Marko Hamppula

Modernit web-teknologiat pelikehityksessä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Opinnäytetyö

03.12.2015

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Marko Hamppula Modernit web-teknologiat pelikehityksessä 33 sivua 03.12.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Simo Silander, Lehtori Miikka Mäki-Uuro, Lehtori

Tämän opinnäytetyön tavoitteina on tutustua, kuinka modernit web-teknologiat soveltuvat selainpohjaiseen pelikehitykseen. Samalla tutustutaan, miten perinteiset teknologiat ovat kehittyneet ja miten pelit ovat kommunikoineet palvelimen kanssa. Lisäksi tutustutaan, mitä teknologioita on tullut lisää perinteisesti käytössä olevien teknologioiden rinnalle.

Työssä suunniteltiin prototyyppi reaaliaikaiselle selainpohjaiselle pelille, jossa kaikki tärkeät laskentaan liittyvät toimenpiteet suoritetaan palvelinarkkitehtuurin sisällä. Prototyypin teknologioihin valitsin kirjoitushetkellä trendikkäitä web-teknologioita, joiden avulla tutustuttiin, miten kyseiset web-teknologiat soveltuvat selainpohjaisenpelin toteuttamiseen.

Työn tulokseksi saatiin toteutettua toimiva prototyyppi, jonka avulla tutustuttiin, kuinka valitut palvelin- ja selainteknologiat soveltuvat selainpohjaisen pelin toteutukseen. Prototyypin avulla tutkittiin, kuinka palvelinohjelmistot tulivat kommunikoimaan keskenään.

Avainsanat	Node.js, React.js, RefluxJS, WebGL, Three.js, MongoDB

Author(s) Title Number of Pages Date	Marko Hamppula Modern Web Technologies in Game Development 33 pages 3 December, 2015
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Simo Silander, Senior Lecturer Miikka Mäki-Uuro, Senior Lecturer

The goal of this Bachelor's Thesis was to get familiar with how modern web technologies can be used in game development. This Bachelor's Thesis also focuses on how web technologies have developed in the past and how games have communicated with the server. Furthermore, the study introduces the new web technologies that have appeared next to the ones having existed for a relatively long time.

In the thesis a prototype was designed for a real time web based game. All the calculations for the game were done at the server side. To implement the prototype some of the trending technologies were chosen and during the prototype implementation the technologies were evaluated as to how well they fit for web based games.

The result of this Bachelor's Thesis was a proof of concept, which demonstrated how well the chosen server and client side technologies suited for web based game development. The proof of concept was used to get familiar with how different server side applications communicated between each other.

Keywords	Node.js, React.js, RefluxJS, WebGL, Three.js, MongoDB

Sisällys

Lyhenteet

ASP	Active Server Pages, Microsoftin kehittämä web-käyttöliittymäteknologia.
JSP	JavaServer Pages, Java EE -standardin mukainen web-käyttöliittymäteknologia.
PHP	PHP Hypertext Preprocessor, palvelinympäristöissä dynaamisten verkkosivujen luonnissa käytetty ohjelmointikieli.
TL;DR	Too long, didn’t read

1 Johdanto

Selainpohjaiset pelit ovat pitkään pohjautuneet kolmannen osapuolen tarjoamiin teknologioihin, joita ovat esimerkiksi Adoben Flash ja Shockwave. Teknologioiden ongelmana on ollut, ettei käyttäjällä välttämättä ole uusinta versiota tarvittavasta teknologiasta, mikä voi johtaa esimerkiksi tietoturvaongelmiin. Teknologioiden ja päätelaitteiden kehittyessä uudet selainteknologiat ovat tulleet saataville, minkä ansiosta ei olla riippuvaisia kolmansien osapuolten teknologioista. Uusien teknologioiden ansiosta käyttäjän ei tarvitse asentaa kolmannen osapuolen ohjelmistoa pelatakseen peliä.

Käyttäjämäärät ovat lisääntyneet ja internetsivustot muuttuneet koko ajan monimutkaisimmiksi, mikä on luonut kysyntää uusille web-teknologioille. Suuret organisaatiot ovat alkaneet panostamaan uusiin selainteknologioihin. Esimerkiksi Facebook on yksi näistä organisaatioista, jotka ovat lähteneet toteuttamaan omia teknologioita. Viimeaikoina Facebook on esitellyt toteuttamiaan teknologioita ja avannut niitä myös muiden käyttöön.

Opinnäytetyössä käydään läpi, mitä modernit web-teknologiat ovat ja millaista hyötyä niillä voidaan saavuttaa. Lisäksi käsitellään, kuinka selainpohjaiset pelit ovat kehittyneet, sekä suunnitellaan prototyyppi reaaliaikaiselle selainpelille.

2 Modernit web-teknologiat

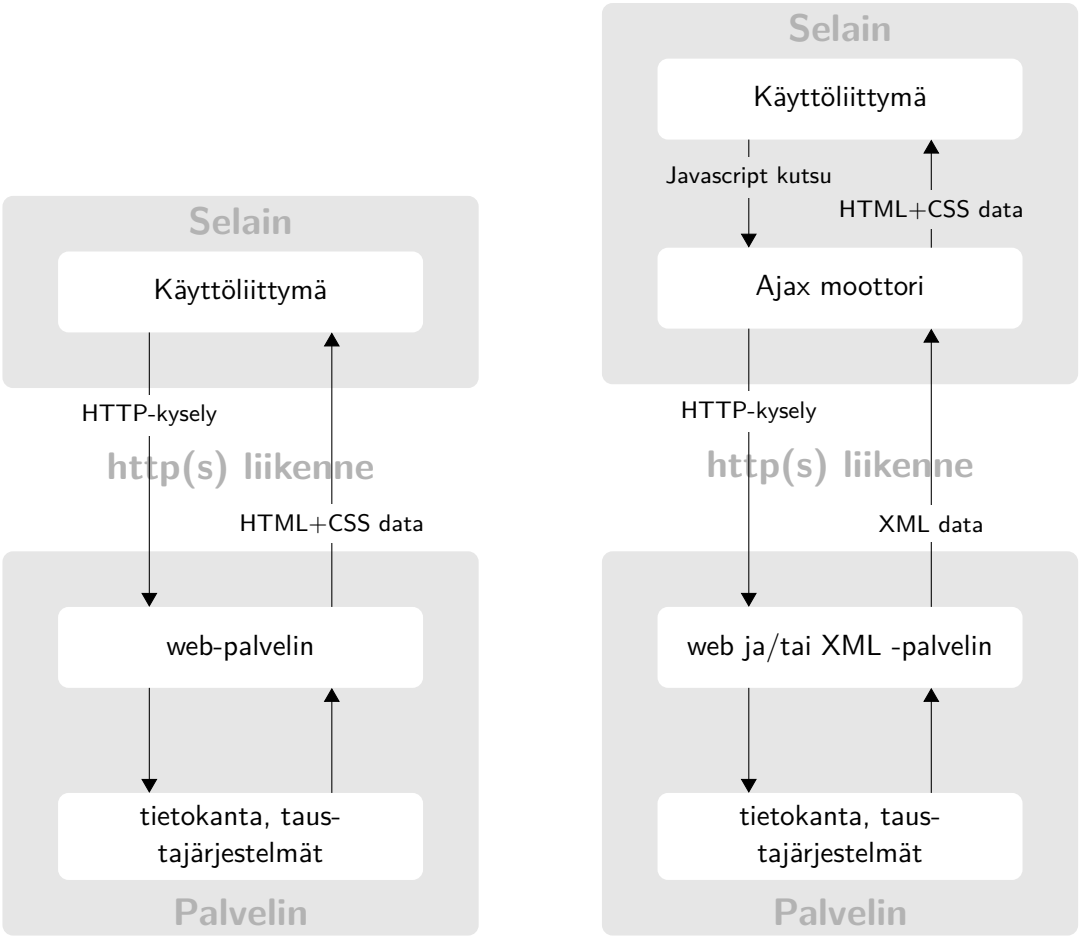
2.1 Palvelinteknologiat

Aikaisemmin verkkosivut olivat pitkälti staattisia sivuja, joiden päivitys vaati HTML-kuvauskielen ymmärtämistä. WWW-palvelimien kehittyessä kehitettiin teknologioita, joiden avulla saatiin toteutettua dynaamisia sivustoja. Näitä teknologioita olivat muun muassa PHP-,

JSP- ja ASP-ohjelmointikielet. Uudemmat teknologiat mahdollistivat tiedon dynaamisen haun esimerkiksi tiedostosta tai tietokannasta. Dynaamisen tiedon perusteella luodaan HTML-dokumentti, joka palautetaan selaimelle. Käyttäjän vaihtaessa sivua kyseinen operaatio toistetaan aina. Teknologioiden kehittyessä on kehittynyt myös palvelinpuolen teknologiat, joista tämän hetken uusimpiin teknologioihin kuuluu Node.js-sovelluskehys. Node.js mahdollistaa asynkronisen palvelinpuolen JavaScript-toteutuksen. [1, s. 15-17.]

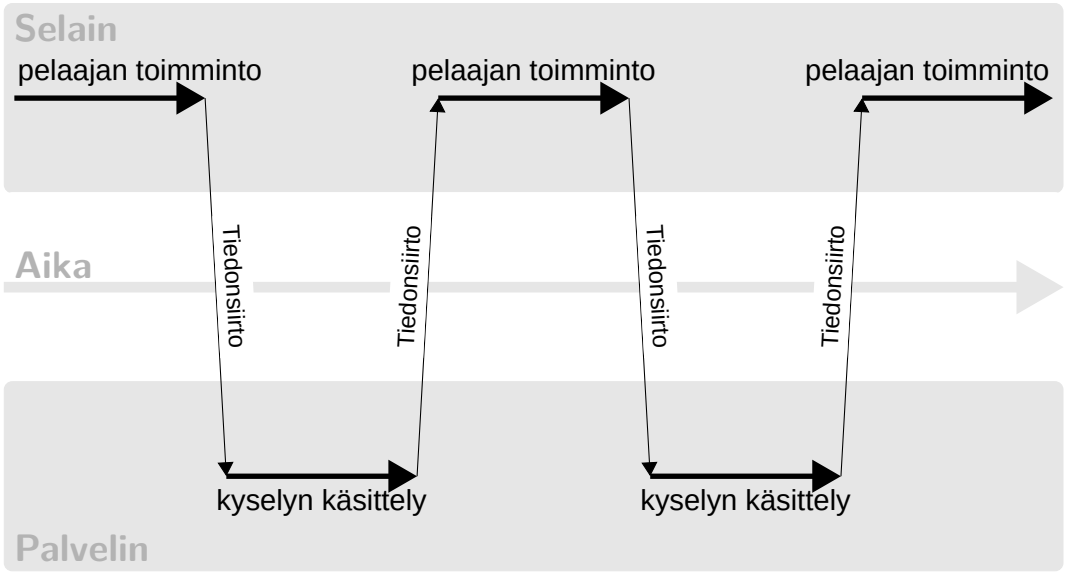
2.2 Selainpohjaiset teknologiat

Päätelaitteiden ja selainten kehittyessä haluttiin lisätä sivustojen käytettävyyttä, jolloin alettiin kehittämään siihen soveltuvia teknologioita. JavaScript on yksi ohjelmointikieli, joka syntyi tämän ajatusmallin pohjalta. JavaScriptin avulla voidaan esimerkiksi muokata sivuston yksittäisiä osia ilman, että koko sivustoa täytyisi ladata uudelleen. Teknologioiden kehittyessä selaimet alkoivat saamaan tuen Ajax-tekniikalle eli Asyncronous JavaScript And XML:lle. Teknologia mahdollistaa tiedon hakemiseen tai lähettämisen palvelimelle JavaScriptin avulla. [2.]



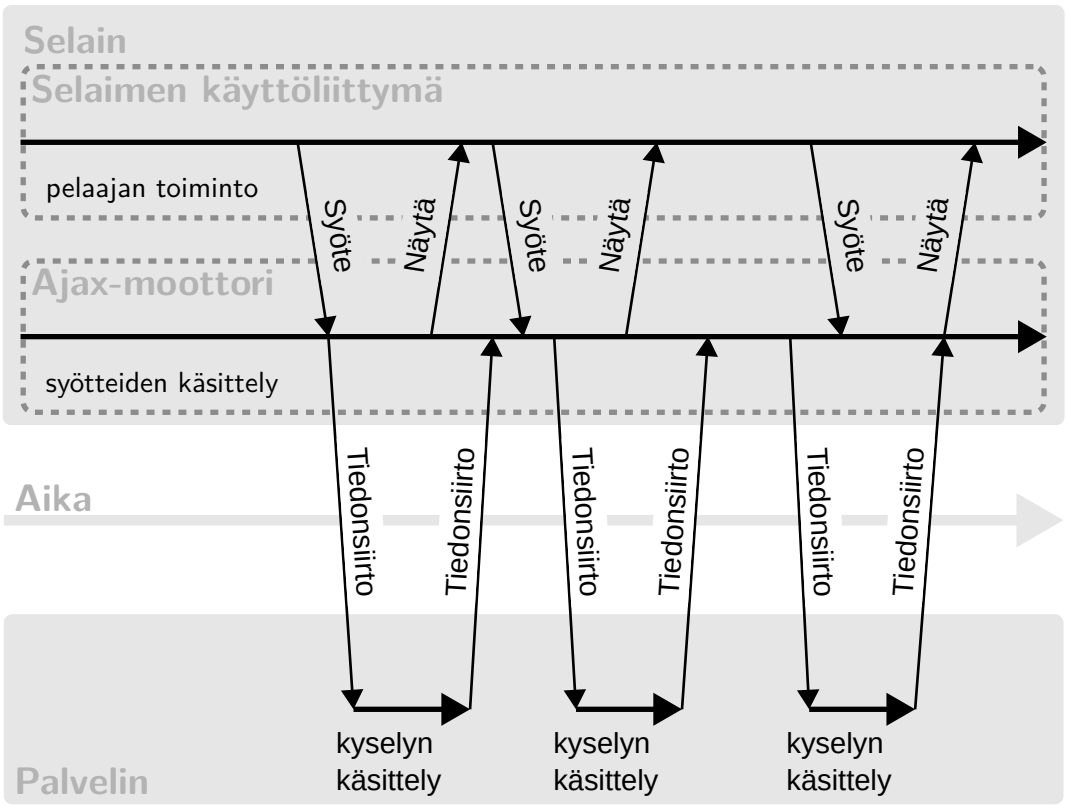
Kuva 1: Web-ohjelmointimallit [3]

Kuvassa 1 vasemmalla on perinteinen web-ohjelmiston malli ja oikealla on Ajaxi-tekniikalla toteutettu web-ohjelmiston malli. Ajax:n tullessa tiedon välittämiseen käytettiin XML-dokumenttirakennetta. Selainten JavaScript tuen kehittyessä eri tahot alkoivat kehittää JavaScript-ohjelmistokehyksiä, jotka mahdollistavat Single Page Application (SPA) -kehittämisen. SPA-ohjelmalla käsitetään sovelluksen toimintatapaa, jossa ensimmäisellä sivunlatauksella ladataan koko sovelluksen toiminnallisuus, jonka jälkeen dynaamista tietoa haetaan Ajax-tekniikan avulla. SPA-ohjelmistokehyksiä ovat muun muassa Ember, Backbone.js sekä AngularJS. Kyseiset ohjelmistokehykset tarjoilevat palvelimelta HTML-sivuston rungon, jota myöhemmin muokataan dynaamisesti JavaScriptin avulla. JavaScriptin avulla voidaan dynaamisesti hakea palvelimelta lisää sisältöä.



Kuva 2: Synkroninen web-ohjelman malli [3]

Kuvassa 2 esitetään, miten perinteisen web-ohjelman suoritus etenee pelaajan edetessä sivustolla. Pelaajan vaihtaessa sivua pyydetään palvelimelta kyseinen sivu.



Kuva 3: Asynkronisen web-ohjelman malli [3]

Kuvassa 3 esitetään, miten Ajax-tekniikka toteutettu sivusto etenee pelaajan tapahtumien perusteella.

Asynkroonisuuden avulla yritetään välttää käyttöliittymän lukkiutumista ja lisätä sivuston käyttömukavuutta. Asynkroonisuus mahdollistaa tiedon haun palvelimelta ilman käyttäjän huomaamista, jolloin sivuston käyttö tuntuu lähes samalta, kuin käytettäisiin normaalia tietokoneohjelmistoa. [4, s. 151-152.]

Ajax-tekniikan tullessa käytettiin XML-dokumenttirakennetta tiedon välittämiseen palvelimelta selaimelle, mutta nykyään JavaScript Object Notation (JSON) -dokumenttirakenne on korvaamassa laajalti käytettyä XML-rakennetta. JSON-dokumentin hyötyjä on mm. dokumentin muuntaminen JavaScript-objektiksi, ja se on helppolukuisempaa. JSON-dokumentti rakenne perustuu avain-arvopareihin, joiden arvona voi olla numero, teksti, totuusarvo, lista tai objekti.

```
1 {
2   "nimi": "Pertti",
3   "kaverit": [
4     {"nimi": "Antti", "pelissa": true},
5     {"nimi": "Pekka", "pelissa": false}
6   ],
7   "sijoitus": 25
8 }
```

Koodiesimerkki 1: JSON-dokumentti

Koodiesimerkissä 1 esitetään yksinkertainen JSON-dokumentti, jolla on nimi, kaverit ja sijoitus. Jokaisella kaverilla on nimi ja tieto siitä, onko hän pelissä.

2.3 Tietokannat

Tietokannat ovat olleet pitkään relaatiotietokantoja, jotka perustuvat valmiiksi suunniteltuihin tauluihin ja taulujen välisiin suhteisiin. Tarpeiden muuttuessa ollaan huomattu, ettei relaatiotietokannat pysty vastaamaan tarpeeseen, joita sovelluksilla on. Relaatiotietokannoille on alettu etsimään vaihtoehtoisia tietokantoja, joita kutsutaan yleisemmin termillä not only SQL (noSQL) -tietokannat, joissa tietojen välillä ei ole suhteita. NoSQL-

tietokantoja on olemassa muutamia eri tyyppisiä, esimerkiksi dokumenttitietokanta, avain-arvokantoja, leveitä sarakepohjaisia kantoja sekä graafikantoja. Jokainen näistä eri tietokannoista soveltuu erilaisiin käyttötarkoituksiin. Dokumenttitietokannoista MongoDB on hyvin suosittu sen helppokäyttöisyyden ja natiivin JSON-muodon tuen takia. Redis on avain-arvokanta, joka soveltuu hyvin väliaikaisten tietojen ylläpitoon. Redis pitää tiedon välimuistissa, josta tieto voidaan löytää nopeasti.

3 Selainpohjaisten pelien teknologiat

Selainpohjaisissa peleissä on pitkään käytetty Flash-, Shockwave-, Java applet- ja muita tekniikoita. Nämä tekniikat vaativat kolmannen osapuolen ohjelmiston asentamisen. Heikkona puolena on, etteivät tekniikat välttämättä päivity tarpeeksi usein, ja tämä voi johtaa esimerkiksi tietoturvaongelmiin. HTML5-standardi muuttaa tilannetta. Voidaan esimerkiksi piirtää 3D-grafiikkaa JavaScriptin avulla, eikä pelaajan tarvitse asentaa kolmannen osapuolen ohjelmistoa. Pelaajan vastuulle jää huolehtia, että web-selain on päivitetty uusimpaan versioon.

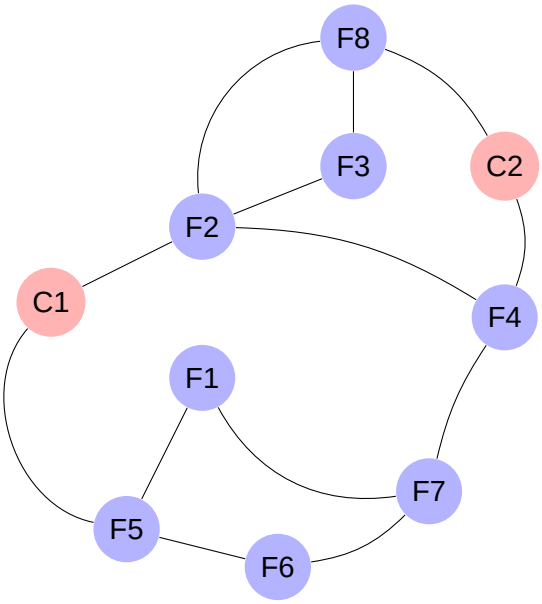
Päätelaitteiden kehittyessä selainpohjaiset pelit kehittyvät yhä visuaalisesti näyttävämmiksi ja voidaan tehdä aikaisempaa enemmän laskentaan liittyviä toimenpiteitä. Uusien teknologioiden avulla päästään eroon päätelaiteriippuvuuksista, eikä kehittäjän tarvitse tehdä jokaiselle eri alustalle samaa peliä moneen kertaan. Modernien teknologioiden avulla voidaan siis toteuttaa selaimessa toimiva peli, mutta se voidaan myös muuttaa natiiviksi mobiiliohjelmaksi.

Perinteisesti pelit ovat tehneet HTTP-kyselyitä Ajax-tekniikan avulla, mutta ongelmana on, ettei palvelin voi lähettää päätelaitteelle tietoa suoraan. Jotta palvelin voi lähettää viestin, on päätelaitteen pitänyt tehdä tietyn ajan välein HTTP-kyselyitä, jotka tarkistavat, onko uutta tietoa tulossa käyttäjälle. Toinen tapa on, että päätelaite ottaa yhteyttä palvelimeen, eikä palvelin sulje yhteyttä ennen kuin sillä on jotakin uutta tietoa lähetettäväksi päätelaitteelle. Tämän jälkeen päätelaite avaa uuden yhteyden, ja tätä toistetaan niin kauan, kunnes pelaaja pelaa peliä. HTML5-standardin tullessa julkaistiin WebSocket, joka mahdol-

listaa kaksisuuntaisen kommunikoinnin palvelimen ja päätelaitteen välille. Tämän avulla palvelin voi lähettää koska tahansa viestin käyttäjälle, eikä päätelaitteelle ja palvelimelle tarvitse toteuttaa monimutkaisia tarkistuksia siitä, onko uutta tietoa saatavilla.

4 Prototyypin suunnittelu

Tavoitteena on toteuttaa monen pelaajan reaaliaikainen strategiapeli. Pelin tavoitteena on kuljettaa tuotteita eri tehtaista kaupunkiin mahdollisimman paljon. Kaupungit ja tehtaat ovat verkkomaisesti kytkettynä toisiinsa. Jokaisella kaupungilla on kerrallaan neljä eri vaadittua tuotetta, joita pelaajien on kuljetettava, jotta kaupunki kasvaa. Kaupungin taso määrittää sen suuruuden. Kaupunkien saavuttaessa tason yksi, yksi vaadituista tuotteista vaihtuu uuteen tuotteeseen, sekä vaadittujen tuotteiden tuomismäärä kasvaa.



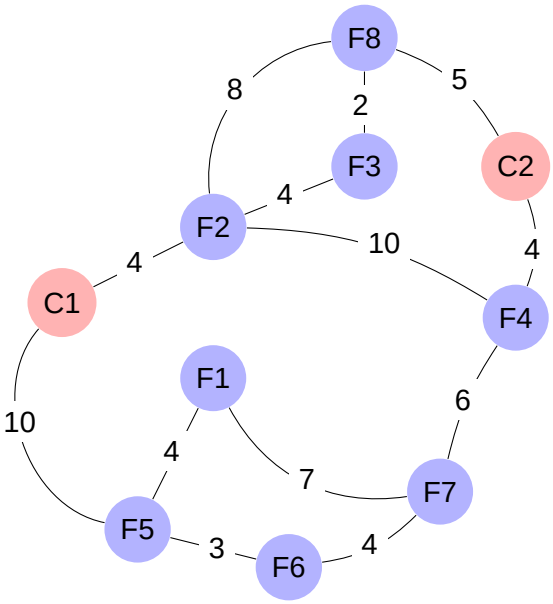
Kuva 4: Verkkomainen kartta

Kuva 4 esittää verkkomuotoista karttaa, jossa punaiset pallot kuvaavat kaupunkeja (Cn) ja siniset pallot tehtaita (Fn). Pelaajan kulkuneuvo voi kulkea verkon reittejä pitkin minkä tahansa tehtaan tai kaupungin lävitse. Jos pelaajan kulkuneuvo pysähtyy tehtaaseen lastamaan tehtaassa valmistettua tuotetta, niin silloin tehtaan odotusaika astuu voimaan, ja pelaajan kulkuneuvo joutuu odottamaan. Pelaaja siis voi kulkea tehtaan ohitse ilman, että siihen määritelty odotusaika astuu voimaan.

Kun pelaaja määrittelee kulkuneuvonsa kulkemaan esimerkiksi tehtaan F5 ja kaupungin C2 välistä reittiä, pelin on etsittävä lyhin mahdollinen reitti. Karttaa katsomalla huomataan, että on neljä erilaista reittiä, joissa kuljetaan yhtä monen tehtaan tai kaupungin kautta, mutta jokaisen tehtaan ja kaupungin välinen reitti on eri pituinen.

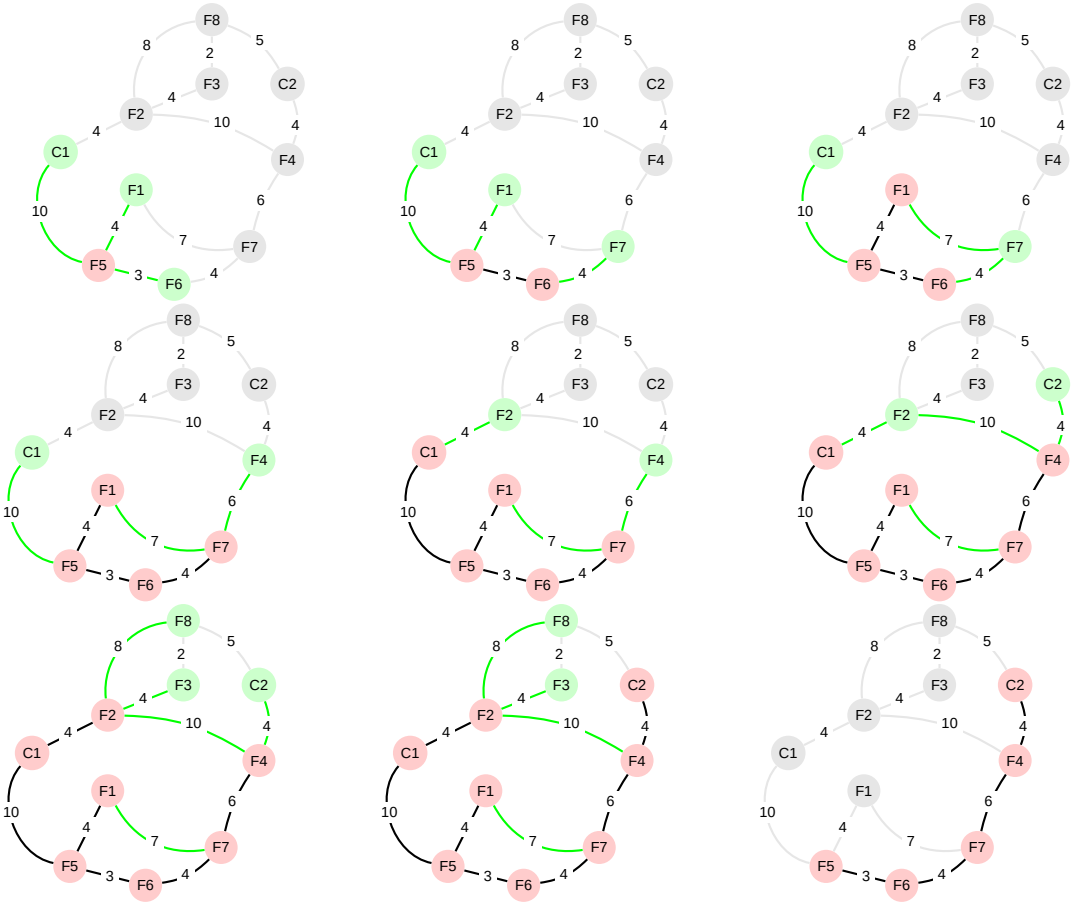
4.1 Lyhimmän reitin etsiminen

Lyhimmän reitin etsintä ei ole uusi matemaattinen ongelma. Tähän ratkaisun on löytänyt mm. Edsger Dijkstra. Edsger Dijkstra julkaisi vuonna 1959 Dijkstran algoritmin, joka on yksi käytetyimmistä tavoista selvittää verkon lyhin reitti. Dijkstran algoritmin tavoitteena on etsiä yhdestä pisteestä mahdollisimman lyhyt reitti. Reitillä tarkoitetaan alku- ja loppupisteen välistä matkaa, esimerkiksi pisteet F5 ja C2 välinen matka on reitti. Polulla tarkoitetaan kahden eri pisteen välistä matkaa, joiden välissä ei saa olla pisteitä, esimerkiksi pisteiden F5 ja F6 välinen matka on polku. Algoritmin vaatimuksena on, että jokaisella polulla on painoarvo, joka ei saa olla negatiivinen. Painoarvon lisäksi vaatimuksena on, ettei samaa pistettä käydä uudelleen lävitse. Painoarvoksi voidaan esimerkiksi valita polun pituus, kuten myös tässäkin tilanteessa. [5, s. 73-76.]



Kuva 5: Painotettu kartta

Kun kartta on painotettu kuten kuvassa 5, voidaan tämän jälkeen lähteä etsimään lyhyintä reittiä.



Kuva 6: Lyhimmän reitin haun vaiheet (F5 -> C2)

Kuvassa 6 nähdään reitin haku pisteestä F5 pisteeseen C2 vaihe vaiheelta. Kun lyhimmän reitin etsimistä aloitetaan, määritellään aloituspiste suljetuksi, ja siitä pisteestä lähtevien polkujen päässä olevat pisteet avoimiksi. Käydään avoimia pisteitä lävitse ja valitaan sieltä piste, joka on lähimpänä aloituspistettä. Valittu piste määritellään suljetuksi, jotta sitä ei käydä enää uudelleen lävitse, sekä valitusta pisteestä lähtevien polkujen päässä olevat pisteet lisätään avoimiksi. Tätä toimenpidettä jatketaan, kunnes päästään haluttuun pisteeseen. [5, s.73-76.]

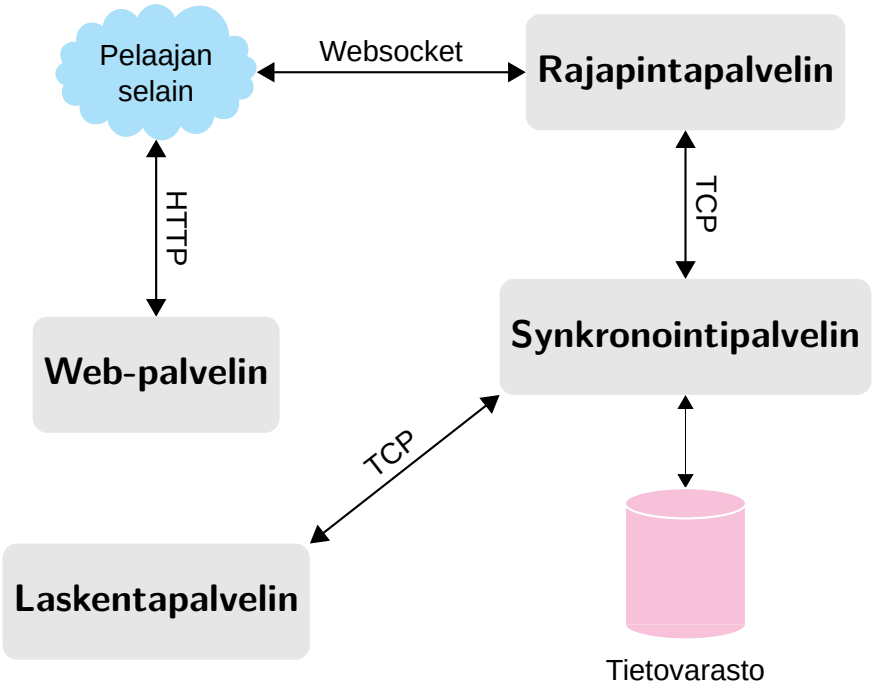
4.2 Arkkitehtuuri

Jotta peliä voitaisiin pelata, tarvitaan pelille arkkitehtuuri, joka vastaa pelin toiminnasta. Palvelimen on tarjottava pelaajalle käyttöliittymä, jossa hän voi pelata peliä sekä mistä pe-

laaja saa lisätietoja peliin liittyvistä tiedoista, kuten säännöt ja mitä pelissä kuuluu tehdä. Pelin reaaliaikaisuudella tarkoitetaan, että pelaajien kulkuneuvot kulkevat, vaikka pelaajilla ei oli missään päätelaitteessa peliä auki. Palvelimen on laskettava kaikki mahdolliset kulkuneuvojen liikkeet pelaajan määräämän reitin välillä. Jotta peliä voitaisiin pelaa, niin palvelimen ja pelaajan päätelaitteen välille tarvitaan yhteys, jossa välitetään kaikki peliin liittyvät tiedot.

Monet suuret pelialan yritykset eivät mielellään jaa tietoa peliensä arkkitehtuureista, mutta internetistä löytyy jonkin verran erilaisia sovelluksia. Lähdin tutkimaan, miten voisin hyödyntää erilaisia ratkaisuja omassa prototyypissäni, jotta pelaajia voitaisiin palvella mahdollisimman helposti. Päädyin ratkaisuun, jossa eri ominaisuudet on eroteltu toisistaan, mikä mahdollistaa tiettyjen komponenttien helpon skaalauksen, kun käyttäjien määrä kasvaa. Arkkitehtuuri koostuu neljästä eri palvelinohjelmistosta, jotka ovat:

1. Web-palvelin, jonka tehtävänä on tarjota pelaajalle käyttöliittymä peliin.
2. Laskentapalvelin, jonka tehtävänä on hoitaa kaikki peliin liittyvät laskentaprosessit
3. Rajapintapalvelin, jonka tehtävänä on taata kahdensuuntainen reaaliaikainen kommunikointi pelaajan selaimen kanssa.
4. Synkronointipalvelin, jonka tehtävänä on tarjota rajapintapalvelimelle pelin tietoa ja päivittää tietokantaan muuttunutta tietoa, jota laskentapalvelin lähettää.



Kuva 7: Palvelimien välinen kommunikointi

Kuvassa 7 esitetään palvelinarkkitehtuuri ja se, miten eri palvelimet kommunikoivat keskenään. Päädyin käyttämään eri palvelinohjelmistojen välillä TCP-yhteyttä, jonka avulla voidaan siirtää tietoa kahden eri fyysisen palvelimen välillä. TCP-yhteys mahdollistaa kahdensuuntaisen kommunikoinnin. Pelaajan päätelaitteen ja rajapintapalvelimen välillä käytetään WebSocket-yhteyttä, joka takaa kahdensuuntaiseen kommunikoinnin reaaliajassa.

4.2.1 Web-palvelin

Jotta pelaaja voi pelata peliä, tarvitaan palvelin, joka tarjoaa käyttöliittymän pelille. Web-palvelimen tehtävänä on tarjota pelaajalle sivustoa, jossa kerrotaan peliin liittyvät tiedot. Palvelin tarjoaa pelaajalle mahdollisuuden luoda henkilökohtaisen käyttäjätunnuksen tai vaihtoehtoisesti pelaaja voi kirjautua eri sosiaalisen median palveluilla. Kirjautuneelle pelaajalle tarjotaan palvelinlista, josta pelaaja voi valita, minkä eri alueen palvelimelle hän haluaa liittyä. Kun pelaaja on valinnut palvelimen, mihin hän haluaa liittyä, hänet ohjataan sivulle, jossa pelaaja näkee pelin kulun.

4.2.2 Rajapintapalvelin

Jotta pelaaja saa tiedon pelin kulusta, tarvitaan yhteys palvelimelle, josta voidaan kysyä tarvittavia tietoja. Lisäksi pelaajalla pitää olla mahdollisuus esimerkiksi vaihtaa kulkuneuvojen reittejä. Tätä toiminnallisuutta varten tarvitaan rajapintapalvelin, jonka tehtävänä on tarjota pelille rajapinta, josta käyttöliittymä voi kysyä tai lähettää pelaajan pyytämiä tietoja. Palvelimen tehtävänä on myös tarkistaa, onko yhteys tullut käyttöliittymältä vai jostakin muusta lähteestä, millä voidaan estää peliin liittyvää huijausta. Lisäksi palvelimen on tarkistettava kaikki syötteet, joita pelaaja tai pelinäkymä lähettää.

4.2.3 Synkronointipalvelin

Jotta tietokantaa ei kuormittuisi liikaa, tarvitaan ohjelmisto, jonka tehtävänä on tarjota peliin liittyvää tietoa sekä päivittää tietokantaan muuttuneet tiedot. Synkronointipalvelimen tehtävänä on tarjota TCP-yhteys, johon laskentapalvelin tai rajapintapalvelimet voivat ottaa yhteyttä. TCP-yhteyden avulla laskentapalvelin saa uusimman tiedon pelaajien kulkuneuvoista, sekä rajapintapalvelimet voivat kysyä tämän hetkistä tietoa pelaajien kulkuneuvoista tai vaihtoehtoisesti voidaan päivittää kulkuneuvojen reittejä.

4.2.4 Laskentapalvelin

Pelin pelaamiseksi tarvitaan ohjelmisto, jonka tehtävänä on huolehtia kaikesta mahdollisesta laskennallisesta toiminnasta. Laskentapalvelimen tehtävänä on hoitaa kaikki laskenta sekä toimittaa muuttuneet tiedot synkronointipalvelimelle. Erilaisia laskentaan liittyviä toimintoja ovat kulkuneuvojen sijaintien päivittäminen, tehtaiden ja kaupunkien ominaisuuksian päivittäminen sekä muut lisättävät ominaisuudet seuraavissa versioissa.

4.3 Skaalautuvuus ja suorituskyky

Pelin arkkitehtuurin suunnittelussa on otettu huomioon, se että pelaajien lisääntyessä on mahdollista lisätä web-palvelimien ja rajapintapalvelimien määrä, jolla voidaan taata mahdollisimman hyvä pelikokemus. Pelaajien määrän lisääntyessä lisätään web-palvelimien tai rajapintapalvelimien määrää ja suoritetaan niitä rinnakkain samalla tai erillisillä fyysisillä palvelimilla. Kuormantasaajien avulla voidaan jakaa pelaajat eri palvelinohjelmistoille, joiden avulla taataan, että voidaan tarjota kaikille pelaajille nopea vasteaika.

4.4 Tietoturva

Pelaajien turvallisuuden takaamiseksi pelaajan ja palvelimen väliset yhteydet suojataan kolmannen osapuolen kirjoittamalla SSL-varmenteella. Tämä takaa sen, että pelaaja voi luottaa sivustoon, ja hänen tiedot eivät päädy vääriin käsiin.

Lisäksi palvelimien välinen kommunikointi tulee suojata vähintään itse kirjoitetulla SSL-varmenteella. Eri palvelinkomponenttien tulisi myös tunnistautua toisilleen, jotta voidaan varmasti todeta, että yhdistävä palvelin on luotettava. Tunnistautuminen voidaan esimerkiksi toteuttaa Token-avaimiin perustuvalla tekniikalla, jossa asiakas-ohjelmisto luo avaimen, jonka jälkeen ohjelmisto lähettää avaimen palvelimelle, joka tarkistaa, onko avain luotettava.

5 Käytetyt teknologiat

Päätin keskittyä kirjoitushetkellä oleviin trendikkäisiin teknologioihin. Tällä hetkellä yksi suosituimmista web-palvelinsovelluskehyksistä on Node.js, käyttäen Express.js-pakettia [6]. Yksi trendikkäimmistä JavaScript-sovelluskehyksistä on React.js, joka tarjoaa yhdessä Flux-suunnittelumallin avulla uudenlaisen tavan toteuttaa käyttöliittymiä. Socket.IO-sovelluskehys tarjoaa kahdensuuntaisen tavan kommunikoida käyttöliittymän ja palveli-

men välillä. Pelinäkymän toteutukseen käytetään Three.js-sovelluskehystä, jonka tavoitteena on helpottaa 3D-grafiikan piirtoa. Pysyvän tiedon tallentamiseen tullaan käyttämään MongoDB-tietokantaohjelmistoa ja väliaikaisten tietojen säilyttämiseen tullaan käyttämään Redis-tietokantaohjelmistoa.

Tässä opinnäytetyössä keskitytään moderneihin web-teknologioihin, joten tässä ei käydä lävitse laskentapalvelimen ja synkronointipalvelimien teknologoita. Laskenta- ja synkronointipalvelimissa käytetään Qt-sovelluskehystä yhdessä C++-ohjelmointikielen kanssa.

5.1 React.JS

React.js on kohtalaisen uusi JavaScript-sovelluskehys, joka poikkeaa viime aikoina paljon huomiota saaneista JavaScript-sovelluskehyksistä, kuten Angular.js:stä, Ember.js:stä ja Backbonesta. Toisin kuin aikaisemmin mainitut sovelluskehykset perustuvat MV*-arkkitehtuuriin, niin arkkitehtuurirakenteesta React.js toteuttaa vain näkymäkerroksen (View). React.js-näkymät toteutetaan komponenttipohjaisesti, ja yksi komponentti voi sisältää useita komponentteja. Komponenttien ideana on toteuttaa esimerkiksi yksi toiminnallisuus, joka voisi olla lomake.

Reactin näkymät perustuvat niin sanottuun virtuaaliseen HTML-dokumentin objektimallin (DOM) rakenteeseen toisin kuin esimerkiksi Angular.js, joka perustuu suoraan DOM-puun muokkaamiseen, mikä on suhteellisen raskasta. Angular.js:n raskaus johtuu sovelluskehyksen tavasta tarkastaa, onko jokin DOM-puun tieto muuttunut huolimatta, siitä ollaanko siihen tekemässä muutoksia. [7.]

React-komponentteja voidaan toteuttaa JSX-syntaksin avulla tai vaihtoehtoisesti JavaScript-syntaksilla. Reactin heikkona puolena voitaisiin mainita JSX-syntaksin heikko tuki selaimissa, jonka takia JSX joudutaan kääntämään JavaScript-syntaksiksi, jota kaikki selaimet ymmärtäisivät. Yksi yleisesti käytössäoleva työkalu JSX-syntaksin kääntämiseen JavaScriptiksi on nimeltään Babel.

```
1  var Lomake = React.createClass({
2    onChange: function(e)
3    {
4      this.props.setNimi(e.target.value);
5    },
6    render: function()
7    {
8      return (
9        <form>
10         <label>Aseta nimesi:</label>
11         <input type="text" onChange={this.onChange} />
12       </form>
13     );
14   }
15 });
```

Koodiesimerkki 2: React-komponentti

Koodiesimerkissä 2 luodaan yksinkertainen React-komponentti. Rivillä 1 luodaan React-komponentti. Riveillä 2-5 määritellään metodi, joka käsittelee tekstikentästä tulevia tapahtumia ja toimittaa niitä eteenpäin. Riveillä 6-14 määritellään, mitä komponentti näyttää selaimessa. Esimerkkikomponentti toteuttaa lomakkeen, jossa pyydetään kirjoittamaan nimi.

React-komponentille on mahdollisuus määritellä tiloja. Yleisesti tiloihin tallennetaan tietoa, jota halutaan näyttää käyttäjälle. Tiedon tallennus laukaisee tapahtumaketjun, joka päivittää komponentit, joita muutos koskee.

```
1  var Main = React.createClass({
2    getInitialState: function() {
3      return {nimi: ""};
4    },
5    setNimi: function(nimi) {
6      this.setState({nimi: nimi});
7    },
8    render: function() {
9      return (
10       <div>
11         <Lomake setNimi={this.setNimi} />
12         <p>Nimesi: {this.state.nimi}</p>
13       </div>
14     );
15   }
16 });
```

Koodiesimerkki 3: React-komponentin käyttö toisessa komponentissa

Koodiesimerkissä 3 esitetään, miten itsetehtyä komponenttia voidaan käyttää toisessa komponentissa sekä miten saadaan toisesta komponentista välitettyä tietoa toiselle komponentille. Riveillä 2-4 alustetaan komponentin tila. Riveillä 5-7 määritellään metodi, joka päivittää nimiä, edellä mainittuun tilaan. Rivillä 11 lisätään esimerkkikoodin 2 lomake sekä välitetään setNimi-metodi, jotta saadaan käyttäjän nimi päivitettyä tilaan. Rivillä 12 näytetään käyttäjän nimi, joka haetaan tilasta, kun nimeä päivitetään lomakkeella. Vain tätä DOM-elemttiä päivitetään.

```
1 ReactDOM.render(
2   React.createElement(Main),
3     document.getElementById('container')
4 );
```

Koodiesimerkki 4: React-komponentin liittäminen web-sivuun

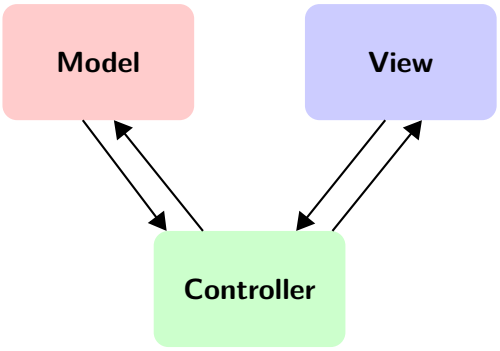
Koodiesimerkissä 4 esitetään, miten React-komponentti saadaan sivustolle näkyviin. Rivillä 2 liitetään esimerkkikoodi 3. Rivillä 3 määritellään, minkä elementin sisälle edellä esitetty komponentti lisätään. Kuvassa 8 esitetään, miltä toteutettu komponentti näyttää päätelaitteessa.

Kuva 8: Selaimessa näkyvä lopputulos

Reactin käyttäminen käyttöliitymien toteutuksessa toimii erittäin hyvin. JSX-syntaksin avulla voidaan liittää yhteen tiedostoon helposti kaikki sivuston logiikka ja eri toiminnallisuudet voidaan erotella helposti eri komponentteihin. Yksittäisten komponenttien hallinta on paljon helpompaa kuin kokonaisen sivuston. Lisäksi komponenttien etuja on myös niiden uudelleen käytettävyys. Teknologian haittapuolena on taas JSX-syntaksin tuki eri selaimissa, tästä johtuen JSX joudutaan kääntämään JavaScript muotoon, joka voi olla isoissa projekteissa hieman hidasta.

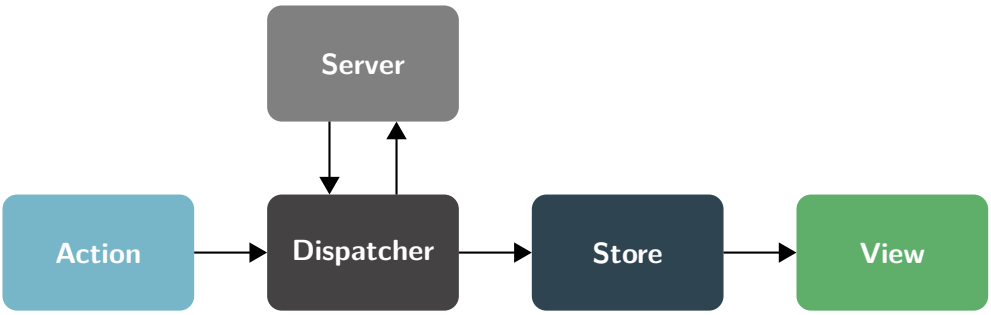
5.2 Flux ja RefluxJS

Flux-suunnittelumalli perustuu yhdensuuntaiseen datavirtaan. Sen tehtävänä on helpottaa eri React-komponenttien välistä kommunikointia ja huolehtia selaimen ja palvelimen välisestä kommunikoinnista. Fluxin ideana on, että käyttäjän tehdessä muutos, lähetetään tieto asynkronisesti palvelimelle ja tehdään muutos käyttäjälle. Käyttäessä Flux-suunnittelumallia, ei tällaistä ohjelmistoa pidä sekoittaa MV*-arkkitehtuuriin. Fluxin idea on korvata MV*-arkkitehtuuri. [8.]



Kuva 9: MVC-arkkitehtuuri

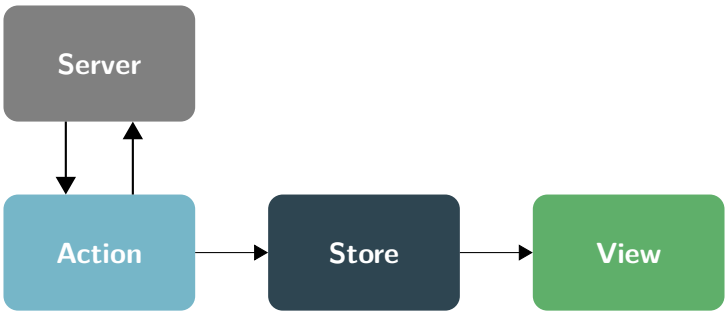
Kuvassa 9 esitetään miten MVC-arkkitehtuuri toimii. MVC-arkkitehtuurin ideana on, että näkymä (View) lähettää pyynnön ohjaimelle (Controller), joka puolestaan pyytää mallilta (Model) tietoja. Mallin käsiteltyä pyyntö, toimittaa se tiedot ohjaimelle, joka puolestaan toimittaa tiedot näkymälle.



Kuva 10: Flux-ohjelmiston yksisuuntainen tietovirta [8]

Kuvassa 10 esitetään miten Flux:n yksisuuntainen tietovirta toimii. Fluxia käyttäessä, käyttöliittymässä muutos käynnistää tapahtuman (Action), joka lähettää tiedon käsittelijälle

(Dispatcher). Käsittelijä käsittelee tiedon ja tarvittaessa lähettää tiedon palvelimelle asynkroonisesti. Käsittelyn jälkeen tiedot päivitetään varastoon (Store). Kun varasto on saanut päivitettyä tiedot, päivittää varasto tiettyä näkymää (View). [8.]



Kuva 11: Reflux-ohjelmiston suunnittelmalli

Kuvassa 11 esitetään Reflux-ohjelmistokehyksen toimintamalli. Reflux on sovelluskehys jonka tarkoitus on yksinkertaistaa Flux-suunnittelumallin käyttöä ja toimintaa. Refluxin ideana on yhdistää Dispatcher ja Actions. [9.]

Flux-suunnittelumallin hyvinä puolina on, että jokaiselle eri toiminnallisuudelle voidaan toteuttaa omat tapahtumat ja varastot, joka helpottaa eri toiminnallisuuksien ylläpitoa. Toisin kuin MVC-mallissa, jossa sovelluslogiikka toteutetaan yhteenpaikkaan. Flux-suunnittelumallin huonoina puolina on suunnittelumallin monimutkaisuus, joka eroaa MVC-mallista huomattavasti. Kehittäjällä saattaa kestää jonkin aikaa sisäistää, kuinka sovelluslogiikka toimii.

5.3 Node.js

Vuonna 2009 nuori ohjelmoija Ryan Dahl esitteli projektiaan, jossa hän oli yhdistänyt mm. Googlen V8 JavaScript -moottorin ja matalantason I/O-rajapinnan. Projekti sai nimekseen Node.js, mutta kehittäjät käyttävät yksinkertaisempaa Node-nimitystä. Rajapinta tarjoaa mm. mahdollisuuden käsitellä tiedostoja sekä käyttää eri verkkoprotokollia. Yksinkertaisen rajapinnan lisäksi kehittäjä voi käyttää kolmannen osapuolen paketteja, joiden avulla voidaan toteuttaa monipuolisempia ohjelmistoja. [10, s. 3-4, 9.]

Noden rinnalle on kehitetty Node Package Manager (NPM)-ohjelmisto, jonka tehtävänä on helpottaa kolmansien osapuolien pakettien hallintaa, joita käytetään omissa projekteissa. NPM:n tarkoituksena on helpottaa projektin pakettien asennusta. Tällöin kaikkia paketteja ei tarvitse asentaa erikseen, eikä pakettaja tarvitse siirtää toiseen ympäristöön. Projektin mukana kulkee JSON-tiedosto, jossa kerrotaan kaikki projektiin liittyvät tiedot sekä projektiin liittyvät paketit ja niiden versiot. Kohdepalvelimella kutsutaan NPM-ohjelmaa, joka lukee JSON-tiedostosta kaikki paketit ja niiden versiot, jonka jälkeen NPM lataa ja asentaa kaikki paketit.

```
1 var http = require('http');
2
3 var server = http.createServer(function(req, res)
4 {
5   res.writeHead(200, {"Content-Type": "text/html"});
6   res.end("<html><body><h1>Tervetuloa peliin</h1></body></
        html>");
7 });
8
9 server.listen(3000);
```

Koodiesimerkki 5: Node.js:n rajapinnalla toteutettu yksinkertainen web-palvelin

Koodiesimerkissä 5 esitetään Noden rajapintaan pohjautuva web-palvelin. Rivillä 1 otetaan http-rajapinta käyttöön. Rivillä 3 luodaan web-palvelin, palvelimen luonti saa parametriksi funktion, jossa käsitellään kaikki selaimelta tulleet pyynnöt. Rivillä 9 määritellään web-palvelin kuuntelemaan porttia 3000.

Tähän pohjautuva web-palvelin soveltuu hyvin pieniin projekteihin, jotka eivät vaadi suuria toiminnallisuuksia. Tämä web-palvelin ei sisällä helppoa tapaa käsitellä selaimelta tulleita pyyntöjä, vaan kehittäjän on itse toteutettava logiikka, jolla tarkistetaan kaikki tulevat pyynnöt. Ongelmaa varten on toteutettu paketti, joka helpottaa eri ominaisuuksien lisäystä, sekä helpon tavan hallita selaimelta tulevia pyyntöjä.

```
1 var ejs = require('ejs');
2 var express = require('express');
3 var app = express();
4
5 app.engine('html', ejs.renderFile);
6 app.set('view engine', 'html');
7 app.set('views', __dirname + '/views');
8 app.set('view cache', false);
```

```
 9
10 app.get('/', function(req, res)
11 {
12   res.render('index', {title: "Tervetuloa peliin"});
13 });
14
15 app.listen(3000);
```

Koodiesimerkki 6: Node.js express -pakettiin pohjautuva web-palvelin

Koodiesimerkissä 6 esitetään yksinkertainen Express.js pakettiin perustuvaa web-palvelinta käyttäen ejs-moottoria. Riveillä 1-2 ladataan halutut paketit, joita halutaan käyttää. Rivillä 3 luodaan Express-palvelin. Riveillä 5-8 määritellään ejs-moottori. Rivillä 10 esitetään esimerkkitapaus, miten käsitellään selaimelta tuleva pyyntö. Rivillä 15 määritellään porttia 3000, jossa web-palvelin pyörii.

Nodea käytetään web- ja rajapintapalvelimessa. Web-palvelimessa käytetään Noden kanssa Express-nimistä pakettia ja rajapintapalvelimessa käytetään Noden kanssa Socket.IO-sovelluskehystä, jota käsitellään lisää luvussa 5.5.

Käytettäessä Nodea voidaan toteuttaa asiakas- ja palvelinohjelmisto samalla ohjelmointikielellä. Noden etuina on koodin uudelleenkäytettävyys, eli samaa sovelluslogiikkaa voidaan käyttää sekä asiakas- ja palvelinohjelmistoissa, millä voidaan välttää saman koodin uudelleenkirjoitusta. Noden heikkona puolena voidaan pitää esimerkiksi vaikealukuisuutta. Sovelluslogiikan kasvaessa toiminnallisuuden hahmotettavuus saattaa vaikeutua, joka johtuu osittain Noden asynkroonisuudesta.

5.4 MongoDB ja mongoose

MongoDB on yksi eniten käytössä oleva NoSQL-tietokanta, mikä on tyypiltään dokumenttitietokanta. MongoDB:n JSON-tuen ansiosta dokumenttien käsittely on suhteellisen vaivatonta. Tallennettaessa JavaScript-objekteja, ne voidaan tallentaa suoraan JSON-muodossa, jolloin voidaan välttää turhaa objektejen muokkaamista. Samoin haettaessa tietoa se saadaan suoraan JSON-muodossa, jolloin siitä on helppo luoda JavaScript-objekteja tai käsitellä muulla tapaa.

```
db.pelaajat.insert({
    "kayttanimi": "simo",
    "sahkoposti": "simo@metropoli.fi",
    "salasana"  : "salasana123"
});
```

Koodiesimerkki 7: Tiedon lisääminen MongoDB-tietokantaan

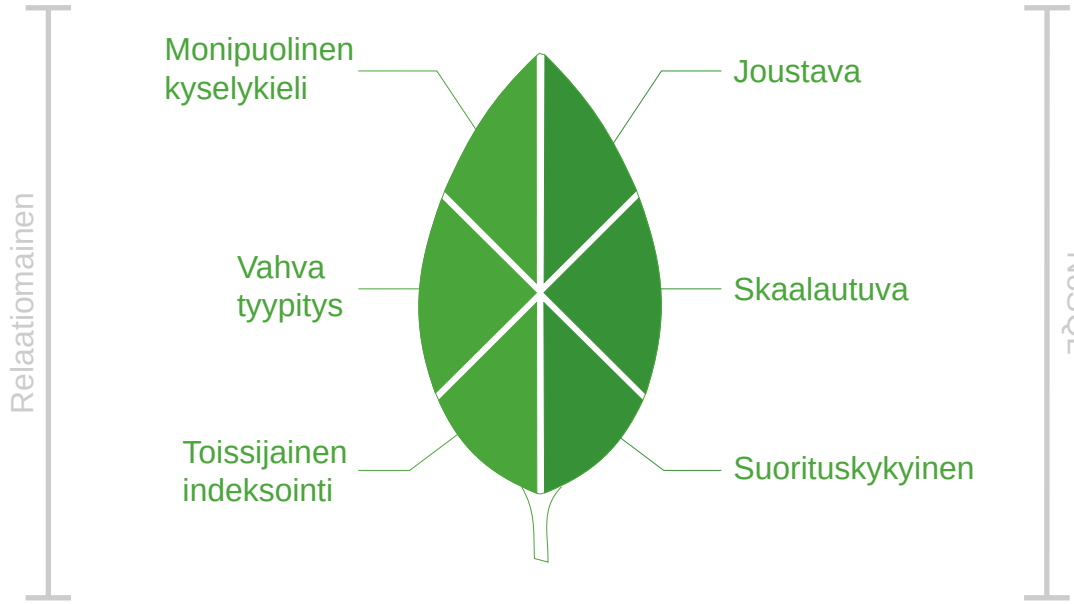
Koodiesimerkissä 7 esitetään, miten MongoDB-tietokantaan lisätään tietoa. Koodiesimerkissä ”pelaajat” määrittävät kokoelman, johon tallennetaan objekti, jolla on tiedot käyttäjänimi, sähköposti ja salasana.

```
db.pelaajat.find({
    "kayttanimi": "simo"
});

Hakutulos:
{
    "_id"       : ObjectId("55d5a7b2693ec71d65080e45"),
    "kayttanimi": "simo",
    "sahkoposti": "simo@metropoli.fi",
    "salasana"  : "salasana123"
}
```

Koodiesimerkki 8: Tiedon haku MongoDB-tietokannasta

Koodiesimerkissä 8 esitetään, miten MongoDB-tietokannasta voidaan hakea tietoa. Koodiesimerkissä haetaan pelaajat joiden ”käyttäjänimi” on ”simo”. Hakutuloksessa näytetään kaikki hakutulokset, joiden käyttäjänimi on ”simo”. Esimerkissä huomataan, miten MongoDB on luonut lisätylle objektille ”_id”-kentän sekä luonut objektille id:n.



Kuva 12: MongoDB-tietokannan ominaisuudet [11]

Kuvassa 12 esitetään MongoDB-tietokannan eri ominaisuuksia. MongoDB:n vahvuuksia verrattuna relaatiotietokantoihin ovat esimerkiksi monipuolinen kyselykieli, joka mahdollistaa monimutkaisia kyselyitä, sekä tiedon suodattamista ja muokkaamista ennen tuloksen palautusta. MongoDB tukee useita indeksejä, joiden ansiosta useasti käsiteltyyn tietoon on mahdollista päästä käsiksi nopeasti. MongoDB ei ole riippuvainen kokoelman dokumenttirakenteesta, vaan se tarjoaa joustavan tavan tallentaa erilaisia dokumentteja samaan kokoelmaan. MongoDB on lisäksi erittäin suorituskykyinen ja helposti skaalautuva. [11.]

Mongoose on kolmannen osapuolen paketti Node.js:lle, joka helpottaa MongoDB-tietokannan käyttöä. Jotta tiedon käsittely olisi mahdollisimman helppoa, voidaan mongoosen avulla luoda malli (Schema), joka kuvaa aina yhtä kokoelmaa, jonka avulla haetaan tai tallennetaan tietoa. Mallissa määritellään, mitä eri tietoja halutaan tallentaa tai hakea.

```
1  var mongoose = require('mongoose');
2
3  var pelaajaSchema = mongoose.Schema({
4    kayttanimi  : String,
5    sahkoposti  : String,
6    salasana  : String,
7    kaverit   : [
8      { nimi: String }
9    ]
10 });
11
12 var Pelaaja = mongoose.model('pelaajat', pelaajaSchema);
```

Koodiesimerkki 9: Mongoose-malli

Koodiesimerkki 9 esittää pelaajien tietokantakokoelmaa. Rivillä kolme aloitetaan mallin määrittely. Esimerkkimallissa määritellään neljä eri tietoa, joita halutaan käsitellä. Mallissa määritellään käyttäjätunnus-, sähköposti- ja salasanakentät tekstimuotoiseksi. Kaverit määritellään listaksi, joka sisältää objekteja, joiden tietona on kaverin nimi. Lopuksi kerrotaan mongooselle, että tämä malli löytyy Mongo-tietokannasta, jonka kokoelman nimi on pelaajat.

```
1  mongoose.connect('mongodb://localhost/game');
2
3  Player.findOne({_id: playerId}, function(err, player)
4  {
5    if(err)
6    {
7      // virheen käsittely
8    }
9
10   // käsitellään pelaaja objektia
11 });
```

Koodiesimerkki 10: MongoDB:n käyttö Node.js:n kanssa

Koodiesimerkissä 10 esitetään, miten saadaan haettua pelaajan tiedot kokoelmasta. Aluksi luodaan yhteys palvelimelle, jossa määritellään tietokannan nimi. Seuraavaksi käytetään koodiesimerkissä 9 luotua mallia, josta haetaan yksi pelaaja id-tunnisteen avulla. Id-tunniste saadaan esimerkiksi koodiesimerkin 8 hakutuloksista. Rivillä viisi käsitellään mahdollinen virhe. Jos virheitä ei ilmaannu, voidaan tämän jälkeen käsitellä pelaajaobjektia.

Tässä työssä MongoDB-tietokantaa käytetään hyväksi pysyvän tiedon tallentamiseen. Tietokantaan tallennetaan pelaajien tiedot sekä kaikki peliin liittyvät tiedot, kuten pelaajien kulkuneuvot. Esimerkiksi kulkuneuvojen kaikki tiedot pitää tallentaa tietokantaan, jos esimerkiksi synkronointipalvelimen fyysinen laitteisto hajoaa. Näin ollen voidaan ottaa käyttöön uusi väliaikainen palvelin ja vian korjausten jälkeen voidaan jatkaa pelinkulkua.

MongoDB-tietokannan hyviä puolia on, että kokoelmien dokumentit eivät ole sidottuna valmiiksi suunniteltuihin kenttiin ja kokoelmien välisiin relaatioihin, jonka ansiosta voidaan dokumentteja laajentaa sovelluksen ominaisuuksien kasvaessa. Lisäksi dokumentit voivat sisältää dynaamisia listoja tai objekteja, joille ei välttämättä tarvitse tehdä omaa kokoelmaa, esimerkiksi pelaajan tietoihin voidaan tallentaa kaverilista. MongoDB:n huonona puolena on, ettei kokoelmien välissä ole relaatioiden tarkistusta, kun esimerkiksi tietoja poistetaan. MongoDB-tietokantaa käyttäessä relaatioiden poistaminen eri kokoelmista täytyy toteuttaa sovelluslogiikassa.

5.5 Socket.io

Socket.io tarjoaa yksinkertaisen rajapinnan, joka helpottaa palvelimen ja selaimen välistä kommunikointia. Socket.io tarjoaa oman protokollan, jonka hyötynä on, että se tukee myös vanhoja selaimia. Oletuksena Socket.io yrittää käyttää WebSocket-yhteyttä, jos selain ei tue kyseistä ominaisuutta, koittaa Socket.io käyttää Ajax-tekniikkaa. [12, s. 87-88.]

```
1  var io = require('socket.io')(2222);
2
3  io.on('connection', function(socket)
4  {
5    // Uusi yhteys saapunut
6
7    socket.on('tapahtuma', function(data)
8    {
9      // Vastaanotettiin selaimelta tietoa
10   });
11
12   // Lähetetään selaimelle tietoa
13   socket.emit('tapahtuma2', data);
14 }
```

Koodiesimerkki 11: Socket.io-palvelin

Koodiesimerkissä 11 esitetään yksinkertainen Socket.io-palvelin node.js-sovelluskehyksen avulla. Ensimmäisellä rivillä luodaan Socket.io-palvelin. Palvelin kuuntelee porttia 2222, jonne Socket.io-asiakasohjelmistot ottavat yhteyden. Rivillä kolme määritellään tapahtuma, joka käynnistetään, jos uusi asiakasohjelmisto ottaa yhteyden. Rivillä seitsemän määritellään tapahtuma, jonka Socket.io laukaisee, kun asiakasohjelmistolta tulee samanniminen tapahtuma. Rivillä kolmetoista lähetetään asiakasohjelmistolle tietoa.

```
1  // Luodaan yhteys pelvelimelle
2  var client = io.connect("http://example.domain.com:2222");
3
4  client.on('connection', function(socket)
5  {
6    // Yhteys luotu onnistuneesti
7  }
8
9  client.on('tapahtuma2', function(data)
10 {
11   // vastaanotettiin palvelimelta tietoa
12 });
13
14 // Lähetetään palvelimelle tietoa
15 client.emit('tapahtuma', data);
```

Koodiesimerkki 12: Socket.io-asiakas

Koodiesimerkissä 12 esitetään yksinkertainen Socket.io-asiakas. Rivillä kaksi määritellään, mihin osoitteeseen Socket.io ottaa yhteyden. Rivillä neljä määritellään tapahtuma, jossa saadaan tieto, kun yhteys on valmis. Rivillä yhdeksän määritellään tapahtuma, jonka Socket.io laukaisee, kun palvelimelta tuleva viesti tulee perille. Rivillä viisitoista lähetetään palvelimelle tapahtuma. Eri tapahtumia voidaan luoda rajaton määrä. Tapahtumia voisi olla esimerkiksi kulkuneuvojen tietojen hakeminen, tehtaan ja kaupungin tietojen hakeminen sekä uusien kulkuneuvojen hankkiminen.

Käyttöliittymän halutessa tietää pelaajan kulkuneuvojen sijainnit lähettää käyttöliittymä rajapintapalvelimelle tapahtuman koodiesimerkin 12 rivin 15 mukaisesti. Tämä toimenpide voidaan esimerkiksi toteuttaa käyttöliitymällä ajastetusti tietyn ajan välein. Rajapintapalvelimen saatua pyynnön lähettää palvelin pyynnön synkronointipalvelimelle, jossa pyyntö käsitellään, ja käsittelyn jälkeen tiedot toimitetaan takaisin rajapintapalvelimelle, josta tiedot toimitetaan käyttöliittymälle koodiesimerkin 11 rivin 13 mukaisesti. Erilaisia tapah-

tumia voidaan luoda niin paljon, kuin on tarve ja tapahtumien nimiä voisi esimerkiksi olla ”kaikkien kulkuneuvojen haku” ja ”yhden kulkuneuvon tietojen haku”.

Socket.io-sovelluskehyksen hyviä puolia on kahdensuuntainen kommunikointi käyttöliittymän ja palvelimen välillä, pelaajan selaimen ei tarvitse tukea WebSocket-yhteyttä. Huonoina puolina voidaan pitää Socket.io-sovelluskehyksen skaalautuvuutta, käyttäjämäärien kasvaessa tarvittavien resurssien määrää saattaa olla vaikeaa ennustaa.

5.6 Redis

Redis on NoSQL-tietokantapalvelin ohjelmisto, joka perustuu avain-arvo pareihin. Oletuksena Redis tallentaan kaikki tiedot välimuistiin, jonka etuna on nopea lukeminen ja kirjoittaminen. Redis sisältää konfiguroitavia avaimia, joille voidaan määritellä voimassaoloaika, transaktiot sekä julkaista ja seurata ominaisuuksia. [13, s. 1.]

Voimassaoloajalla voidaan määritellä, kuinka kauan avain on saatavilla. Tätä ominaisuutta voidaan esimerkiksi käyttää käyttäjän istunnon pituuden määrittämiseen. Transaktiot mahdollistavat useiden komentojen suorittamisen yhdellä kerralla. Asiakasohjelmiston laittaessa seurannan jollekin avaimelle, saa hän ilmoituksen, kun seurattuun avaimeen julkaistaan tietoa.

Tiedon lisääminen ja hakeminen Redis-palvelimelta on erittäin nopeaa, ja juuri siksi se soveltuu hyvin väliaikaisten tietojan tallentamiseen. Kun pelaajamäärät lisääntyvät pelissä, joudutaan lisäämään rinnakkaisia web-palvelimia sekä rajapintapalvelimia. Jotta käyttäjän istunto voitaisiin jakaa näiden kaikkien palvelinohjelmistojen välillä, voidaan ottaa Redis käyttöön. Pelin kehittyessä enemmän lisätään siihen uusia ominaisuuksia. Yksi uusi ominaisuus voisi olla esimerkiksi pelin sisäinen chatti, jolla pelaajat voisivat keskustella keskenään. Chatti voisi tallentaa kaikki viestit Redis-palvelimeen, jolla voidaan jakaa kaikki viestit eri chattipalvelimien kesken sekä voitaisiin asettaa viesteille olemassaoloaika, koska chatti-viestien ei tarvitse säilyä ikuisesti. Redis-palvelinta käytetään tällä hetkellä vain pelaajien istuntojen jakamiseen eri Node-palvelimien välillä.

5.7 Three.js

WebGL on standardi 3D-grafiikalle, jota voidaan käyttää web-selaimessa. WebGL:n avulla kehittäjä voi hyödyntää tietokoneen grafiikkaohjainta käyttäen JavaScript-ohjelmointikieltä. Ennen kuin selaimet alkoivat tukemaan WebGL-standardia, käyttäjät joutuivat lataamaan kolmannen osapuolen ohjelmiston esimerkiksi Adobe Flash -ohjelmiston. WebGL on matalan tason rajapinta, joten sen kirjoittaminen voi olla joissain tilanteissa haastavaa. Tätä varten on kehitetty muutamia korkean tason työkaluja, jotka helpottavat ja nopeuttavat kehitystä. WebGL toimii suurimmalla osalla työpöytä- ja mobiiliselaimia, joten sillä saadaan toteutettua alustariippumattomia pelejä. [14, s. 2.]

Three.js on avoimen lähdekoodin JavaScript-kirjasto, joka sallii luoda ja renderöidä 3D-näkymän suoraan selaimessa. Three.js tarjoaa helppokäyttöisen rajapinnan, millä voidaan luoda ja manipuloida 3D-objekteja ja -näkymää. Rajapintaa voidaan käyttää ilman, että tiedetään, miten matalantason WebGL-rajapinta toimii tai tiedetään, miten käsitellään monimutkaisia matemaattisia kaavoja. [15, s. 7-8.]

```
1 var maailma = new THREE.Scene();
2
3 var kamera = new THREE.PerspectiveCamera( 75, window.
     innerWidth / window.innerHeight, 1, 10000 );
4 kamera.position.z = 1000;
5
6 var renderer = new THREE.WebGLRenderer();
7 renderer.setSize( window.innerWidth, window.innerHeight );
8
9 document.body.appendChild( renderer.domElement );
```

Koodiesimerkki 13: 3D-maailman luominen

Koodiesimerkissä 13 esitetään, miten saadaan luotua 3D-maailma Three.js-kirjaston avulla. Rivillä yksi luodaan 3D-maailma. Rivillä kolme luodaan perspektiivikamera, jossa määritellään kameralle perspektiivin kulma, kuvasuhde sekä kuinka läheltä ja kaukaa halutaan nähdä. Rivillä kuusi luodaan renderer-objekti, jonka jälkeen määritellään näkymän haluttu koko. Lopuksi renderer-objekti lisätään HTML-dokumenttiin, jotta kuvaa voidaan näyttää pelaajille.

```
1 var geometria = new THREE.CubeGeometry(200,400,200);
2 var materiaali = new THREE.MeshLambertMaterial({
3   color: "red"
4 });
5
6 var kuutio = new THREE.Mesh(geometria, materiaali);
7 maailma.add(kuutio);
```

Koodiesimerkki 14: Objektin luominen

Koodiesimerkissä 14 luodaan yksinkertainen punainen kuutio. Ensimmäisellä rivillä määritellään kuution muotoinen geometria, jonka kooksi määritellään 200x400x200. Seuraavalla rivillä määritellään materiaali ja materiaalille väri. Kun muoto ja materiaali on määritelty, voidaan tämän jälkeen luoda itse objekti. Lopuksi luotu kuutio lisätään maailmaan.

```
1 function paivita()
2 {
3     requestAnimationFrame( paivita );
4
5     kuutio.rotation.x += 0.01;
6     kuutio.rotation.y += 0.02;
7
8     renderer.render( maailma, kamera );
9 }
```

Koodiesimerkki 15: 3D-maailman luominen

Koodiesimerkissä 15 esitetään, miten 3D-näkymää päivitetään. Jotta 3D-näkymään saadaan liikettä, täytyy ensiksi luoda funktio, jota kutsumalla saadaan näkymä päivittymään. Rivillä kolme kutsutaan selaimen rajapinnan funktiota, joka pyytää selaimelta vuoroa, milloin voidaan seuraavaksi päivittää pelinäkymää. Riippuen pelin objektien määrästä, voidaan saavuttaa n. 60 päivitystä jokaista sekuntia kohden. Esimerkissä jokaisella päivityskerralla pyöräytetään kuutiota hieman, jotta saadaan tasainen pyöriminen aikaisesti. Lopuksi renderer-objektille kerrotaan, että halutaan näkymä piirtää pelaajalle.

Kuva 13: Selaimessa näkyvä kuutio

Kuvassa 13 esitetään koodiesimerkkeihin 13-15 perustuva kuutio. Saman objektin piirto käyttäen WebGL-rajapintaa vaatii huomattavasti enemmän koodia.

Three.js-kirjastolla saadaan toteutettua pelinäkymä paljon pienemmällä vaivalla, kuin käyttäen suoraan matalantason WebGL-rajapintaa. Kirjasto tarjoaa valmiiksi erilaisia muotoja, materiaaleja, sekä valaistukseen liittyviä lamppuja. Kirjasto tarjoaa myös tavan toteuttaa minkä tahansa muotoisia objekteja ja niihin liittyviä tekstuureita. Prototyypin toteutuksen aikana en päässyt vielä toteuttamaan minkäänlaista suorituskykyyn liittyviä kokeiluja pelille, mutta kirjaston esimerkkiprojekteista löytyy erikokoisia projekteja, mitä kirjastolla on toteutettu. Lisäksi Three.js tarjoaa myös tuen vanhemmille selaimille, jotka eivät tue WebGL-rajapintaa, näissä tapauksissa Three.js yrittää käyttää jotakin muuta teknologiaa.

6 Yhteenveto

Opinnäytetyössä tutustuttiin moderneihin web-teknologioihin selainpohjaisessa pelikehityksessä. Samalla tutustuttiin, miten perinteiset teknologiat ovat kehittyneet sekä miten pelit ovat kommunikoineet palvelimien kanssa. Lisäksi tutustuttiin, mitä teknologioita on tullut lisää perinteisesti käytössä olevien teknologioiden rinnalle.

Opinnäytetyön aikana suunniteltiin prototyyppi reaaliaikaiselle selainpohjaiselle pelille, jossa kaikki tärkeä laskenta tullaan toteuttamaan palvelinarkkitehtuurin sisällä. Pelin kartta toteutetaan verkkomaisella ratkaisulla, jossa lyhimpien reittien laskentaan käytetään Edsger Dijkstran toteuttamaan matemaattista algoritmia. Palvelinarkkitehtuuri sisältää neljä eri palvelinohjelmistoa, jotka voidaan hajauttaa eri palvelimilla. Hajautuksen avulla voidaan rajapinta- ja web-palvelimien määrää kasvattaa helposti, kun pelaajien määrä kasvaa.

Prototyyppiin valitsin kirjoitushetkellä olevia trendikkäimpiä teknologioita, joiden avulla lähdin tutustumaan, miten nämä eri teknologiat soveltuvat yhteen ja miten näitä voitaisiin käyttää. Käyttöliittymään valitsin React.js- ja Reflux-ohjelmistokehykset, joiden avulla voidaan toteuttaa käyttöliittymiä helposti. React.js-ohjelmistokehyksen avulla voidaan käyttää JSX-syntaksia, joka helpottaa käyttöliittymä komponenttien toteutusta. Reflux tuo helpotetun tavan käyttää Flux-suunnittelumallia, joka helpottaa eri React.js-komponenttien välistä kommunikointia. Pelaajan päätelaitteen ja palvelimen välillä käytetään Socket.io-sovelluskehystä, jonka avulla saadaan toteutettua kahdensuuntainen kommunikointi. Three.js-kirjasto tuo korkeantason rajapinnan 3D-grafiikan tuottamiseen selainympäristössä, mikä helpottaa 3D-maailman luomista ja maailman päivittämisen reaaliajassa.

Web-palvelimeksi valitsin Node.js-sovelluskehyksen, joka perustuu Googlen V8 JavaScript -moottoriin, jossa käytetään Express.js-pakettia, joka tarjoaa helpon tavan toteuttaa web-palvelimia. Tietokannaksi valitsin MongoDB-dokumenttitietokannan, joka sovel-

tuu Node.js-sovelluskehyksen kanssa yhteen erittäin hyvin, koska MongoDB käyttää kokoelmissa JSON-tietotyyppiä, jota on helppo muuttaa JavaScript-objektiksi.

Työn tuloksena saatiin toteutettua toimiva prototyyppi, jonka avulla tutustuttiin, kuinka valitut palvelin- ja selainteknologiat soveltuvat selainpohjaisen pelin toteutukseen. Prototyypin avulla tutkittiin, kuinka palvelinohjelmistot tulevat kommunikoivat keskenään. Prototyypin kehityksen yhteydessä havaittiin muutamia mahdollisia asioita, joita voitaisiin toteuttaa paremmin ja mitä lisäominaisuuksia voitaisiin lisätä.

Lähteet

1 S. Wang, Paul. Dynamic Web Programming and HTML5. USA: CRC Press. 2012.

2 A Short History of JavaScript. 2012. Verkkodokumentti. <https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript>. Luettu Marraskuu 8, 2015.

3 James Garret, Jesse. Ajax: A New Approach to Web Applications. 2015. Verkkodokumentti. <http://adaptivepath.org/ideas/ajax-new-approach-web-applications/>. Luettu Lokakuu 20, 2015.

4 Wright, Tim. Learning Javascript. USA: Addison-Wesley. 2012.

5 Joyner, David, Van Nguyen, Minh ja Cohen, Nathann. Algorithmic Graph Theory. 2011. Verkkodokumentti. <http://code.google.com/p/graph-theory-algorithms-book/>. Luettu Marraskuu 5, 2015.

6 Most starred packages.. Verkkodokumentti. <https://www.npmjs.com/browse/star>. Luettu Marraskuu 23, 2015.

7 Reactjs Performance & Speed?. 2015. Verkkodokumentti. 500Tech. <http://blog.500tech.com/is-reactjs-fast/>. Luettu Marraskuu 16, 2015.

8 Flux | Application Architecture for Building User Interfaces. 2015. Verkkodokumentti. Facebook. <https://facebook.github.io/flux/docs/overview.html>. Luettu Lokakuu 28, 2015.

9 Brassman, Mikael. Deconstructing ReactJS's Flux. 2014. Verkkodokumentti. <http://spoike.ghost.io/deconstructing-reactjss-flux/>. Luettu Lokakuu 28, 2015.

10 Teixeira, Pedro. Professional Node.js: Building JavaScript-Based Scalable Software. USA: John Wiley & Sons, Inc.. 2013.

11 MongoDB Architecture. 2015. Verkkodokumentti. <https://www.mongodb.com/mongodb-architecture>. Luettu Marraskuu 16, 2015.

12 Rai, Rohit. Socket.IO Real-time Web Application Development. UK: Packt Publiching. 2013.

13 Dayvson Da Silva, Maxwell ja Lopes Tavares, Hugo. Redis Essentials. UK: Packt Publiching. 2015.

14 Parisi, Tony. WebGL: Up and Running. USA: O’Reilly Media. 2012.

15 Dirksen, Jos. Three.js Essentials. UK: Packt Publiching. 2014.