Opinnäytetyö (AMK)

Tietojenkäsittelyn koulutus

2021

Ilmari Tyrkkö

PELIPALVELUN LUOMINEN PILVIPALVELIMELLE

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely

2021 | 34 sivua

Ilmari Tyrkkö

PELIPALVELUN LUOMINEN PILVIPALVELIMELLE

Tämän opinnäytetyön tavoitteena oli toteuttaa www-palvelu, jossa käyttäjät voivat omissa ryhmissään eli kimpoissaan, kilpailla siitä, kenen valitsemat joukkueet pärjäävät NHL-liigassa parhaiten. Joukkueet valitaan ennen pelin alkua varaustilaisuudessa, jonka alkamisajan voi määrittää ryhmän ylläpitäjä. Kun joukkueet on valittu, voi itse pelin seuranta alkaa. Pelaaja saa pisteen valitsemansa joukkueen voitosta ja vain voitosta, eli esim. tasapelistä ei saa pisteitä. Runkosarjan päätteeksi eniten pisteitä kerännyt pelaaja on voittaja.

Opinnäytetyön alussa sovellukselle kartoitettiin palvelut ja valittiin kehitystekniikat. Käyttöliittymän (frontend) tekniikaksi valittiin React sen laajan tuen vuoksi. Taustasovelluksen (backend) tekniikka toteutettiin NodeJS:llä, sillä se on Reactin tapaan JavaScript-pohjainen. Tietokannaksi projektiin valittiin MongoDB, koska siihen löytyy hyviä kirjastoja NodeJS:stä. Käyttöliittymän ja taustasovellusten ajamiseen hankittiin pilvipalvelin, jolle asennettiin Nginx-palvelinohjelmisto ohjaamaan liikennettä. Palvelulle rekisteröitiin myös verkkotunnus kiakkoterot.fi, jonka liikenne ohjattiin hankitulle pilvipalvelimelle.

Lopputulemana opinnäytetyön tavoitteet saatiin täytettyä, eli toimiva www-palvelu saatiin rakennettua ja onnistunut varaustilaisuus saatiin järjestettyä. Ensimmäinen peli päättyy huhtikuussa 2022. Opinnäytetyön sivusto jäi monelta osin viimeistelemättömäksi, mutta samalla sen jatkokehitys ja laajennettavuus on mahdollista. Etenkin sivuston ulkoasu jäi hyvin yksinkertaiseksi, sillä opinnäytetyössä keskityttiin ensisijaisesti tekniseen toimivuuteen.

Opinnäytetyön koko lähdekoodi on saatavilla GitHubista.

ASIASANAT:

React, NodeJS, pilvipalvelu, WebSocket, MongoDB, Nginx

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology

2021 | 34 pages

Ilmari Tyrkkö

CREATING A GAME SERVICE TO A CLOUD SERVER

The goal of this thesis was to implement a www service where users can compete in their own groups, whose chosen teams do best in the National Hockey League (NHL). The teams would be selected prior to the start of the game at a draft event, which start time is determined by the group administrator. Once the teams would have been selected, the game would begin. The player would receive a point for every victory of the team of his/her choice and only for the victory but no points would be awarded for a draw, for example. The player with the most points at the end of the regular season would be the winner.

At the beginning of the thesis, services were mapped for the application and development technologies were selected. React was chosen as the frontend technology because of its extensive support. The backend technology was implemented with NodeJS, as it is JavaScript-based like React. MongoDB was chosen as the database for the project because it has good libraries to use with NodeJS. To run backend and frontend applications, a cloud server was purchased and Nginx server software was installed to control traffic. The domain name kiakkoterot.fi was also registered for the service, the traffic of which was directed to the above-mentioned cloud server.

In the end, the thesis met its requirements and a successful draft event was held. The first game will end in April 2022. The project remained largely unfinished, but, at the same time, its further development and expandability is possible. In particular, the layout of the site remained very simple as the thesis focused primarily on technical functionality.

The full source code for the project is available at GitHub.

KEYWORDS:

React, NodeJS, cloud service, WebSocket, MongoDB, Nginx

SISÄLTÖ

KUVAT

KOODIT

KÄYTETTY SANASTO

Backend (back end)	Sovellus, joka toimii rajapintana tietokannan ja frontendin välillä
Draft	Varaustilaisuus
Frontend (front end)	Sovellus, joka näkyy käyttäjälle esim. selaimessa
HTML	HyperText Markup Language, Www-sivuissa käytetty merkintäkieli
HTTP	HyperText Transfer Protocol on protokolla, jota käytetään yleisesti dokumenttien ja datan siirtoon internetissä
JSON	Avoimen standardin tiedostomuoto tiedonvälitykseen
NHL	National Hockey League on jääkiekkosarja Pohjois-Amerikassa
Token	Sessionhallintaan luotu tunniste, jolla sessio ja käyttäjä voidaan tunnistaa

1 JOHDANTO

Nykypäivänä harva jaksaa tilastoida mitään käsin paperille, vaan apukeinoksi on otettu tietokone, jolla voi tallentaa tilastoja esimerkiksi Excel-taulukoihin. Excel-taulukoihin kirjaaminen on toki sekin jo itsessään näppärää, mutta eikö taulukoiden täyttämisen voisi automatisoida kokonaan? Kyllä voi ja tässä opinnäytetyössä käymme lävitse, miten ennen Excel-taulukoissa ylläpidetty peli automatisoidaan www-ympäristöön.

Tässä opinnäytetyössä siis luodaan www-palvelu, joka laitetaan yleisesti saataville internetiin. Samalla opinnäytetyössä dokumentoidaan kyseisen palvelun luomisen kokonaisprosessi. Prosessin osioihin kuuluvat tietokannan, backend-sovelluksen, frontend-sovelluksen ja palvelimen luonti sekä verkkotunnuksen rekisteröinti. Työn laajuuden vuoksi prosessista dokumentoidaan vain keskeisimmät osiot. Prosessin toimeenpanon jälkeen opinnäytetyössä tarkastellaan kriittisesti palvelun onnistumista ja sen jatkokehitysmahdollisuuksia.

Tässä opinnäytetyössä luodaan pelipalvelu, johon käyttäjät voivat liittyä ja luoda keskenään omia sarjojaan, joista pelipalvelussa käytetään nimitystä kimpat. Pelin ideana on veikata NHL-liigan (National Hockey League) joukkueiden menestymistä runkosarjassa. Ennen pelisarjan alkua jokainen pelaaja valitsee vuorollaan joukkueen, joiden runkosarjavoitoista pelaaja saa pisteen. Valintavuorot arvotaan ennen sarjan alkua. Pelaajan joukkueiden määrä riippuu pelisarjaan osallistuvien pelaajien määrästä. Esimerkiksi kaudella 2021–22 NHL:ssä pelaa 32 eri joukkuetta. Näin ollen jos kimppaan osallistuu 9 pelaajaa, saa jokainen pelaaja valita 3 joukkuetta ja 5 joukkuetta jää valitsematta. Runkosarjan lopussa eniten pisteitä kerännyt pelaaja luonnollisesti voittaa pelin.

Pelin idean takana on vuodesta 2013 toiminut WhatsApp-yhteisö KiakkoTerot Ry, jonka jäsenet koostuvat n. 30–40-vuotiaista miehistä. Yhteisön jäseniä yhdistää se, että kaikki ovat jossain vaiheessa elämäänsä asuneet Tampereen Hervannassa ja pitävät jääkiekosta. Vuosien varrella yhteisön kesken on järjestetty erilaisia leikkimielisiä visailuja, joiden tuloksia on hallittu tähän mennessä Excel-taulukoilla tai ruutupaperilla. Opinnäytetyössä käsiteltävän pelin hallinta edellä mainituilla tekniikoilla on ollut kuitenkin verrattain työlästä ja kömpelöä, eikä yhteisön jäsenten aika ole enää riittänyt sen mielekkääseen ylläpitoon. Siispä tarve automatisoidulle pelipalvelulle on ollut akuutti.

Tämän opinnäytetyön tutkimusongelma on miten automatisoida peli www-ympäristöön. Opinnäytetyön tutkimusmenetelmä on konstruktiivinen, eli ongelmaan kehitetään ratkaisua. Opinnäytetyö on osin myös evaluoiva, koska valittujen ratkaisujen ja niiden vaihtoehtojen soveltuvuutta tehtävään tarkastellaan jälkikäteen.

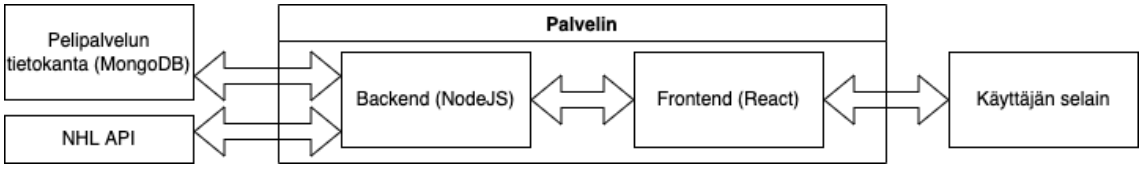
Opinnäytetyön lopullisena tavoitteena on saada aikaan toimiva pelipalvelu, jossa pelin varaustilaisuus voidaan suorittaa reaaliaikaisesti. Varaustilaisuuden jälkeen palvelussa pitää pystyä seuraamaan pelin kulkua eli NHL-liigan runkosarjan edistymistä. Lopuksi opinnäytetyön onnistumista ja valittuja ratkaisuja tarkastellaan kriittisesti.

2 SOVELLUKSEN VAATIMUKSET JA PALVELUIDEN KARTOITTAMINEN

Tässä luvussa käydään läpi, mitä vaatimuksia sovellukselle oli ja miksi valittuihin palveluihin ja tekniikoihin päädyttiin. Tarkoituksena on avata lyhyesti myös mitä muita vaihtoehtoja olisi ollut tarjolla.

Sovelluksen perusideana oli luoda alusta, johon käyttäjät voivat kirjautua millä tahansa www-selaimella internetin yli ja pelata peliä, sekä seurata sen edistymistä. Tämän toteuttamiseksi vaaditaan selaimessa näkyvä frontend -sovellus, mahdollisesti myös frontendin ja tietokannan välillä keskusteleva backend-sovellus ja itse tietokanta, jolle em. sovelluksett lähettävät pyyntöjä, sekä vastaanottaa dataa. Erillinen backend-sovellus ei tietyissä tapauksissa ole pakollinen, mutta tähän pelipalveluun sellainen rakennettiin (ks. Backend), koska palvelun piti ottaa yhteyttä myös kolmannen osapuolen APIin, josta saatiin NHL-liigan dataa.

Jotta näihin saadaan yhteys mistä tahansa, tarvitaan myös palvelin, jossa tarvittavat osat koko pelipalvelusta sijaitsevat fyysisesti. Tämä palvelin hankittiin tämän opinnäytetyön tapauksessa pilvipalvelusta, josta voi ostaa minuuteissa palvelintietokoneen käyttöönsä. Pilvipalvelusta saadaan käyttöön palvelimen ip-osoite, johon palvelun käyttäjät ottavat yhteyttä. Helpottaaksemme käyttäjän osaa, päädyimme kuitenkin myös rekisteröimään pelipalvelulle verkkotunnuksen, kiakkoterot.fi, jotta palveluun siirtyminen olisi helpompaa. Tiedon kulun palvelussa voi havainnollistaa alla olevasta kuvasta (Kuva 1).



Kuva 1. Tiedon kulku palvelussa

2.1 Frontend

Frontendin päävaatimuksina olivat seuraavat:

- käyttäjän rekisteröityminen ja kirjautuminen
- kimpan luominen, siihen käyttäjien kutsuminen ja draftin aloitusajan valinta
- reaaliajassa tapahtuva draft, jossa käyttäjät valitsevat joukkueet
- kimpan tilanteen seuraaminen
- tuki yleisimmille selaimille: Chrome, Safari, Firefox.

Frontend päädyttiin toteuttamaan React (ReactJS) -JavaScript-kirjastoa avuksi käyttäen lähinnä sen laajan käyttäjäkunnan ja hyvän tuen vuoksi. Muita vaihtoehtoja olisi ollut esim. AngularJS tai VueJS, joilla olisi pystynyt toteuttamaan samanlaisen käyttöliittymän, mutta Reactin valintaa puolsi myös omat hyvät kokemukseni.

2.2 Backend

Backendin päävaatimuksina olivat seuraavat:

- ohjata liikennettä frontendin, tietokannan ja NHL APIn välillä
- automatisoida tietojen haku NHL APIsta
- toimia API-rajapintana frontendille
- validoida (tarkistaa) frontendistä lähetetty data ennen tietokantaan lähettämistä
- pitää yllä WebSocket-yhteyksiä draftin aikana.

Backend toteutettiin NodeJS-sovelluksena, jonka tarkoitus on toimia rajapintana frontendin ja tietokannan välillä. Reactilla toteutettu frontend-sovellus olisi itsessään jo varsin kykenevä tekemään pyyntöjä suoraan tietokantaan, mutta tällaisen backend-sovelluksen avulla teemme pelipalvelun käyttämisestä käyttäjälle turvallisempaa (salasanat käsitellään erillisessä sovelluksessa, eikä front endissä) ja pystymme paremmin hallitsemaan automatisoitua datan hakua NHL-APIsta.

Backend -sovelluksen olisi voinut toteuttaa usealla eri ohjelmointikielellä (Python, C++, jne.) ja tavalla, mutta NodeJS tuntui loogiselta vaihtoehdolta, sillä myös se Reactin tapaan, on JavaScript-pohjainen.

2.3 Tietokanta

Tietokannan päävaatimuksena oli pitää tietoa yllä seuraavista:

- käyttäjät (nimi, käyttäjätunnus, salasana)
- NHL (joukkueet, joukkueiden pisteet)
- kimpat (osallistujat, aloitusaika, valitut joukkueet)

Tietokannaksi valikoitui MongoDB, koska tämän pelipalvelun mittakaavassa sitä on ilmaista käyttää. MongoDB on ns. NoSQL-tietokanta, joka mahdollistaa joustavamman tavan käsitellä dataa tietokannassa. MongoDB voidaan asentaa joko lokaalisti, omalle palvelimelle tai käyttää sitä pilvitietokantana. Tässä projektissa päädyin käyttämään pilvipalveluvaihtoehtoa, jotta työ ei skaalaudu liian isoksi.

Muista NoSQL-tietokantatyypeistä Google Firebase olisi voinut soveltua palveluun melko hyvin, mutta jatkokehitystä varten MongoDB oli parempi vaihtoehto, sillä Firebasea ei voi asentaa lokaalisti. Muita varteenotettavia vaihtoehtoja oli perinteisemmät relaatiotietokannat, joista etenkin MySQL ja SQLite.

2.4 NHL API -ohjelmointirajapinta

Pelin dataa varten kartoitin erilaisia API-rajapintoja, joista NHL-liigan dataa voisi HTTP-pyynnöillä saada. Vaihtoehtoja oli useita, mutta suurin osa näistä oli joko kokonaan maksullisia tai toimivat epäluotettavasti, eli dataa tuli ulos vain silloin tällöin tai dokumentointi ei ollut enää ajan tasalla.

API-palveluista valikoitui käyttöön SportsData.io, jolla oli palvelustaan hyvä dokumentointi, se toimi luotettavasti ja pelipalvelun tarpeisiin käyttö oli ilmaista. Kyseinen API mahdollisti ilmaiskäyttäjälle 1000 pyyntöä kuukaudessa, joilla sarjataulukon päivittäminen oli helposti mahdollista (SportsData.io 2021: https://sportsdata.io/developers/faq).

Mikäli peliä haluaisi laajentaa esim. käyttämään pelaajien tai otteluiden dataa, ei em. API enää ilmaiskäytössä soveltuisikaan käyttöön, sillä se ilmaiskäyttäjille osa datasta on nk.

”scrambled dataa”, eli datan arvoja on muunneltu satunnaisesti 5–20 %. Sarjataulukoita kyseinen ”scrambled data” ei kuitenkaan sisällä, joten sarjataulukko on luotettava.

2.5 Palvelin

Frontend- ja backend-sovellusten ajamiseen tarvittiin palvelin, jotta niihin saataisiin yhteys aina tarvittaessa. Palvelin päätettiin hankkia pilvipalvelusta, eli kolmannelta osapuolelta, sillä lokaalin palvelimen ylläpito on työlästä ja etenkin tilaa vievää.

Pilvipalvelimien tarjoajia on lukuisia ja päätöksessä mentiinkin pääasiassa hinta edellä. Tarjoajaksi valikoitui Hetzner, jolla on hosting-palveluita Suomessa, kilpailukykyiseen hintaan. Hetznerin valintaa puolsi myös mahdollisuus lopettaa tilaus välittömästi näin halutessaan.

Palvelintietokoneen kokoonpano oli seuraava:

- Intel Xeon -prosessori
- 2GB RAM-muistia
- 20GB SSD-kiintolevy
- mahdollisuus 20TB liikenteeseen
- Ubuntu 20.04.1 -käyttöjärjestelmä.

Palvelimelle asennettiin Nginx-palvelinohjelmisto. Toisena vaihtoehtona oli Apache, mutta Nginx vaikutti kevyempänä sopivammalta tehtävään.

2.6 Verkkotunnus

Jotta käyttäjien ei tarvitsisi muistaa pelipalveluun päästääkseen vaikeaselkoista ip-osoitetta, rekisteröitiin palvelulle verkkotunnus kiakkoterot.fi. Verkkotunnuksen rekisteröinti tehtiin suomalaiselta palveluntarjoajalta, jonka käyttöliittymän kautta päästiin asettamaan verkkotunnuksen DNS-tietueisiin Hetznerin pilvipalvelimen IP-osoite, jonka jälkeen kaikki liikenne ohjautui oikeaan paikkaan.

3 KÄYTETYT TEKNIIKAT

Tässä luvussa syvennytään tarkemmin edellisessä kappaleessa jo mainittuihin teknologioihin, ohjelmistokirjastoihin, niiden lisäosiin ja protokolliin.

3.1 React-kirjasto

React (ReactJS) on erityisesti SPA (single-page application) -applikaatioiden ja niiden käyttöliittymien luomiseen suunniteltu JavaScript-kirjasto. Se mahdollistaa dynaamisten sivujen luomisen, eli data sivulla voi vaihtua ilman että käyttäjä lataa sivua uudestaan. React on luotu alun perin vuonna 2011 Facebookin insinöörien toimesta ja sen ympärille on vuosien varrella kasvanut laaja tuki ja yhteisö. Sen ydintoiminta perustuu elinkaarimetodeihin ja komponentteihin, joita voi toistaa koodissa haluamansa määrän. (Pandit, Nitin 2021: What And Why React.JS. Osoitteessa: https://www.c-sharpcorner.com/article/what-and-why-reactjs/)

Reactissa komponentteja voi rakentaa joko luokkakomponenteiksi tai funktionaalisiksi komponenteiksi. Funktionaaliset komponentit (React Hooks) ovat käytännössä jo syrjäyttäneet vanhemmat luokkakomponentit, joten myös tässä opinnäytetyössä keskitytään vain funktionaalisiin komponentteihin. (Grujicic, Nikola 2021: Are React class components still needed in 2021. Osoitteessa: https://www.framelessgrid.com/are-react-class-components-still-needed-in-2021/)

Komponentit rakennetaan JSX-syntaksilla, mikä mahdollistaa JavaScriptin ja HTML-koodin yhdistämisen tehokkaasti. Näitä komponentteja voidaan kutsua toisten komponenttien sisältä ja näin ollen voidaan luoda helpommin luettavaa koodia ja vähentää koodin toistoa. (Pandit, Nitin 2021: What And Why React.JS. Osoitteessa: https://www.c-sharpcorner.com/article/what-and-why-reactjs/)

Alla (koodi 1) nähdään esimerkki opinnäytetyössä käytettävästä React-komponentista Matches, joka tulostaa seuraavan yön otteluparit taulukkoon. Komponentille lähetetään attribuutteina teamData- ja draft-objektit, joiden sisältöä hyödynnetään ottelutaulukon rakentamisessa. Seuraavalla rivillä määritetään komponentille tilamuuttuja (State Hook) matchData, joka voidaan päivittää kutsumalla sille määritettyä funktiota setMatchData.

Tilamuuttujalle annetaan oletusarvo kutsumalla Reactin sisään rakennettua funktiota useState ja sille annetaan oletusarvoksi tyhjä taulukko (array), eli [].

Seuraavaksi koodissa tulee vastaan niin ikään Reactin sisään rakennettu funktio useEffect, jolle parametreiksi annetaan funktio ja tyhjä taulukko (array). Annettu funktio lähettää erillisen axios-kirjaston avulla HTTP GET-pyynnön backend-sovellukseen. Kun backend-sovellus vastaa oikein, asetetaan vastauksessa tuleva data matchData-muuttujaan, lähettämällä data setMatchData-funktiolle. Tätä ennen data siivilöidään (filter), niin että jäljelle jää vain seuraavan yönä pelattavat ottelut. Tämä tapahtuu vertailemalla Date-objekteja ja lisäämällä toiseen arvoon 25200000 millisekuntia (7 tuntia) aikaeron korjaamiseksi. Mikäli backend-sovellus ei vastaa tai vastaa virheellisesti, komponentti tulostaa (console.log) virheviestin konsoliin.

Koska useEffectille annettiin parametriksi tyhjä taulukko, kutsutaan sille annettua funktiota yhden kerran kun Matches-komponentti ladataan. Taulukon sisään voi laittaa myös muuttuja-arvoja, joka johtaa siihen, että kyseiselle useEffectille annettua funktiota kutsutaan vain, kun muuttuja-arvo on vaihtunut.

Komponentti palauttaa (return) JSX-syntaksia. Syntaksin alussa tarkistetaan (matchData.length > 0) onko matchData-taulukko tyhjä. Mikäli taulukko on tyhjä, komponentti palauttaa tyhjän string-muuttujan, ''. Mikäli taas taulukossa on dataa, komponentti rakentaa taulukon HTML-koodia ja komponentin objekteja (teamData, draft ja matchData) hyödyntäen.

Koodi 1. Esimerkki React-komponentista

```
import React, { useEffect, useState } from 'react'
import axios from 'axios'
import { Table } from 'react-bootstrap'
import { APIURL } from '../services/addresses'

export default function Matches({ teamData, draft }) {
  const [matchData, setMatchData] = useState([])

  useEffect(async () => {
    await axios.get(`${APIURL}/nhl/matches`)
      .then(res => setMatchData(res.data.filter(d => {
        return ((new Date(d.DateTime).getTime() + 25200000) > new
Date().getTime())
      }
      )))
```

```
    .catch(e => console.log(e))
}, [])

return (
  <>
    {matchData.length > 0 ?
      <div>
        <h5>Seuraavat pelit</h5>
        <Table>
          <thead>
            <tr>
              <th></th>
              <th>Koti</th>
              <th></th>
              <th>Pelaaja</th>
              <th></th>
              <th>Pelaaja</th>
              <th></th>
              <th>Vieras</th>
            </tr>
          </thead>
          <tbody>
            {matchData.map(m => {
              let h = draft.teamsChosen.find(e => e.team.Key === m.HomeTeam)
              let a = draft.teamsChosen.find(e => e.team.Key === m.AwayTeam)
              let hData = teamData.find(t => t.Key === m.HomeTeam)
              let aData = teamData.find(t => t.Key === m.AwayTeam)

              return (
                <tr key={m.GameID}>
                  <td>{new Date(m.DateTime).toLocaleString('fi-FI')}</td>
                  <td>{hData.City} {hData.Name}</td>
                  <td>
                    <img
                      alt={hData.Key}
                      src={hData.WikipediaLogoUrl}
                      height='20px'
                    />
                  </td>
                  <td>{h ? h.user.name : ''}</td>
                  <td>vs</td>
                  <td>{a ? a.user.name : ''}</td>
                  <td>
                    <img
                      alt={aData.Key}
                      src={aData.WikipediaLogoUrl}
                      height='20px'
                    />
                  </td>
```

```
                    <td>{aData.City} {aData.Name}</td>
                  </tr>
                )
              })}
            </tbody>

          </Table>
        </div> : ''
      }
    </>
  )
}
```

3.2 NodeJS-ajoympäristö ja Express-kirjasto

NodeJS (Node.JS) on ilmainen avoimeen lähdekoodiin perustuva JavaScript-ajoympäristö, joka ajaa koodia selaimen ulkopuolella. Se toimii asynkronisesti, eli se voi käsitellä useampaa pyyntöä yhtäaikaisesti. NodeJS:ään on Reactin tapaan saatavilla paljon lisäosia (plugins), joista Express (Express.js) on yleisin. (Ivanovs, Alex 2020: Top 32 NPM Packages for Node.js Developers 2020. Osoitteessa: https://colorlib.com/wp/npm-packages-node-js/)

NodeJS:n päällä pelipalvelu ajaa Express-verkkosovelluskehystä, joka mahdollistaa tehokkaasti HTTP-pyyntöjen reitittämisen. Alla olevassa esimerkissä (koodi 2) on Expressiä hyödyntävä NodeJS-sovellus, jossa luodaan palvelin kuuntelemaan pyyntöjä portista 3000. Kun sovellus on päällä ja em. porttiin lähetetään GET-pyyntö, sovellus vastaa pyyntöön ’Hello World!’.

Koodi 2. Esimerkki NodeJS / Express -sovelluksesta.

```
const express = require('express')
const app = express()
const port = 3000

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

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`)
})
```

Expressin oheen NodeJS-sovellukseen lisättiin WebSocket-protokollan tuki (ks. 3.3 WebSocket), joka mahdollistaa kaksisuuntaisen kommunikaation NodeJS-sovelluksen ja käyttäjän selaimen välillä.

3.3 WebSocket-protokolla

Koska palveluun oli tarve saada rakennettua reaaliajassa päivittyvä draft-tilaisuus, ei HTTP-protokollan kyvyt (ns. half-duplex) pyyntöjen lähettämiseen tähän soveltuneet, sillä sen käyttö tarkoittaisi jatkuvia pyyntöjä selaimelta palvelimille, mikä voisi johtaa palvelimen ruuhkautumiseen tms. virheeseen.

Ratkaisuksi ongelmaan löytyi WebSocket-protokolla, joka mahdollistaa ns. yhtäaikaisen keskustelun palvelimen ja selaimen välillä (full-duplex). Kun WebSocket-yhteys selaimen ja palvelimen välille on luotu, voi palvelin lähettää dataa selaimelle suoraan ilman, että selaimen tarvitsee sitä erikseen pyytää. Tämä mahdollistaa yhteyksien käytön vain tarvittaessa. (Ahmed, Osama 2019: WebSockets Not In A Nutshell. Osoitteessa: https://medium.com/@osama.scream2/websockets-in-a-nutshell-11dce96daf4e)

WebSockettia hyödyntääkseen täytyy sekä Expressiin että Reactiin asentaa WebSocket-yhteyden mahdollistavat lisäosat (plugins). Näitä lisäosia on useita erilaisia, mutta tähän projektiin valittiin Expressille express-ws-kirjasto ja Reactiin W3C WebSocket -kirjasto.

3.4 MongoDB-tietokanta

MongoDB valittiin projektiin käyttöön toki siksi, että projektin mittakaavassa se on ilmainen mutta myös isoksi osaksi siksi, että sille löytyy hyvä NodeJS:lle suunniteltu kirjasto: Mongoose.

MongoDB:n parhaita puolia ovat sen nopeus ja sen helppokäyttöiset skeemat datalle, jotka osaltaan mahdollistavat datan validoinnin ilman erillisiä tarkistuksia. Skeemoihin voidaan määritellä dokumentin eri muuttujien tyypit (string, number, jne.) ja niille voidaan lisämääreitä, kuten maksimipituus. Jos dokumenttia tallentaessa jokin tyyppi ei täsmää, dokumenttia ei tallenneta ja siitä annetaan virheviesti. Skeeman perusteella voidaan rakentaa malliobjekti (model), joita voidaan käyttää tallentaessa ja haettaessa dataa tietokannasta. (MongoDB 2021: Schemas. Osoitteessa: https://docs.mongodb.com/realm/schemas/)

Koodi 3. Esimerkki skeemasta ja mallista.

```
var mongoose = require('mongoose')
var Schema = mongoose.Schema

var SomeModelSchema = new Schema({
  a_string: String,
  a_date: Date
})

var SomeModel = mongoose.model('SomeModel', SomeModelSchema )

module.exports = SomeModel
```

3.5 NHL API -ohjelmointirajapinta

SportsDataIO:n NHL API toimii siten, että rekisteröidyttyään palveluun, sieltä saadaan API-avain, jota täytyy käyttää aina kun palvelusta pyytää dataa. Datapyynnöt tapahtuvat HTTP GET -pyynnöillä ja vastaukset tulee JSON-muodossa, josta on helppo saada tarvittava data käyttöön.

Esimerkki curl-pyynnöstä ja vastauksesta NHL APIin:

```
curl –k https://fly.sportsdata.io/v3/nhl/scores/json/CurrentSea-
son?key=APIKEY
{"Season":2022,"StartYear":2021,"EndYear":2022,"Descrip-
tion":"2021–22","RegularSeasonStartDate":"2021–10–
12T00:00:00","PostSeasonStartDate":"2022–05–01T00:00:00","Sea-
sonType":"REG","ApiSeason":"2022REG"}
```

Esimerkki-pyynnöllä haimme kuluvan NHL-kauden tietoja ja NHL API vastasi JSON-muodossa.

3.6 Nginx-palvelin

Nginx on avoimen lähdekoodin palvelinohjelmisto, jota tämän projektin aikana käytetään www-palvelimena, samalla hyödyntäen sen käänteisproxy-ominaisuuksia. Nginx taipuu tarvittaessa lähes mihin tahansa, mitä palvelimelta haluaa, mutta tämän opinnäytetyön osalta keskityin vain muutamaan työssä tarvittuun ominaisuuteen: www-palvelin, websocket ja käänteisproxy, jotta opinnäytetyön laajuus pysyisi kohtuullisena. (Bali, Kavya 2021: Apache Vs NGINX – Which Is The Best Web Server for You?. Osoitteessa: https://serverguy.com/comparison/apache-vs-nginx/)

4 TOIMEENPANO

Tässä luvussa käydään läpi tarkemmin itse opinnäytetyön vaiheita. Toisin kuin edellisissä luvuissa, joissa edettiin frontendistä backendiin ja tietokantaan, pyritään tässä luvussa käymään asiat läpi käänteisessä järjestyksessä. En koe mielekkäänä käydä läpi kaikkea opinnäytetyöhön käytettyä koodia, joten siitä käydään läpi vain olennaisimmat osat. Linkki lähdekoodiin löytyy liitteistä. Ohjelmistojen tai niiden lisäosien asennuksia ei käsitellä erikseen, mutta mainitaan se, että ne on suurelta osin tehty node package managerin (npm) avulla.

4.1 Backend

Tässä luvussa käydään läpi, miten backendin keskeisimmät ominaisuudet toteutettiin. Aluksi esitellään backend-sovelluksen kansiostruktuuri ja kansioiden sisältö, joka oli seuraava:

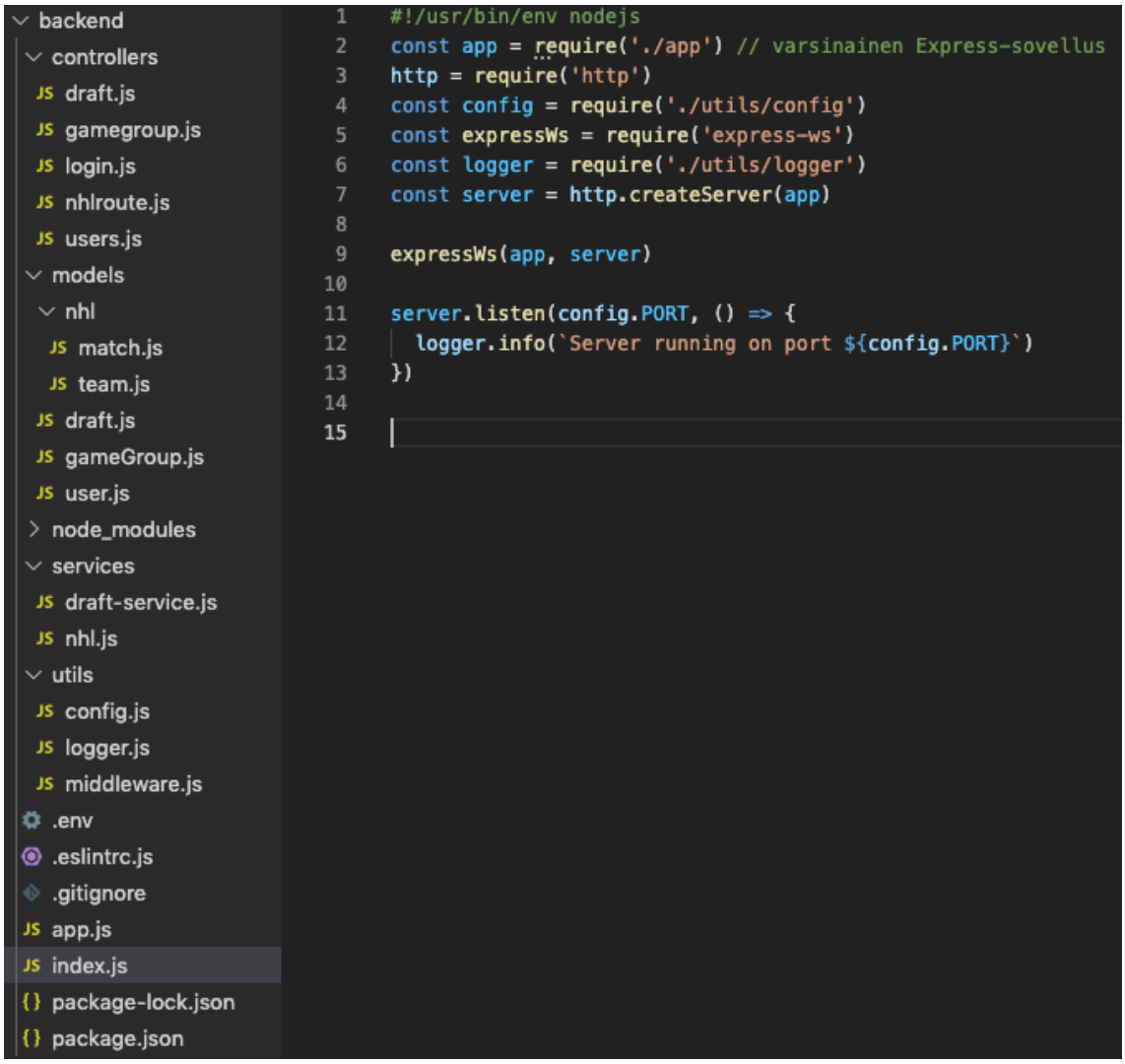
- controllers-kansio – sisältää reititykset frontendin tarpeisiin
- models-kansio – sisältää eri dokumenttimallit MongoDB:lle
- node_modules – sisältää kaikki sovellukseen asennetut kirjastot ja paketit
- services-kansio – palvelut, joita backend-sovellus käyttää
- utils-kansio – “Työkalut”, jotka siistii ja lyhentää koodia muualla ja nopeuttaa sovelluksen hallintaa.

Pohjahakemistossa löytyy seuraavat tiedostot:

- .env – sisältää globaaleja muuttujia, kuten salasanoja ja portteja, joita on turvallisempi ja helpompi käsitellä erillisessä tiedostossa
- .eslintrc.js – sisältää koodin tyyliasetukset, kuten sisennyksen merkkien määrän
- .gitignore – sisältää tiedot siitä mitä tiedostoja ei lähetetä GitHubiin – tässä tapauksessa node_modules -kansio ja .env-tiedosto
- app.js – Express-sovellus
- index.js – Sovelluksen käynnistystiedosto (kuva 2).

- package-lock.json ja package.json – sovelluksen asetukset, joissa määritellään mm. käytettävien kirjastojen versionumerot, käynnistyskomennot yms.

Index.js-tiedostossa (kuva 2) tuodaan sovellukseen sisään (require) tarvittavat kirjastot (http, express-ws) ja komponentit (app, config, logger), jotta sovellusta voidaan ajaa. Lopuksi itse backend-palvelin käynnistetään funktiolla server.listen, jolle annetaan attribuutteina config-tiedostossa määritetty portti ja funktio, joka tulostaa konsoliin viestin palvelimen onnistuneesta käynnistyksestä.



Kuva 2. Backendin kansiorakenne ja index.js-tiedosto.

4.1.1 Tietokannan luonti ja yhteys tietokantaan

MongoDB:n pilvipalveluun (Atlas) tietokannan luonti on verrattain helppoa sen Www-käyttöliittymän kautta. Palveluun luodaan projekti, jonka alle luodaan tietokanta (cluster). Tietokannan luomisessa kestää muutama minuutti ja tämän jälkeen käyttö voidaankin käytännössä aloittaa. Ensin kuitenkin tietokannalle määritettiin käyttäjä ja sille salasana. Tämän jälkeen käyttöliittymä tarjosi osoitteen, jolla pystyi yhdistämään tietokantaan NodeJS:ssä.

MongoDB:n tarjoama koodi:

```
mongodb+srv://kiakkoadmin:<password>@kiakkoterot.r3ehh.mongodb.net/myFirstDatabase?retryWrites=true&w=majority
```

Lisäksi tietokannalle määritettiin aluksi sallitut IP-osoitteet avoimeksi, jotta kehitystyö ja testaus onnistuisi mistä tahansa. Kun lopullinen backend-palvelin (Nginx) oli pystytetty, määritettiin sallittuihin IP-osoitteisiin vain kyseisen palvelimen IP-osoite.

Sovelluksessa yhteys luodaan koodin 4 mukaisesti (Huom! Erikseen asennettu mongoose-lisäosa, joka helpottaa MongoDB:n käyttöä NodeJS:n kanssa).

Koodi 4. Esimerkki mongoosen käytöstä.

```
const mongoose = require('mongoose')
const mongoOptions = {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  useFindAndModify: false,
  useCreateIndex: true
}
mongoose.connect(config.MONGODB_URI, mongoOptions)
  .then(() => {
    logger.info('connected to MongoDB')
  })
  .catch((error) => {
    logger.error('error connecting to MongoDB:', error.message)
  })
```

Kun yhdistäminen tietokantaan onnistuu, on tietokanta saavutettavissa kaikista sovelluksen alitiedostoista. MongoDB:n www-käyttöliittymästä saatu koodi on nyt määritetty

muuttujaan config.MONGODB_URI. Muuttuja mongoOptions sisältää asetuksia, jotka liittyvät MongoDB:n tiedonkäsittelyyn ja virheviesteihin.

4.1.2 Mallit

MongoDB käyttää dokumentin tallentamisessa hyödykseen ns. malleja (models), joita voitaisiin sanoa ennalta määritetyiksi objekteiksi. Määrittely tapahtuu skeemojen avulla (koodi 5), jotka sisältävät tiedon siitä, minkä nimisiä ja minkä tietotyypin muuttujia skeema pitää sisällään.

Koodi 5. Esimerkki sovelluksen Draft-mallista lyhennettynä.

```
const mongoose = require('mongoose')

const draftSchema = mongoose.Schema({
  gameGroup: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'GameGroup',
    required: true
  },
  status: { type: String, default: 'scheduled', required: true },
  startingTime: {
    type: Date,
    default: new Date('October 5, 2021 17:00:00'),
    required: true
  },
  teamsChosen: [
    {
      user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User'
      },
      team: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Team',
        required: true
      }
    }
  ],
```

```
  fullDraftOrder: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User'
    }
  ],
  round: { type: Number, default: 1 },
  totalRounds: Number,
})

draftSchema.plugin(require('mongoose-autopopulate'))

const Draft = mongoose.model('Draft', draftSchema)

module.exports = Draft
```

Tästä huomaa, että skeeman sisällä voidaan tehdä viittauksia muihin malleihin, avaimessa gameGroup viitataan GameGroup-nimiseen malliin tai teamsChosen-avaimelle on määritetty taulukko, joka sisältää objekteja, joiden sisällä on sekä User-, että Team-malli. Tämä helpottaa tiedonkäsittelyä huomattavasti.

Skeemojen tietueille voidaan määrittää myös oletusarvo, kuten avaimella status, se on ’scheduled’. Vaihtoehtoisesti voidaan tyytyä määrittämään pelkkä muuttujan tyyppi, kuten kohdassa totalRounds: Number, eli numeroarvo.

Lopuksi skeeman avulla luodaan malli Draft ja se mahdollistetaan importoitavaksi module.exports -komennolla.

Sovelluksessa näitä eri malleja on yhteensä 5 ja ne ovat seuraavat:

- Match – yksittäisen ottelun tiedot
- Team – yksittäisen joukkueen tiedot
- Draft – yksittäisen draftin tiedot, GameGroupin alla
- GameGroup – yksittäisen kimpan tiedot
- User – yksittäisen käyttäjän tiedot

4.1.3 Reititykset

Reititystiedostoissa (controllers) (koodi 6) määriteltiin eri reitit (osoitteet), joista tietoa voidaan hakea ja lähettää. Nämä kaikki reitit sijaitsevat luonnollisesti kiakkoterot.fi/ alla, esim. alla olevassa koodissa määritellään osoite kiakkoterot.fi/api/nhl vastaamaan HTTP GET-pyyntöihin lähettämällä vastauksena tietokannan kaikki joukkueet. HTTP GET-pyyntö osoitteeseen kiakkoterot.fi/api/nhl/matches palauttaa taasen tietokannasta löytyvät ottelut.

Koodi 6. nhlRoute.js

```
app.use('/api/nhl', nhlRouter)
const nhlRouter = require('express').Router()
const Match = require('../models/nhl/match')
const Team = require('../models/nhl/team')

nhlRouter.get('/', async (request, response) => {
  const teamData = await Team.find({})

  if (teamData) {
    response.status(200).send(teamData)
  } else {
    response.status(400).send()
  }
})

nhlRouter.get('/matches', async (request, response) => {
  const matchData = await Match.find({})

  if (matchData) {
    response.status(200).send(matchData)
  } else {
    response.status(400).send()
  }
})

module.exports = nhlRouter
```

Kaikki sovelluksen reitit ja selitys:

- GET /api/nhl – palauttaa kaikki tietokannan joukkueet
- GET /api/nhl/matches – palauttaa kaikki tietokannan ottelut
- GET /api/users – palauttaa käyttäjien ID:t
- POST /api/users – luo uuden käyttäjän
- POST /api/users/invite – kutsuu käyttäjän kimppaan
- GET /api/users/invitations – hakee käyttäjän kutsut
- GET /api/login – tarkistaa onko käyttäjän token voimassa
- POST /api/login – kirjautuu sisään ja palauttaa käyttäjän tiedot
- POST /api/gamegroup/create – luo uuden kimpan
- GET /api/gamegroup/:id – palauttaa kimpan tiedot id:n perusteella
- DELETE /api/gamegroup/:id – poistaa käyttäjän kimpan
- PUT /api/gamegroup/accept/:id – liittää käyttäjän kimppaan
- GET /api/gamegroup/draft/:id – palauttaa draftin tiedot id:n perusteella
- PUT /api/gamegroup/draft/:id – muokkaa kimppaa
- PUT /api/gamegroup/draft/:id/prePicks – muokkaa esitäytettyä joukkuelistaa
- WEBSOCKET /draft/:draftId/:clientId – reitti WebSocket-liikenteelle

Käyttäjähallinta ja kirjautuminen on toteutettu JSON Web Tokenia hyödyntämällä. Kaikki käyttäjädataa käyttävät reitit varmistavat tokenien avulla, että käyttäjä on oikealla asialla. Token on määritetty vanhentumaan tunnissa, jonka jälkeen käyttäjän on kirjauduttava uudelleen sisään.

JSON Web Token

JWT (JSON Web Token) on avoin standardi, jonka avulla voidaan jakaa turvallisesti dataa ohjelmistojen välillä. Jokainen JWT sisältää nimensä mukaisesti JSON-muotoista dataa, jolla voidaan todentaa käyttäjä. Yleisin JWT:n käyttöskenaario on käyttäjäkirjautuminen. (JWT.io: Introduction to JSON Web Tokens. Osoitteessa: https://jwt.io/introduction)

4.1.4 Palvelut

Sovellus sisältää 2 eri palvelua (services): draftille (draft-service.js) ja nhl-datalle (nhl.js). Molemmat käynnistetään (koodi 7) samalla, kun itse backend-sovellus käynnistetään. Nämä palvelut tarjoavat apufunktioita reiteille ja toisekseen ne ajavat aikataulutettuja funktioita itsekseen.

Koodi 7. app.js – palvelut käynnistetään.

```
const nhlService = require('./services/nhl')
const draftService = require('./services/draft-service')
nhlService.initialize()
draftService.initialize()
```

NHL-datan hakemisen aikataulutus on toteutettu käyttämällä JavaSciptiin sisäänrakennettua setTimeOut()-funktiota. Sarjataulukko päivitetään 2 tunnin välein kutsumalla funktioita updateTeamData() ja joukkueiden, sekä otteluiden data päivitetään 12 tunnin välein kutsumalla funktioita getTeams() ja updateGamesByDate() (koodi 8).

Koodi 8. services/nhl.js

```
const initialize = async () => {
  if (shouldUpdateTeamData) {
    shouldUpdateTeamData = false

    setTimeout(() => {
      shouldUpdateTeamData = true
    }, (12 * 60 * 60 * 1000)) // hour * minute * second * millisecond

    teams = await getTeams()
    updateGamesByDate()
  }

  setTimeout(() => {
    initialize()
  }, (120 * 60 * 1000)) // minute * second * millisecond

  updateTeamData()
}
```

Draftien alkamiset on toteutettu node-schedule-kirjastolla (koodi 9), joka mahdollistaa helpon tiettyyn ajankohtaan perustuvan funktion alkamisen.

Koodi 9. Lyhennetty esimerkki schedulerin käytöstä.

```
const scheduler = require('node-schedule')
let startingDate = new Date()
scheduler.scheduleJob(startingDate, async () => {
  logger.info('job started')
  ...
})
```

Kun backend-sovellus käynnistetään, node-schedule aikatauluttaa kaikki tietokannasta löytyvät draftit alkamaan sieltä löytyvän ajan perusteella. Uudet kimpat aikataulutetaan automaattisesti, sekä käyttäjän tekemät aikataulumuutokset.

Draft-service.js pitää sisällään suurimman osan logiikasta millä reaaliaikainen draft-tilaisuus toteutetaan WebSocket-protokollaa hyödyntäen. Draftin alkaessa se arpoo draftjärjestyksen, jonka jälkeen se ottaa vastaan joukkuevalintoja käyttäjiltä samalla tarkistaen, että on oikeasti kyseisen pelaajan vuoro. Kun joukkueet on valittu, se asettaa draftin tilan valmistuneeksi.

4.1.5 Työkalut

Backend-sovelluksen utils-kansiosta löytyy erilaisia työkaluja, jotka helpottaa ja yksinkertaistaa koodin suorittamista muissa tiedostoissa.

Config.js – Pitää sisällään globaaleja muuttujia, eli portit ja MongoDB-osoite

Logger.js – Aputyökalu, joka käytännössä korvaa backendin puolella console.login käytön. Syöttää konsoliin viestejä värikoodein sisältäen ajan (koodi 10).

Koodi 10. Logger.js esimerkki

```
const info = (...params) => {
  const timestamp = new Date().toLocaleString()
  const format = '\x1b[2m\x1b[36m%s\x1b[33m%s\x1b[0m\x1b[32m%s\x1b[0m'
  console.log(format, timestamp, ' - ', ...params)
}
```

Middleware.js – Käsittelee virheelliset pyynnöt ja antaa virheen perusteella virheviestin. Muokkaa tokenit helpommin käsiteltäväksi, vähentäen koodin toistamista (koodi 11).

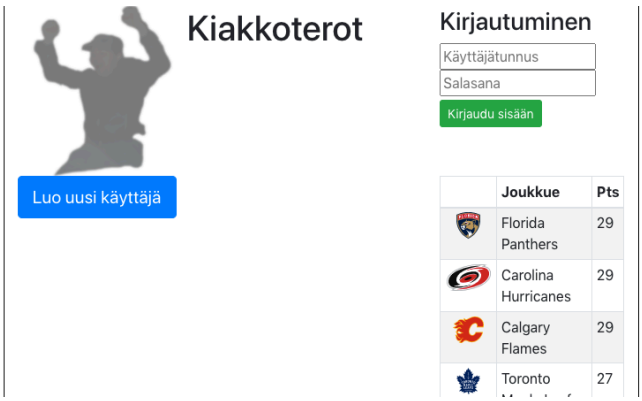
Koodi 11. Middleware.js – tokenin siirto suoraan request-objektiin.

```
const tokenExtractor = (request, response, next) => {
  const authorization = request.get('authorization')
  if (authorization && authorization.toLowerCase().startsWith('bearer')) {
    request.token = authorization.substring(7)
  }
  next()
}
```

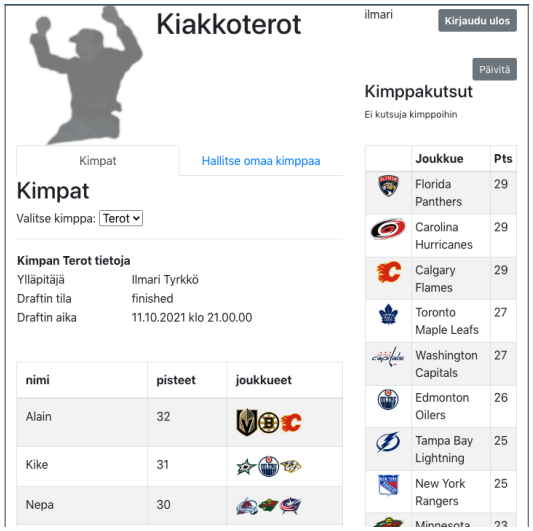
4.2 Frontend

Frontendissä eli palvelun käyttöliittymässä keskityttiin ensisijaisesti tekniseen toimivuuteen, joten aikataulun puitteissa käyttöliittymän visuaalisuus ja ulkoasu jäi melko yksinkertaiseksi. Ulkoasussa keskityttiinkin lähinnä sen toimivuuteen niin työpöytä-, kuin mobiililaitenäkymässä.

Ulkoasun (kuva 3, kuva 4) toteuttamisessa käytettiin hyväksi Bootstrap-kirjastoa, joka tarjoaa React-elementtejä, kuten Buttonit (napit) ja Tablet (taulukot), joiden käyttö ja muotoilu on helpompaa, kuin että kirjoittaisi niiden koodin alusta asti itse.



Kuva 3. Näkymä etusivusta, kun palveluun ei olla kirjauduttu sisään



Kuva 4. Näkymä, kun käyttäjä on kirjautunut sisään. Draft on suoritettu loppuun.

Frontendin pääkomponentin App.js (koodi 12) logiikka perustuu suurelta osalta lähinnä siihen, onko käyttäjä kirjautunut sisään. Mikäli käyttäjä on kirjautunut sisään, näytetään GameGroupHandler-komponentti ja jos ei, näytetään CreateUser-komponentti, jolla palveluun voi luoda käyttäjän.

Koodi 12. App.js:n JSX-osuus (kokonainen lähdekoodi GitHubissa).

```
    <div>
      {message}
      <Container style={{ border: '1px solid black' }}>
        <Row>
          <Col xs={4} md={8}>
            <img
              alt='alfonso'
              src={`${process.env.PUBLIC_URL}logo192.png`}
              style={{
                float: 'left',
                filter: 'blur(1px) contrast(5%)'
              }}
            />
            <h1>Kiakkoterot</h1>
          </Col>
          <Col xs={8} md={4}>
            <Login user={user} setUser={setUser}
              createMessage={createMessage} />
```

```
          {user ?
            <AcceptInvitation
              user={user}
              setUser={setUser}
              createMessage={createMessage}
            />
            : '' }
        </Col>
      </Row>
      <Row>
        <Col xs={12} md={8} style={{ marginBottom: '3em' }}>

          {user ?
            <GameGroupHandler
              user={user}
              setUser={setUser}
              createMessage={createMessage}
              teamData={teamData}
            />
            :
            <CreateUser createMessage={createMessage} />
          }

        </Col>
        <Col xs={12} md={4}>
          <Standings teamData={teamData} />
        </Col>
      </Row>
      <Row>
        <Col>
          &copy; 
          <a href='mailto:tyrkkoilmari@gmail.com'>
            Ilmari Tyrkkö
          </a> 2021
        </Col>
      </Row>

    </Container>
  </div>
```

Käyttäjälle tulevat viestit (väärä salasana, asetusten tallentaminen) näytetään App.js-koodin kohdassa {message}, jota hallitaan funktiolla createMessage(). Kyseinen funktio lähetetään kaikille alikomponenteille, jotta niillä voi tarvittaessa luoda viestejä käyttäjälle (kuva 5). Viestit näytetään ruudun ylälaidassa.

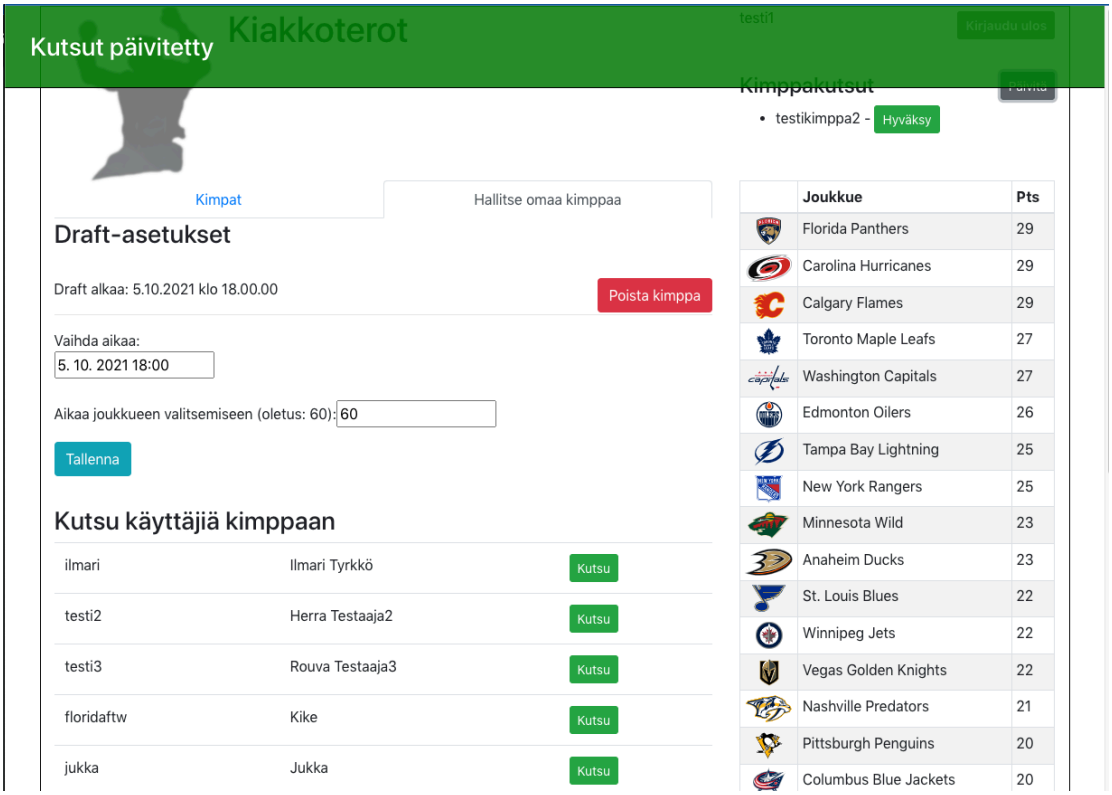
Kuva 5. Virheviesti sivuston ylälaidassa

Kaikki sivuston data, kuten sarjataulukko haetaan HTTP GET -pyynnöillä backendistä. Datan hakemiseen käytetään Axios-kirjastoa, jonka avulla pyyntöihin käytettävä koodin määrä lyhenee merkittävästi (koodi 13).

Koodi 13. Esimerkki Axios-pyynnöstä. Tässä pyydetään sarjataulukon data.

```
useEffect(async () => {
    axios.get(`${APIURL}/nhl/`)
      .then(res => setTeamData(res.data))
      .catch(e => console.log(e))
  }, [])
```

Kun käyttäjä on luonut kimpan, hän pääsee muokkaamaan varaustilaisuuden asetuksia (kuva 6), eli alkamisaikaa ja vuoron kestoa. Samasta näkymästä hän voi kutsua muita käyttäjiä kimppaansa. Kutsun saaneiden käyttäjien täytyy tämän jälkeen hyväksyä kutsu kimppaan liittyäkseen.



Kuva 6. Näkymä oman kimpan hallintapaneelista, kirjautunut käyttäjä on kutsuttu myös toiseen kimppaan (testikimppa2)

Joukkuevalinnat voidaan tehdä myös ennen draftin alkamista (kuva 7), jolloin backend valitsee parhaan listalla olevan vapaan joukkueen. Listassa joukkueita siirretään drag&drop-menetelmällä.

Tämä mahdollistaa sen, että käyttäjän ei ole pakko olla draft-tilaisuudessa läsnä, saadakseen itselleen mieluisat joukkueet.

Kuva 7. Joukkueiden valinta etukäteen

Draftin ollessa käynnissä käyttäjä näkee vuorojärjestyksen ja kuka on tällä hetkellä vuorossa (kuva 8). Kun on käyttäjän oma vuoro, käyttöliittymään ilmestyy lista joukkueista, jotka ovat vielä valitsematta. Valinta täytyy varmistaa erillisellä Valitse-napilla.

Muiden pelaajien tekemät joukkuevalinnat näkyvät draft-näkymässä reaaliajassa.

Kuva 8. Näkymä kun draft on käynnissä. Käyttäjä testi3 on valinnut joukkueen Philadelphia Flyers.

Koontiversio

Kun frontend-sovellus oli valmis, siitä tehtiin ns. koontiversio (build), ajamalla komento ’npm run build’ sen juurihakemistossa. Tämä komento luo uuden hakemiston ’build’ projektin alle, joka sisältää optimoidun version React-sovelluksesta. Tämän build-kansion sisältö siirretään sellaisenaan palvelimelle.

4.3 Nginx-palvelin

Kun sekä backendin ja frontendin tarvittavat tiedostot oli siirretty Nginx-palvelimelle, asennettiin palvelimelle PM2-ohjelmisto. PM2-ohjelmisto on prosessinhallintaohjelma, joka on suunniteltu erityisesti NodeJS-sovelluksille. PM2 mahdollistaa sen, että sovellus uudelleen käynnistetään, jos se sattuisi kaatumaan. Sekä frontend-, että backend-

sovellukset lisättiin PM2:n hallintaan ja niille määritettiin käyttöön palvelintietokoneen portit 3000 (frontend) ja 3001 (backend).

Tämän jälkeen täytyi itse Nginx-palvelin konfiguroida siten, että se osaa ohjata liikenteen seuraavasti:

- (www.)kiakkoterot.fi – frontend-sovellus
- api.kiakkoterot.fi – backend-sovellus
- ws.kiakkoterot.fi – backend-sovelluksen websocket-päätepiste

Frontend-pyynnöt portista 80 saatiin ohjattua suoraan käyttämään lokaalia tiedostopolkua, mutta backend-sovellus vaati käyttöönsä käytännössä 2 eri porttia: HTTP-liikenteelle ja WebSocket-liikenteelle. Tämä oli ehkä projektin suurin haaste ja tarvitsi paljon kokeiluja, mutta lopulta alla olevalla (koodi 13) konfiguraatiolla liikenne alkoi ohjautua oikein.

Koodi 14. Nginx-konfiguraatio.

```
server {
  root /var/www/kiakkoterot.fi/html;
  index index.html;

  server_name kiakkoterot.fi www.kiakkoterot.fi;

  location / {
    try_files $uri $uri/ /index.html;
  }
  listen 80;
}

server {
  server_name api.kiakkoterot.fi;

  location / {
    proxy_pass "http://localhost:3001" ;
  }
  listen 80;
}

map $http_upgrade $connection_upgrade {
  default upgrade;
```

```
  '' close;
}

upstream websocket {
  server localhost:8010;
}

server {
  server_name ws.kiakkoterot.fi;
  location / {
    proxy_pass "http://localhost:3001";
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection $connection_upgrade;
    proxy_set_header Host $host;
  }
}
```

5 PALVELUN KÄYTTÖÖNOTTO

Ennen palvelun lopullista käyttöönottoa palvelua testattiin kymmeniä kertoja omilla laitteilla, usealla eri selaimella ja useilla testitunnuksilla. Testaukset onnistuivat pienten vastoinkäymisten jälkeen hyvin, eli draft-valinnat kaikkien testikäyttäjien osalta onnistuivat, joten oli aika järjestää oikea draft.

Ensimmäinen yritys draftin järjestämiseksi oli 3. lokakuuta 2021 klo 20.00, jolloin paikalla oli 9 käyttäjää. Draft käynnistyi, ja kaikki näytti sujuvan hyvin, kunnes n. 10. kierroksen kohdalla draft-järjestys sekosi ja väärällä vuorolla oleva pelaaja pystyi valitsemaan toiselle käyttäjälle joukkueen. Draft keskeytettiin ja muutaman uudelleenyrityksen jälkeen päätettiin draft siirtää toiseen ajankohtaan.

Ennen uutta yritystä vuoron tarkistuksen logiikkaa tarkasteltiin ja kävi ilmi, että kun ensimmäinen valintakierros (9. vuoroa) oli päättynyt, palvelun olisi pitänyt kääntää valintajärjestys seuraavalle kierrokselle (ensimmäisellä kierroksella viimeisenä valinnut valitsee joukkueen ensimmäisenä toisella kierroksella), mutta näin ei tapahtunut. Syy jäi epäselväksi, sillä testeissä valintavuorojen järjestys toimi aivan kuten piti.

Valintavuorojen tarkistuksen logiikkaa muutettiin siten, että sen sijaan että varausvuorotaulukon (9 käyttäjää, 9 objektia) järjestys käännettiin joka valintakierrokselle, tehtiin koko varaustilaisuuden järjestys taulukkoon valmiiksi (9 käyttäjää, 27 objektia), kun varaustilaisuus käynnistyi.

Koodin korjauksen jälkeen, muutama päivä ensimmäisen yrityksen jälkeen draft saatiin lopulta onnistuneesti läpi.

6 POHDINTA

Opinnäytetyön tavoitteena oli luoda pelipalvelu, jossa NHL-liigan ottelutilastoja käyttämällä voi luoda kimppoja jäsenien kesken. Tässä päämäärässä lopulta onnistuttiinkin, joskin matkan aikana oli myös kompastuskiviä ja palveluun jäi vielä paljon kehitettävää.

Backend-sovelluksen logiikkaa ja pyyntöjen tarkistuksia olisi syytä kehittää entisestään, sillä niihin jäi heikkouksia datan oikeellisuuden tarkistukseen, ja pitemmällä aikavälillä tällaiset heikkoudet yleensä kostautuvat. Ulkoasuun olisi hyvä tuoda hieman väriä ja parantaa yleistä käytettävyyttä.

Vaikka kahden erillisen sovelluksen (frontend, backend) hallittavuus erikseen vaikutti hyvältä vaihtoehdolta, niin sitä se ei välttämättä ole. Opinnäytetyön aikana tutustuin mm. NextJS-viitekehykseen, joka on NodeJS-pohjainen ja mahdollistaa React-pohjaiset sivut. Kyseisellä viitekehyksellä palvelun voisi luoda yhtenä sovelluksena.

Tällä hetkellä on käynnissä pelin seurantavaihe, joka jatkuu aina NHL-liigan runkosarjan päättymiseen asti eli huhtikuuhun 2022. Tämän jälkeen onkin syytä tarkastella, onko peliä tarvetta kehittää esim. pelaajien tilastoihin liittyvällä lisäpelillä. Vaikka pelipalvelua ei enää kehitettäisi, sitä tullaan käyttämään ainakin tällaisenaan myös tulevina NHL-kausina.

LÄHTEET

Ahmed, Osama 2019: WebSockets Not In A Nutshell. Medium.com. Viitattu 23.11.2021. https://medium.com/@osama.scream2/websockets-in-a-nutshell-11dce96daf4e

Bali, Kavya 2021: Apache Vs NGINX – Which Is The Best Web Server for You?. Serverguy. Viitattu 23.11.2021. https://serverguy.com/comparison/apache-vs-nginx/

Fox, Johnny 2017: WebSocket + Node.js + Express – Step by step tutorial using Typescript. Medium.com. Viitattu 23.11.2021. https://medium.com/factory-mind/websocket-node-js-express-step-by-step-using-typescript-725114ad5fe4

Grujicic, Nikola 2021: Are React class components still needed in 2021. Frameless Grid. Viitattu 23.11.2021. https://www.framelessgrid.com/are-react-class-components-still-needed-in-2021/

Ivanovs, Alex 2020: Top 32 NPM Packages for Node.js Developers 2020. Colorlib. Viitattu 23.11.2021. https://colorlib.com/wp/npm-packages-node-js/

JWT.io: Introduction to JSON Web Tokens. JWT.io. Viitattu 7.12.2021. https://jwt.io/introduction

MongoDB 2021: Schemas. MongoDB. Viitattu 23.11.2021. https://docs.mongodb.com/realm/schemas/

Pandit, Nitin, 2021: What And Why React.js. Viitattu 6.12.2021. https://www.c-sharpcorner.com/article/what-and-why-reactjs/

SportsData.io 2021: Frequently Asked Questions. Sportsdata.io. Viitattu 6.12.2021. https://sportsdata.io/developers/faq

Statista 2021: Most used web frameworks among developers worldwide, as of 2021. Statista.com. Viitattu 23.11.2021. https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/

Tyrkkö, Ilmari 2021: Opinnäytetyön lähdekoodi, __TEROT. Github. Viitattu 23.11.2021. https://github.com/moukkari/__TEROT/