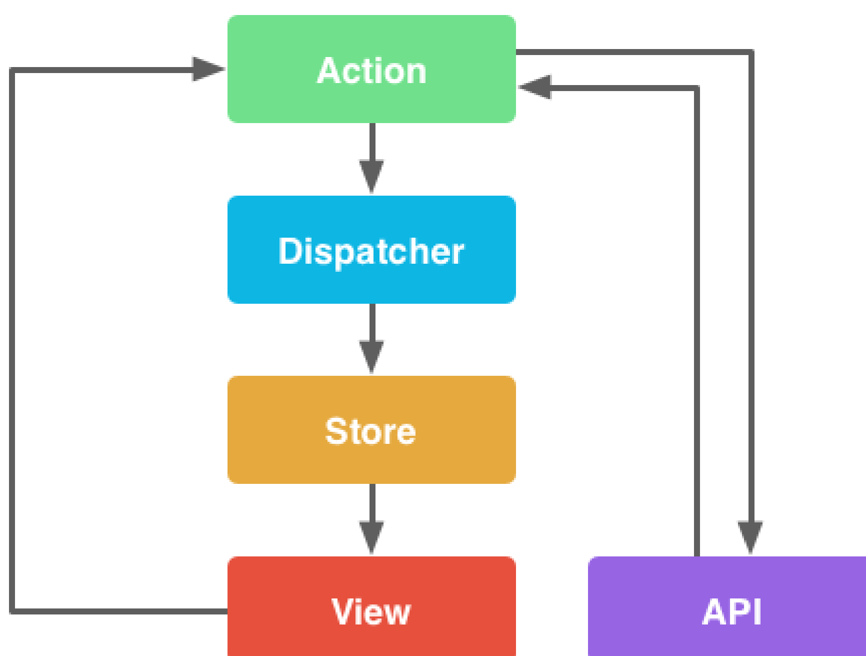
Tämä esimerkkikoodin 1 komponentti luo HTML:n div elementin, jonka sisään tulostetaan komponentille annettu nimi. Komponentin render-funktion return-osio on kirjoitettu JSX:ää käyttäen. Jokaisella komponentilla on oma sisäinen tilansa state ja komponentille luomisvaiheessa annettavat ominaisuudet props, joita komponentti itse ei voi muokata. Esimerkkikoodissa 2 halutaan esitellä sisäisen tilan käyttö komponentissa. Esimerkkikoodin 2 komponentti kysyy käyttäjältä hänen olotilaansa. Komponentti tuottaa painikkeet, jolla käyttäjä voi vastata kysymykseen. Kun käyttäjä painaa painiketta, komponentin sisäinen tila asettuu painikkeen määrittämään tilaan ja tulostaa sen ruudulle.

```
class HowIFeel extends React.Component {

  constructor(props) {
   super(props);
   this.state = {
   mood: ''
   }
   this.setMood = this.setMood.bind(this);
  }

  setMood(mood) {
   console.log(mood);
   this.setState({mood: mood});
  }

  render() {
    return (<div>
      <h2>How do you feel? </h2>
        <div>
         <button onClick={() => this.setMood('Good')}> Good
         </button>
        <button onClick={() => this.setMood('Bad')}> Bad
         </button>
        </div>
       <div>
        I feel {this.state.mood}
```

```
                </div>
              </div>
            );
          }
        }
```

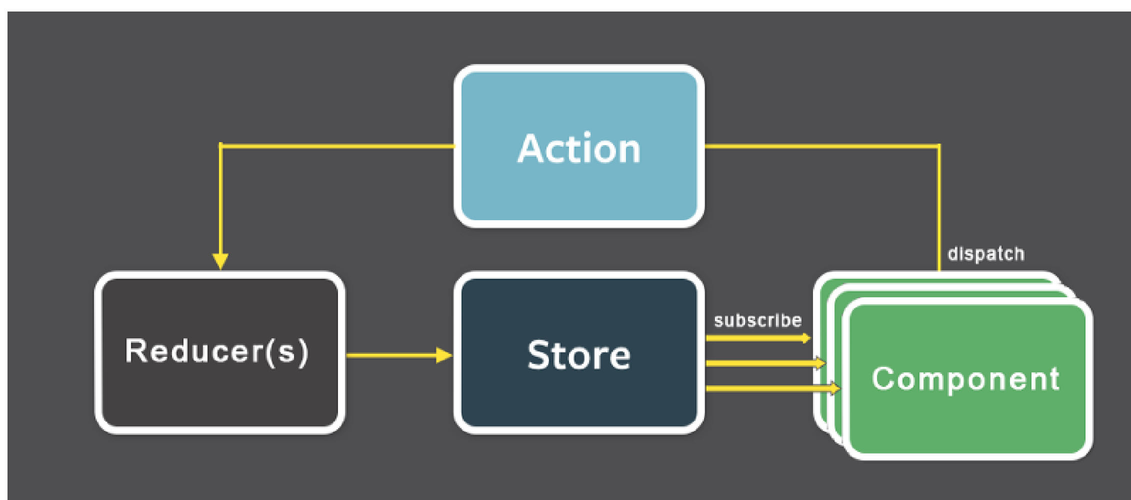
Esimerkkikoodi 2. React komponetti jolla on vaihdettava sisäinen tila.

Reactin komponenttipohjaisuus on samaan aikaan sekä uhka että mahdollisuus. Sovellusten kasvaessa kasvaa myös komponenttien ja tilojen määrä. Kun sisäkkäisiä komponentteja alkaa olemaan liikaa, joutuu sovelluksen sisäistä tilaa usein välittämään hyvin monelle komponentille komponenttipuussa, joka voi käydä raskaaksi. Tätä varten Facebook kehitti Flux arkkitehtuurin sovelluksen tilojen ylläpitämiseen. Flux poikkeaa MVC mallista sen yksisuuntaisen tiedonkulun vuoksi (kuva 11) [24].



Kuva 11. Flux-arkkitehtuurin tiedonkulku [24].

Redux on Flux arkkitehtuuriin pohjautuva kirjasto. Reduxin voi ajatella olevan yksinkertaistettu versio Fluxista, jossa käytetään vain yhtä varastoa (store), joka sitten hallinnoi kaikkia sovelluksen tiloja (kuva 12). Fluxissa vastaavasti voi käyttää useita storeja tilanhallintaan. Reduxissa ei myöskään ole laukaisinta (dispatcher), vaan toiminnot (actions) välitetään käyttämällä funktiopohjaisia tilamuokkaimia (reducers). [25.]



Kuva 12. Reduxin tiedonkulku [25].

Reduxin avulla sovelluksen tilanhallinta helpottuu huomattavasti, mutta sen käyttäminen ei ole välttämätöntä. Sovellusten monimutkaistuessa tilanhallinnan keskittäminen yhteen paikkaan voi helpottaa sovelluksen hallinnointia ja kehittämistä huomattavasti, mutta yksinkertaisissa sovelluksissa reduxin käyttäminen voi monimutkaistaa sovelluksen rakennetta. Reduxia käytettäessä komponenttien hallitsema tilapuu korvataan reduxin tilapuulla. Mikä tahansa komponenteista voidaan liittää varastoon, eikä tilaa tarvitse välittää koko komponenttihierarkian läpi. Usein React ja Redux pohjaisessa sovelluksessa on niin sanottuja tyhmiä komponentteja ja fiksuja komponentteja. Tyhmiä komponentteja ei kiinnostaa muu kuin ulkonäkö ja fiksut komponentit vastaavasti hallinnoivat sovelluksen toimintaa ja esimerkiksi kutsuvat toimintoja ja sitten välittävät tiedon tyhmille esityksestä vastaaville komponenteille. [25.]

4.2 Node.js ja Express

Työssä haluttiin perehtyä Fullstack JavaScript kehitykseen, jonka vuoksi Node.js oli ainoa varteenotettava vaihtoehto verkkopalvelimen toteuttamiseen. Node.js on JavaScriptillä toteutettu palvelimella ajettava tapahtumapohjainen, asynkroninen kirjasto, joka tarjoaa tehokkaat työkalut verkkopalvelimille. Node julkaistiin vuonna 2009 ja on osittain sen ansiota, että JavaScript on muuntautunut vain selaimessa käytettävästä käyttöliittymien ohjelmointiin tarvittavasta kielestä yhdeksi tärkeimpiä nykyajan kieliä. Node käyttää Googlen V8 Javascript moottoria, joka perustuu ES6:een. [26.]

Noden suurimpia etuja on yksisäikeisyys. Monisäikeisyys on yksi suurimmista ohjelmointivirheitä aiheuttavista asioita ja usein näiden virheiden löytäminen sekä korjaaminen voi olla hyvin työlästä. Node pitää kiinni selaimessa käytettävästä mallista. Koodi on ohjeita, joita ajetaan yksi kerrallaan ja mitään ei ajeta rinnakkain. Hitaat operaatiot voisivat jumittaa tällaisen toimintamallin, jos esimerkiksi ladataan isoa tiedostoa. Ratkaisuna tähän ongelmaan on tapahtumapohjainen malli. Kaikki ohjeet eli koodi on ladattu valmiiksi ja kun jotain tapahtuu, ennalta määritetty toiminto suoritetaan. Näin vältetään moniajossa usein ilmenevät ongelmat, esimerkiksi resurssipullonkaulat ja säikeiden välinen kilpailu samoista resursseista. Noden tapauksessa nämä hitaat operaatiot ovat usein luku- ja kirjoitusoperaatioita joko levyltä tai verkosta ja hitauden välttämiseksi käytetään tapahtumapohjaisuuta, asynkronisia rajapintoja ja esteettömiä luku- ja kirjotusoperaatioita. Viimeisin näistä tarkoittaa Noden tapaa käsitellä verkkopyynnöt, missä ohjelma on vapaa tekemään muuta, kun verkkopyyntöä käsitellään. Kun vastaus saapuu, suoritetaan ennalta määritetty toiminto sen käsittelemiseksi. [26.]

Työskentelyä Nodella helpottavat myös monet valmiit moduulit, kirjastot ja työkalut, joita usein palvelinpuolella tarvitsee. Perus luku- ja kirjoitustoimintojen (fs) sekä verkkopalvelin kirjastojen (net, http, https) lisäksi mainittakoon Nodelle uniikki tapahtumakirjasto, joka on pohja hyvin monelle muulle moduulille. Nodella on todella yksinkertaista tehdä asioita ja JavaScriptin modulaarisuus tulee tässä hyvin esille. Esimerkiksi verkkopalvelimen pystyttäminen on hyvin helppoa, joka esitetään esimerkkikoodissa 3. [26.]

```
const http = require('http');
const port = 8080;

const server = http.createServer((req, res) => {
  res.end('Hello, world.');
});

server.listen(port, () => {
  console.log('Server listening on: http://localhost:%s',
port);
});
```

Esimerkkikoodi 3. Verkkopalvelimen pystyttäminen Nodella [26].

Moduulien hallinta on erittäin helppoa sisäänrakennetulla npm paketinhallinnalla, joka on kriittinen osa Node-infrastruktuuria. Npm:n avulla voidaan asentaa moduuleita julkisesta npm rekisteristä, jota hallinnoi npm Inc. Npm:ää voi kuitenkin käyttää myös omiin julkisiin tai suljettuihin projekteihin ja yritys tarjoaakin palveluita, jossa rekisteriä voi käyttää omien moduulien säilyttämiseen. Npm on erittäin helppo tapa hallita omia JavaScript projekteja. Jokaisella npm projektilla on package.json tiedosto, joka sisältää tiedot kaikista projektin riippuvuuksista. Riippuvuudet ovat joko kehityksen aikaisia (devDependencies) tai ajon aikana tarvittavia kokoaikaisia riippuvuuksia (dependencies). Kun kehittäjä asentaa uuden moduulin projektiinsa, npm huolehtii package.json tiedoston päivittämisestä. Moduulit voidaan myös asentaa globaalisti, jolloin ne ovat myös käytettävissä projektin ulkopuolella. Pääasiassa ainoastaan erilaisia työkaluja tulisi asentaa globaalisti ja projektin itse käyttämät moduulit, niin kehityksen aikaiset kuin kokoaikaset moduulit, tulisi pitää sidottuna projektiin. [26.]

Edellisellä sivulla näytettiin, miten verkkopalvelin luodaan käyttämällä Nodea. Nodelle on kuitenkin luotu monia valmiita sovelluskehyksiä verkkopalvelimen pystyttämiseen ja hallintaan, jotka tekevät verkkosovellusten tekemisen vielä helpommaksi kuin pelkän puhtaan Noden käyttäminen. Yksi näistä sovelluskehyksistä on minimalistinen Express ja monet muut sovelluskehykset onkin rakennettu käyttäen Expressiä hyväksi. Express sisältää monet verkkopalvelimen perustoiminnot kuten staattisten tiedostojen jakamisen, reitittämisen ja sovelluksen konfiguroinnin. Expressin minimalistisuus tulee ilmi siinä, että se ei pakota kehittäjää mihinkään tiettyyn muottiin, vaan antaa vapaat kädet projektin rakenteelle ja toteutukselle. Express soveltuu hyvin niin verkkosovelluksien kuin vain REST APIen tekemiseen. Esimerkkikoodissa 4 pystytetään verkkopalvelin käyttäen Expressiä. Kuten voi huomata, se näyttää hyvin paljon samalta kuin puhtaan Noden käyttäminen (ks. esimerkkikoodi 3) [27.]

```
const express = require('express')
const app = express()

app.get('/', (req, res) => res.send('Hello World!'))

app.listen(3000, () => console.log('Example app listening on
port 3000!'))
```

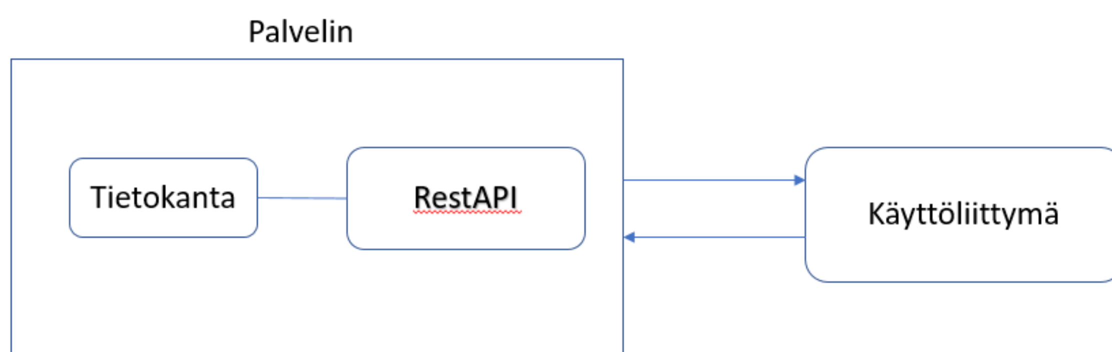
Esimerkkikoodi 4. Verkkopalvelimen pystyttäminen Expressillä. [27.]

4.3 MariaDB

MariaDB on yksi maailman suosituimmista relaatiotietokannoista. Se on kehitetty open source vaihtoehdoksi MySQL:lle sen muuttuessa kaupalliseksi tuotteeksi. MariaDB on yleisesti käytössä niin pankkimaailmasta perus verkkosivujen tietovarastoksi. Ekosysteemi on hyvin vahva johtuen laajasta takautuvasta yhteensopivuudesta MySQL:n kanssa. MariaDB tukee monia tietokantamoottoreita joita ovat mm. innoDB ja MyISAM. MariaDB:n kenties suurimmaksi eduksi voidaan lukea sen open source kehitysmalli, joka takaa sen kehityksen jatkuvuuden ja ekosysteemin monipuolisuuden. [28.]

5 Sovelluksen toteuttaminen ja toteutettu sovellus

Sovelluksen toteuttaminen lähti liikkeelle pyytämällä järjestävältä taholta kaikki tiedot Lunaliigan turnausformaatista ja työnimeksi sovellukselle tuli Lunabrackets. Vaikka monet formaatin asiat olivat jo valmiiksi tiedossa, niin täydellisen vaatimuslistan helpotti toteutusta checklist-hengessä. Epäselvissä kohdissa järjestävään tahoon myös oltiin yhteydessä ja tarkennettiin epäselviä kohtia formaatin säännöistä mm. pelaajien ranking tasatilanteessa ja korvaavan pelaajan valinta. Työssä ei erikseen lähdetty tekemään toiminallista ja teknillistä määrittelyä vaan toteuttaminen tehtiin hyvin pitkälti koodi edellä. Työn toteutus jaettiin karkeasti kahteen osa-alueeseen, käyttöliittymään ja palvelinpuoleen, jotka olivat molemmat oma npm projektinsa. Palvelimen puoli jakaantui vielä erikseen tietokantaan ja REST-rajapinnan luomiseen (kuva 13).



Kuva 13. Sovelluksen jako

Lisäksi sovelluksen tietomalli (kuva 14) eriytettiin muusta koodista ja siitä luotiin oma npm projektinsa, jota sitten käytettiin sekä käyttöliittymä että palvelin puolella. Kaikki koodi tallennettiin git-repositorioon. Git on hajautettu open source versionhallintajärjestelmä, joka on tällä hetkellä maailman yksi yleisimmin käytetyistä versionhallintajärjestelmistä.

```
Bracket.js                    MegaEliminationBracket.js    Tournament.js
DoubleEliminationBracket.js   Player.js                    index.js
Group.js                      RoundRobinBracket.js         package.json
League.js                     Season.js
Match.js                      SingleEliminationBracket.js
```

Kuva 14. Sovelluksen tietomalli.

Käyttöliittymää lähdettiin toteuttamaan Reactilla ja Reduxilla. React oli jo entuudestaan tuttu käyttöliittymäteknologia, mutta sen integroiminen Reduxiin oli alkuun haastavaa. Kun Reduxin toimintatavasta pääsi jyvälle, oli uusien ominaisuuksien ja tilojen lisääminen hyvin helppoa. Lisäksi otin projektissa käyttöön React routerin. React router on Reactille ja Reduxille kehitetty kirjasto, jolla voi tehdä sivuston reititykset JavaScriptillä. Kun käyttäjä navigoi osoitteeseen lunabrackets/leagues ei lähetetä uutta kutsua palvelimelle, vaan siirtymä tehdään puhtaasti JavaScriptillä. Koodin kehittämisen kannalta oli selkeää jakaa tilat ja toiminnallisuudet viideksi eri tilapuuksi (kuva 15), jotka yhdistetään sovellusta ajettaessa yhdeksi tilapuuksi. Pääasiassa Redux toiminnallisuudet olivat kutsuja REST API:lle, josta haettiin sovelluksen tarvitsemia tietoja turnauksista, otteluista ja pelaajista.

```
index.js
leagueReducer.js
playerReducer.js
seasonReducer.js
tournamentReducer.js
userReducer.js
```

```
leagueActions.js
loginActions.js
playerActions.js
seasonActions.js
tournamentActions.js
userActions.js
```

Kuva 15. Redux-tilapuut ja niitä vastaavat toiminnot.

Käyttöliittymän ulkoasu luotiin käyttämällä erilaisia React komponentteja. React routerille kerrottiin minkä osoitteen tuottaa ja hallinnoi mikäkin React komponentti. Nämä sivuihin liitetyt komponentit ovat ns. fiksuja komponentteja, jotka ovat yhteydessä Redux tilapuu-

hun. Tätä tilaa sitten valutettiin tarvittaessa alikomponenteille. Tässä työssää komponenttihierarkia jäi suhteellisen pieneksi, joten ei ollut tarvetta liittää sivujen alikomponentteja suoraan tilapuuhun, vaikka Reduxilla se olisi ollut täysin mahdollista. Koodiesimerkki 5 esittelee komponentin, jossa määritetään React Routerille mitkä komponentit käsittelevät minkäkin reitin.

```
import React from 'react';
import { Route, Switch } from 'react-router-dom';
import LeagueList from '../components/LeagueList';
import LeagueDetails from './LeagueDetails';
import NoMatch from './NoMatch';

export default class Leagues extends React.Component {

  constructor(props) {
    super(props);
    this.state = {};
  }

  render() {
    return (
      <div>
        <Switch>
          <Route path="/leagues" exact component={League-
List} />
          <Route path="/leagues/:id" exact compo-
nent={LeagueDetails} />
          <Route component={NoMatch} />
        </Switch>
      </div>
    );
  }
}
```

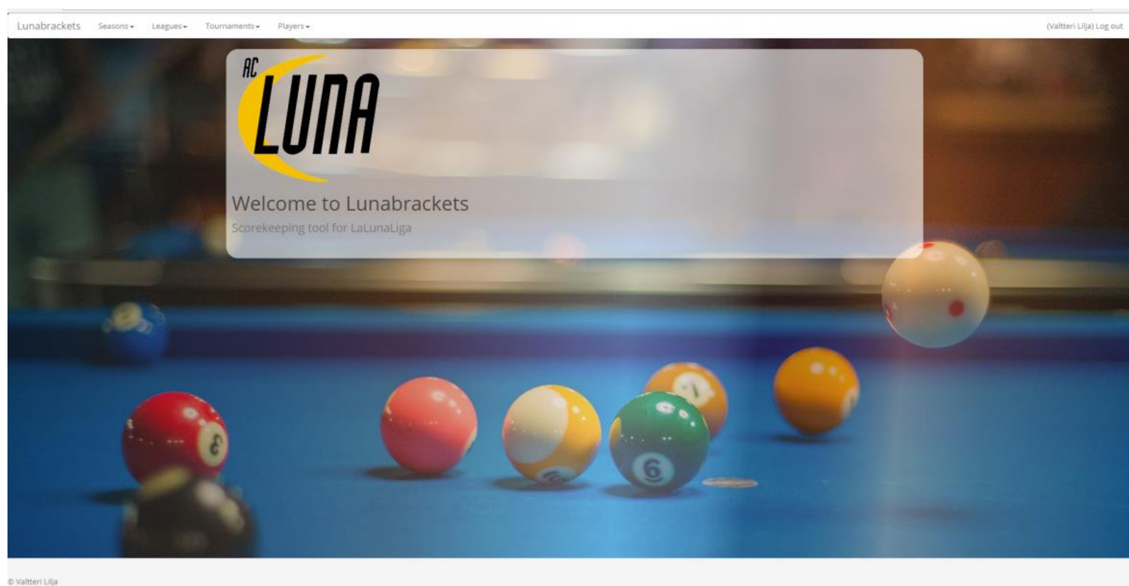
Esimerkkikoodi 5. Leagues osoitteen käsittelevä React-komponentti.

Kaikkien komponenttien sisäistä tilaa käytettiin pääasiassa erilaisten muotoilujen ja ulkoasun piirtämiseen ja muokkaamiseen. Komponenttien ja sivujen tyylin luomiseen käytettiin Bootstrappia. Bootstrap on valmis open source pohjainen verkkosivujen muotoiluun tehty työkalupaketti, jonka päätavoite on mobiiliskaalautuvuudessa [30]. Bootstrap valittiin juuri sen valmiin mobiiliskaalautuvuuden vuoksi, jotta aivan kaikkea ei tarvitsisi tehdä itse. Lievällä kustomoinnilla ja lisätyillä tyylisäännöillä sillä saatiin toteutettua toimiva käyttöliittymä niin normaalinäytössä kuin mobiilissa. Loppujen lopuksi sovellukseen tuli 12 sivua, joista 4 oli erilaisia lomakkeita ja loput sovelluksen esityssivuja. NoMatch on sivu, joka näytetään käyttäjän navigoidessa virheelliseen osoitteeseen, jota sovelluksessa ei ole.

```
Frontpage.jsx       Leagues.jsx  SeasonForm.jsx         TournamentForm.jsx
LeagueDetails.jsx   NoMatch.jsx  Seasons.jsx            Tournaments.jsx
LeagueForm.jsx      Players.jsx  TournamentDetails.jsx  UserForm.jsx
```

Kuva 16. Sovelluksen sivut.

Koko sovellus haluttiin laittaa kirjautumisen taakse ja pois näkyvistä rekisteröitymättömille käyttäjille. Tämän toteuttamisessa päädyttiin käyttämään Facebookin valmista kirjautumista. Facebook oli ilmiselvä vaihtoehto, koska kaikki pelaajat olivat tottuneet käyttämään Facebookia liigan tulosten ilmoittamiseen. Valmiin kirjautumislogiikan ja tunnistautumisen käyttäminen helpottaa myös huomattavasti kehittäjän taakkaa, koska kaikkea tätä logiikkaa ei tarvitse luoda itse. Myös lakisääteiset velvoitteet henkilötietojen tallentamisesta ja käytöstä eivät jää kehittäjän huoleksi. Lisäksi sovellukseen tuli luoda kaksi eri näkymää monille sivuille. Tämä oli helppo toteuttaa Reactilla, koska monissa tapauksissa sivulle voitiin vain lisätä admin komponentti, joka oli piilossa tavallisilta käyttäjiltä. Koska koko sovellus on kirjautumisen takana eikä hakukoneoptimoinnille ole minkäänlaista tarvetta, luotiin sovellus perinteisenä SPA-sovelluksena isomorfisen sovelluksen sijaan. Kuvassa 17 esitetään sovelluksen kirjautunut näkymä.



Kuva 17. Sovelluksen kirjautunut näkymä.

REST APIn luominen alkoi asentamalla Express sovelluskehys. API:n reitit määritettiin ja jaettiin eri tiedostoihin selkeyden vuoksi (kuva 18). Rajapinnan reittien jako muistutti hyvin pitkälti käyttöliittymän toimintojen jakoa. Index tiedostossa jokaiselle reitille annettiin päänreitti, jonka alireitiksi tiedoston reitit tulivat.

```
index.js          playerRoutes.js   tournamentRoutes.js
leagueRoutes.js   seasonRoutes.js   userRoutes.js
```

Kuva 18. REST API:n reittien jako.

Myös reittien (esimerkkikoodi 6) nimeämissä pyrittiin säilyttämään REST API:n hyvät nimeämiskäytännöt parhaan mukaan. Jokaiselle objektille luotiin oman käsittelijä, jota kutsuttiin sille ominaisen reitin kautta. Näin sovelluksen logiikka saatiin eristettyä rajapinnan hallinnasta.

```
const express = require('express');
const tournamentHandler = require('../handlers/tournamentHandler');

const router = express.Router();

router.get('/', (req, res) => {
```

```
    tournamentHandler.getAllTournaments((response) => {
      if (response instanceof Error) {
        res.status(400).send('Error fetching tournaments');
      } else {
        res.json(response);
      }
    });
  });
```

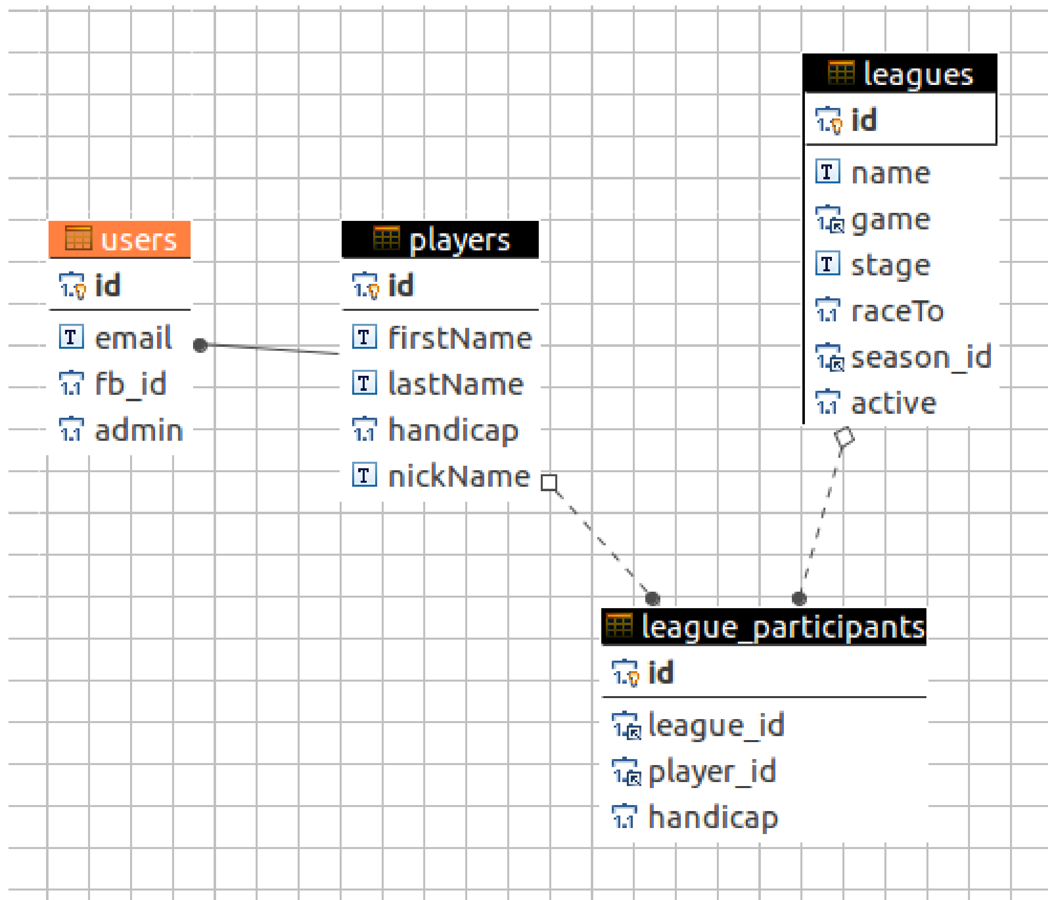
Esimerkkikoodi 6. Kaikkien turnausten hakemiseen käytetty reitti.

REST API:n taakse tuli luoda sovelluslogiikka, joka hoiti kaikki tietokantakutsut ja hallinnoi kaiken sovelluslogiikan. Tiedon tuli pysyä eheänä koko ajan ja kaiken on toimittava virheettä. Tähän tarkoitukseen luotiin käsittelijät, jotka hoitivat erilaisten objektien hallinnoimisen ja käsittelyn. Käsittelijöiden lisäksi sovelluksessa käytettiin muutamia apuluokkia esimerkiksi erilaiseen validointiin. Käsittelijöiden takana oli vielä lisäksi kaikki tietokantaoperaatiot omassa tiedostohierarkiassaan. Kaikki tietokantakutsut toteutettiin JavaScript lupauksina, joka helpotti synkronisen koodin kirjoittamista objektien käsittelyssä. MariaDB:n käyttämiseen JavaScriptillä löytyi suoraan valmis moduuli npm rekisteristä, jolla tietokantakutsut sai tehtyä. Esimerkkikoodissa 7 on esillä kaikki turnaukset tietokannasta hakeva operaatio.

```
    getAllTournaments(c) {
      return new Promise((resolve, reject) => {
        const queryString = 'SELECT * FROM tournaments';
        c.query(queryString, (error, rows) => {
          if (error) {
            reject(new  Error(`[ERROR]  getAllTournaments${er-
  ror}`));
          } else {
            resolve(rows);
          }
        });
      });
    },
```

Esimerkkikoodi 7. Turnaus-tietokantamodulin operaatio.

Sovelluksen tietokannasta tuli suhteellisen iso, mutta relaatiomalli (kuva 19) sopi täydellisesti turnausmallien esittämiseen. Esimerkiksi käyttäjät oli helppo eriyttää pelaajista, jos joku Facebookia ei käyttävä pelaaja haluaisi osallistua turnauksiin. Myös liigat ja ottelut sai mallinnettua relaatiokannalla täydellisesti. Kantaan syntyi kokonaisuudessaan 25 taulua. Tiedon suhteellisen vähäisyyden vuoksi ei kantaan ollut tarvetta rakentaa yhtään indeksiä, koska kaikki hakuoperaatiot toimivat nopeasti ja vaadittujen hakujen relaatiot pysyivät hierarkkisesti matalina.



Kuva 19. Tietokannan taulujen relaatiot käyttäjistä liigoihin.

Koska koko sovellus haluttiin pitää kirjautumisen takana, tuli rajapintakutsut rajata vain rekisteröidyille käyttäjille. Jokaisen kutsun tekemiseen vaaditaan Facebook sisäänkirjautumisen tunnus, jonka voimassaolo ja oikeellisuus tarkistetaan Facebookin rajapinnasta. Jos sovelluksen avointa käyttöä halutaan laajentaa tulevaisuudessa, voidaan vain tietyt operaatiot rajata tunnistautumisen piiriin tulevaisuudessa.

Toimiva sovellus päätettiin julkaista käyttäen AWS:n tarjoamia EC2 palvelimia. Tämä ratkaisu osoittautui erittäin halvaksi, koska AWS tarjoaa ilmaisen micro-kokoisen instanssin per kuukausi. Näin relaatiokanta voitiin asentaa tälle palvelimelle, eikä ollut tarvetta myöskään ostaa erillistä relaatiokantapalvelua AWS:ltä. Verkkosovelluksen julkaiseminen oli hyvin helppoa ja ainoa kustannus joka palvelusta koitui, oli verkko-osoitteen rekisteröinti.

5.1 Jatkokehitys

Tällä hetkellä sovellus toimii vain näille kahdelle hyvin spesifille formaatille (32 pelaajan liigaformaatti ja yksipäiväinen turnaus), mutta muiden turnausformaattien ja pelien lisääminen voi olla tulevaisuudessa kannattavaa. Lunassa myös pelataan joka viikko tästä liigasta erilliset viikkokilpailut, joiden rankingia päivitetään toiseen olemassa olevaan vanhempaan järjestelmään. Esimerkiksi viikkokilpailujen kaaviot ja pistelaskun voisi hoitaa hyvin helposti sähköisesti ja integroida ne osaksi Lunabracketsia. Lisäksi kaiken pelaajien otteludatan voisi valjastaa erilaiseen ottelun lopputulosten ennustamiseen ja muuhun tilastolliseen peliseurantaan.

Lunassa on myös käytössä erillinen Pool tool, joka on biljardiottelun pistelaskuun käytettävä ohjelma. Tämän ohjelman voisi integroida osaksi Lunabracketsia, josta pelaajat voisivat halutessaan lähettää tuloksen suoraan Lunabracketsiin.

Kolmantena jatkokehitysehdotuksena on mobiilikäyttöliittymän parantaminen. Vaikka bootstrap tarjoaakin valmiin mobiiliskaalautuvuuden, voisi mobiilikäyttöliittymää hioa huomattavasti paremmaksi. Tällä hetkellä se on kyllä toimiva, mutta käytettävyys ei ole huipussaan. Esimerkiksi jotkin elementit ja fontit näkyvät liian pieninä, jolloin käyttäjä joutuu manuaalisesti zoomaamaan näyttöä sen sijaan, että voisi vain suoraan käyttää sovellusta.

6 Yhteenveto ja pohdinta

Insinöörityön tavoitteena oli kehittää toimiva turnausjärjestelmä AC Lunalle korvamaan Lunaliiga biljardiliigan nykyinen kirjanpito, joka oli sekalainen kasa Facebook ryhmiä ja

Google Docs dokumentteja. Samalla oli tarkoitus perehtyä Fullstack JavaScript kehittämiseen ja verkkosovelluksen julkaisemiseen pilvipalvelussa. Työssä saavutettiin molemmat tavoitteet ja loppukäyttäjät ovat olleet tyytyväisiä järjestelmään.

Verkkosovelluksen kehittämisen taustoittamisen vuoksi käytiin läpi verkkosovelluksia yleisesti, niiden historiaa sekä muotoutumista yhdeksi nykypäivän tarkeimmistä sovellusmalleista. Työssä käytiin myös läpi kaikki verkkosovelluksen toteuttamiseen tarvittavat osat: käyttöliittymä, palvelintaso ja tietovarasto. Tietoturva haluttiin ottaa myös käsiteltäväksi sen ollessa nykyään erittäin olennainen osa verkkosovelluksen toteuttamista. Aiheen laajuuden vuoksi siihen ei kuitenkaan paneuduttu syvemmin vaan ideana oli enemmänkin muistuttaa sen olemassa olosta ja huomioon otettavuudesta lukijalle verkkosovellusten kehityksessä. Pilvipalvelut haluttiin myös ottaa osaksi raporttia, koska kehitettävä sovellus oli tarkoitus julkaista pilvialustalla.

Koska toteutettava järjestelmä oli tarkoitus luoda biljardille, haluttiin työssä käydä myös nopeasti läpi biljardi lajina ja myös erilaiset turnausmallit. Myös ehdolla olleet turnausjärjestelmät ja niiden soveltuvuus käsiteltiin. Turnausjärjestelmän ominaisuudet olivat kuitenkin niin spesifit, että valmista järjestelmää ei markkinoilta löytynyt.

Työn luonteen ja työn tekijän oman mielenkiinnon vuoksi työ päädyttiin tekemään Fullstack-JavaScript sovelluksena. Työn JavaScript suuntauksen vuoksi myös JavaScript kielenä haluttiin esitellä lukijalle. Sovelluksen teknologioiksi valikoitui React Reduxilla ja palvelin puolella Node ja Express. Tietokannaksi valittiin perinteisempi relaatiokanta MariaDB, koska relaatiomalli heijasti parhaiten turnausten tietomallia.

Työn toteuttamiseen kuitenkin kului huomattavan paljon enemmän aikaa kuin alussa oli suunniteltu. Alussa tämä johtui teknologioihin perehtymisestä ja lopussa kehittäjän laiskuudesta. Laiskuudella viitataan tässä tapauksessa testitapausten tekemättä jättämiseen heti työn alusta pitäen. Työ olisi pitänyt hahmotella paremmin alussa ja turnauksen spesifin luonteen vuoksi moni asia olisi ollut helpompaa, jos heti alusta pitäen olisi luonut testitapaukset koodille, vaikka edes kaikista haastavimmille operaatioille.

Modernissa JavaScript kehittämisessä kuitenkin on hyvät puolensa. Sillä on erittäin nopea saada aikaiseksi prototyyppejä ja sille löytyy lukemattomat määrät valmiita työkaluja käytettäväksi. JavaScript on todellakin noussut yhdeksi tärkeimmistä kieliestä kehittäjän

työkalupakissa. Käyttöliittymien kehittäminen on erittäin helppoa nykyaikaisten kirjastojen ja sovelluskehysten avulla. Myös RestAPI:n pystyttäminen Expressillä käy käden käänteessä.

Tätä kirjoitettaessa turnausjärjestelmä on ollut käytössä noin 3kk ajan ja palaute on ollut positiivista. Kehitysehdotuksia sovellukseen on tullut muutamia ja niitä pyritään toteuttamaan mahdollisuuksien mukaan. Kaiken kaikkiaan sovelluksen toteuttaminen oli haastavampaa kuin alussa kuvittelin ja työtä tehdessä oppi hyvin paljon. Lupaan myös itselleni, että en tee enää yhtäkään projektia ilman testitapauksia. Testitapauksien olemassa olon edut tulevat esiin siinä vaiheessa, kun projektin koko kasvaa, eikä kokonaisuutta pysty enää hahmottamaan ja muistamaan täysin.

Lähteet

1 Nations,Daniel. 2018. What exactly is a Web Application? Verkkodokumentti. Lifewire. 22.2.2018. <https://www.lifewire.com/what-is-a-web-application-3486637> Luettu 3.4.2018.

2 Emery, Colyn. 2016. A Brief History of Web Development. Verkkodokumentti. Techopedia. <https://www.echopedia.com/2/31579/networks/a-brief-history-of-web-development> Luettu 3.4.2018.

3 Facebook newsroom, company info. 2018. Verkkodokumentti. Facebook. <https://newsroom.fb.com/company-info/>. Luettu 3.4.2018.

4 Kohvakka, Rauli. 2014. Suomalaiset ovat Euroopan kärkeä sähköisessä asioinnissa. Verkkodokumentti. Tilastokeskus. <https://www.stat.fi/artikkelit/2014/art_2014-09-29_007.html>. Luettu 3.4.2018.

5 Najim, Maxime & Strimpel, Jason. 2016. Building Isomorphic JavaScript Apps: Chapter 1. Why Isomorphic JavaScript? Verkkodokumentti. <https://www.safaribooksonline.com/library/view/building-isomorphic-javascript/9781491932926/ch01.html>. Luettu 10.4.2018.

6 HTML & CSS. 2018. Verkkodokumentti. W3C. <https://www.w3.org/standards/webdesign/htmlcss>. Luettu 10.4.2018.

7 JavaScript. 2016. Verkkodokumentti. Mozilla Developers Network. <https://developer.mozilla.org/bm/docs/Web/JavaScript>. Luettu 10.4.2018.

8 Crockford, Douglas. 2008. JavaScript: The Good Parts. O'Reilly Media, Inc. Verkkodokumentti. <https://www.safaribooksonline.com/library/view/javascript-the-good/9780596517748/ch01.html>. Luettu 10.4.2018

9 Michael, Bradely. 2018. Server Are The Heart and Lungs of the Internet. Verkkodokumentti. Lifewire. <https://www.lifewire.com/servers-in-computer-networking-817380>. Luettu 11.4.2018.

10 What is a web server?. 2018. Verkkodokumentti. Mozilla Developers Network. <https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server>. Luettu 11.4.2018.

11 Processing dynamic pages. 2018. Verkkodokumentti. eTutorials.org. <http://etutorials.org/Macromedia/Dream+Weaver+Online+Help/Getting+Started+with+Dreamweaver/Understanding+Web+Applications/How+a+web+application+works/Processing+dynamic+pages/>. Luettu 11.4.2018.

12 Database. 2009. Verkkodokumentti. TechTerms. <https://techterms.com/definition/database>. Luettu 12.4.2018.

13 What is NoSQL?. 2018. Verkkodokumentti. mongoDB. <https://www.mongodb.com/nosql-explained>. Luettu 12.4.2018.

14 Foote, Keith. 2016. A Review of Different Database Types: Relational versus Non-Relational. Verkkodokumentti. <http://www.dataversity.net/review-pros-cons-different-databases-relational-versus-non-relational/>. Luettu 12.4.2018.

15 Ajax. 2018. Verkkodokumentti. Mozilla Developers Network. <https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX>. Luettu 12.4.2018.

16 XMLHttpRequest. 2018. Verkkodokumentti. Mozilla Developers Network. <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>. Luettu 12.4.2018

17 Meier, J.D & Mackman, Alex & Dunner, Michael & Vasireddy, Srinath & Escamilla, Ray & Murukan, Anandha. Improving Web Application Security: Threats and CountermeasuresWeb Application Security Fundamentals. 2003. Verkkodokumentti. Microsoft Corporation. <https://msdn.microsoft.com/en-us/library/ff648636.aspx>. Luettu 26.4.2018.

18 Welcome to OWASP. 2018. Verkkodokumentti. OWASP <https://www.owasp.org/index.php/Main_Page>. Luettu 26.4.2018.

19 What is Cloud Computing?. 2018. Verkkodokumentti. Amazon Web Services. <https://aws.amazon.com/what-is-cloud-computing/>. Luettu 30.4.2018.

20 Types of Cloud Computing. 2018. Verkkodokumentti. Amazon Web Services. <https://aws.amazon.com/types-of-cloud-computing/>. Luettu 30.4.2018.

21 What is cloud computing stack?. 2017. Verkkodokumentti. Oracle-Help. <http://oracle-help.com/oracle-cloud/cloud-computing-stack-saas-paas-iaas/>. Luettu 2.5.2018.

22 Application framework. 2018. Verkkodokumentti. Tehcopedia. <https://www.techopedia.com/definition/6005/application-framework>. Luettu 2.5.2018.

23 React. 2018. Verkkodokumentti. Facebook Inc.<https://reactjs.org/>. Luettu 3.5.2018.

24 Wheeler, Ken. 2016. Getting to know Flux. Scotch. Verkkodokumentti. <https://scotch.io/tutorials/getting-to-know-flux-the-react-js-architecture>. Luettu 9.5.2018.

25 Lillie, Dakota.2017. Flux vs Redux: A comparison. Medium. Verkkodokumentti. <https://medium.com/@dakota.lillie/flux-vs-redux-a-comparison-bbd5000d5111>. Luettu 9.5.2018.

26 Young, Alex ym. 2017. Node.js in Action, Second edition. Chapter 1. Welcome to Node.Js. Manning Publications Co.

27 Young, Alex ym. 2017. Node.js in Action, Second edition. Chapter 6. Connect and Express in depth. Manning Publications Co.

28 About MariaDB. 2018. MariaDB. Verkkodokumentti. <https://mariadb.com/about-us>. Luettu 18.5.2018.

29 Deering, Sam. 2012. Do you know what a REST API is?. SitePoint. Verkkodokumentti. <https://www.sitepoint.com/developers-rest-api/>. Luettu 6.6.2018.

30 Bootstrap. 2018. Verkkodokumentti. <https://getbootstrap.com/>. Luettu 6.6.2018.

31 Challonge. 2018. SplitmediaLabs Limited. Verkkodokumentti. <www.challonge.com>. Luettu. 8.6.2018.

32 CueScore. 2018. CueScore International. Verkkodokumentti. <https://cuescore.com>. Luettu 8.6.2018.2