LAB University of Applied Sciences

Tapahtumapalvelun taustajärjestelmän toteuttaminen

LAB-AMMATTIKORKEAKOULU
Insinööri (AMK), Tieto- ja viestintätekniikka
Ohjelmistotekniikka
Syksy 2020
Mikael Petrow

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Petrow, Mikael	Opinnäytetyö, AMK	Syksy 2020
	Sivumäärä 32	

Työn nimi

Tapahtumapalvelun taustajärjestelmän toteuttaminen

Tutkinto

Insinööri (AMK)

Tiivistelmä

Opinnäytetyössä toteutettiin taustajärjestelmä sovelluskokonaisuudelle, joka koostui tapahtumien tiedonkulkua parantavasta mobiilisovelluksesta, sekä sen hallintaan tarkoitetusta web-sovelluksesta. Kokonaisuuden tilaajana oli itsenäisen sahateollisuuden etujärjestö Sahateollisuus RY.

Taustajärjestelmä toteutettiin Node.js web-teknologialla ja palvelin Express.js sovelluskehyksellä. Ohjelmointi suoritettiin JavaScript ohjelmointikielellä. Taustajärjestelmän ominaisuuksiin kuuluu mobiilisovelluksessa esitettävien tapahtumien tietojen haku tietokannasta, tämän tiedon luominen, muokkaaminen ja poistaminen web-sovelluksella, sekä molempien käyttöliittymien autentikointi. Tiedonkulku taustajärjestelmän ja käyttöliittymien välillä tehtiin REST-arkkitehtuurimallin mukaisesti HTTP-pyynnöillä ja vastauksilla käyttäen Expressin reititintä. Taustajärjestelmän tietovarastona käytettiin MongoDB tietokantaa. Tietokannan tietojen mallintaminen ja validointi sekä tietokantakyselyt tehtiin käyttämällä NPM pakettienhallintajärjestelmän tarjoamaa mongoose moduulia. Autentikointi ja istunnon rajallisuus toteutettiin käyttämällä json-webtoken moduulin tarjoamia tunnuksia ja salasanojen salauksessa käytettiin bcrypt moduulia.

Työn tuloksena toteutettiin asiakasvaatimusten mukainen taustajärjestelmä mahdollistamaan sovelluskokonaisuuden käyttöliittymien toiminnallisuutta tiedon luomisen, hakemisen, päivittämisen ja poistamisen osalta, sekä autentikointi molemmille käyttöliittymille. Taustajärjestelmästä löytyy parannettavaa, mutta toteutusta voidaan sanoa onnistuneeksi, koska asiakasvaatimukset täytettiin ja asiakaspalaute on ollut positiivista.

Asiasanat

Taustajärjestelmä, Node.js, REST-arkkitehtuuri, ohjelmistokehitys, JavaScript

Abstract

Author(s)	Type of Publication	Published
Petrow, Mikael	Bachelor's thesis	Autumn 2020
	Number of Pages	
	32	

Title of Publication

Implementation of Backend for Event Service

Name of Degree

Bachelor of Engineering

Abstract

The goal of this thesis was to implement a backend for an event service. The service consists of a mobile application designed to improve accessibility of information in events and a web application used as an administrator panel for managing the data accessed in the mobile application. The project was commissioned by Sahateollisuus RY, an interest group of sawmill industry.

The backend was implemented using Node.js web technology and its server with Express.js. Code for the backend was written with JavaScript programming language. Features of the backend include creation, reading, updating and deletion of data and authentication for both frontends. Communication between the backend and the frontends was implemented according to REST architectural constraints using HTTP requests and responses and Express router. Modeling for the data stored in the backends database was done with mongoose module provided by NPM package manager. The database used was MongoDB. Authentication was implemented with jsonwebtoken module to log in and limit session duration and bcrypt module to hash passwords.

As a result, a backend was implemented for the service including features for data management and authentication as client requested. What comes to the backend there is room for improvement in many fields, but the implementation can be regarded as successful as requirements were met and feedback from the client was positive.

Keywords

Backend, Node.js, REST architecture, software development, JavaScript

Sisällys

Sanasto

REST	Representational State Transfer. Arkkitehtuurimalli
HTTP-protokolla	Hypertext Transfer Protocol. Hypertekstin siirtoprotokolla
Node.js	JavaScriptin ajoympäristö
JavaScript	Ohjelmointikieli pääasiassa web-ympäristöön
Express	Node.js sovelluskehys
Sovelluskehys	Rungon muodostava ohjelmistotuote
URI	Uniform Resource Identifier. Merkkijono, joka kertoo tiedon sijainnin
CORS	Cross-Origin Resource Sharing. Mekanismi palvelimien resurssien lataamisen oikeuksien hallitsemiseen
NPM	Node Package Manager. Ohjelmistorekisteri ja pakettienhallintajärjestelmä
NoSQL	Not only SQL. Käsite kuvaamaan tietokantoja, jotka poikkeavat perinteisestä relaatiomallista
MongoDB	Dokumenttipohjainen tietokanta
JSON / BSON	JavaScript Object Notation / Binääri JSON. Tiedostomuotoja
Token	Tunnuksena käytettävä merkkijono

1 Johdanto

Konferenssien järjestäminen on yleinen tapa tuoda yhteen saman alan kollegoita ja yrityksiä esimerkiksi kouluttautumaan ja keskustelemaan liiketoiminnasta. Tämän kaltaisilla tapahtumilla on aikataulu, johon sisältyy monenlaista ohjelmaa kuten luentoja puhujilta ja paneelikeskusteluja. Sijaintina tapahtumille on useimmiten konferenssikeskus tai vastaava tapahtumapaikka. Konferenssien järjestäjänä toimii yleensä jokin yhdistys tai yritys.

Sahateollisuus RY on vuonna 1945 perustettu itsenäisen sahateollisuuden yritysten etujärjestö, jolla on yli 30 jäsenyhtiötä. Järjestö valvoo jäsenyhtiöidensä etuja edistämällä niiden liiketoimintaa. Sahateollisuus RY:n keskeisinä tavoitteina on, että itsenäinen sahateollisuus tunnetaan Suomessa kansantaloudellisesti tärkeänä kestävän kehityksen toimialana sekä kansainvälisesti luotettavana yhteistyökumppanina. Yhdistys tiedottaa liiketoimintaympäristön muutoksista alan yrityksille tukien näitä. Järjestö myös järjestää Suomessa konferenssityyppisiä tapahtumia sahateollisuuden yrityksille muutamia kertoja vuodessa. (Sahateollisuus RY 2020.)

Tässä opinnäytetyössä luodaan Sahateollisuus RY:n SawmillEvents tapahtumapalvelun sovelluskokonaisuuden taustajärjestelmä. Työn tavoitteena on kehittää yhteinen taustajärjestelmä mobiilisovellukselle ja web-sovellukselle valittuja teknologioita käyttäen asiakasvaatimusten pohjalta.

Sahateollisuus RY tarvitsi käyttöönsä mobiilisovelluksen, jolla parannettaisiin yhdistyksen järjestämien tapahtumien tiedonkulkua. Sovelluksen tarkoitus oli olla jokaisen tapahtumaan osallistuvan henkilön ladattavissa ja tarjota helposti saatavilla oleva ja helppokäyttöinen tietopaketti tapahtumasta. Sovellus tarjoaa tietoa tapahtumasta kuten sen yleiset tiedot, aikataulun, kartan, esiintyvät puhujat, listan muista osallistujista sekä ruokailumahdollisuudet. Tämä mobiilisovellus tarvitsee myös jonkin tavan muokata sen sisältöä ja tähän tarkoitukseen tarvittiin järjestelmänhallintaan tarkoitettu web-sovellus. Toimiakseen nämä kaksi sovellusta tarvitsevat taustajärjestelmän kommunikoidakseen toistensa kanssa ja varastoidakseen tietoa.

Opinnäytetyö on kaksiosainen. Ensimmäisessä osuudessa käydään läpi sovelluksen toimintaympäristöä, asiakasvaatimuksia, taustajärjestelmän kehityksessä käytettyjä teknologioita ja niiden teoriaa. Toisessa osassa kuvataan käytännössä SawmillEvents sovelluksen taustajärjestelmän kehitystä ensimmäisessä osuudessa käsiteltyjen teknologioiden pohjalta.

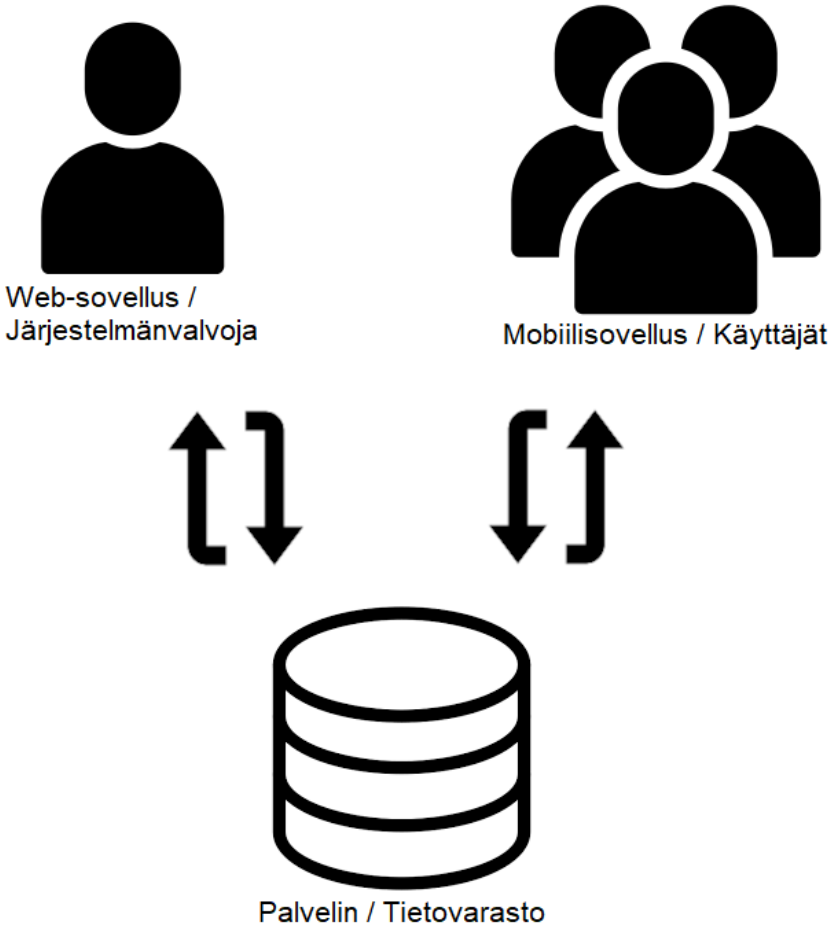
2 Toimintaympäristö

2.1 Sovelluksen toimintaympäristö

Sovelluskokonaisuus on kolmiosainen. Se koostuu mobiilisovelluksesta, web-sovelluksesta ja taustajärjestelmästä. Mobiilisovellusta käyttävät tapahtumiin osallistuvat henkilöt. Sovellukseen kirjaudutaan sisään, jonka jälkeen näytetään lista tapahtumista, joihin käyttäjä on ilmoittautunut. Mobiililaitteen sovellus kommunikoi REST-rajapintaa käyttäen erillisen palvelimen kanssa, jossa sijaitsevasta tietovarastosta se hakee kaiken käyttöliittymässä näytettävän muuttuvan datan. Datan hakeminen tapahtuu mobiilisovelluksen tekemillä HTTP-pyynnöillä, joihin palvelin vastaa palauttamalla mobiilisovellukselle pyynnössä määritellyn datan.

Sovelluskokonaisuuden web-sovellus on tarkoitettu järjestelmänhallintaan. Tapahtuman järjestäjä käyttää sitä omalta tietokoneeltaan kirjauduttuaan ensin järjestelmänhallitsijan tunnuksillaan. Web-sovelluksessa tulee olla mahdollista muokata mobiilisovelluksessa näytettävää dataa kategorioittain eri näkymissä. Tietojen muokkauksen jälkeen ne tallennetaan sovelluskokonaisuuden taustajärjestelmän tietovarastoon. Myös web-sovelluksen ja taustajärjestelmän palvelimen välinen kommunikointi tapahtuu REST:n periaatteiden mukaisesti HTTP-pyyntöjen ja vastausten avulla.

Sovelluksen taustajärjestelmässä on käynnissä ympärivuorokautisesti palvelin Node.js ympäristössä kuunnellen sille lähetettäviä pyyntöjä ja vastaten niihin pyynnön määrittelyn perusteella. Datan vastaanottamisen ja jakamisen lisäksi taustajärjestelmä huolehtii datan varastoimisesta tietovarastoon. Sovelluksen toimintaympäristö on havainnollistettu seuraavassa kuvassa (Kuva 1).



Kuva 1. Mobiilisovelluksen toimintaympäristö havainnollistettuna

2.2 Taustajärjestelmän vaatimukset

Seuraavaksi käydään läpi vaaditut ominaisuudet kummankin käyttöliittymän osalta käyttöliittymän näkymien kautta, joista johdetaan päätelmät siitä, mitä taustajärjestelmältä vaaditaan. Tämä suoritetaan siitä syystä, että asiakkaan vaatimukset sovelluksesta rajoittuvat käyttöliittymä puoleen.

2.2.1 Mobiilisovelluksen osalta tarvittavat ominaisuudet

Mobiilisovelluksessa osalta tarvittavat ominaisuudet alkavat kirjautumisesta. Sovelluksen sisältöön on tarkoitus päästä käsiksi vain kutsutut henkilöt, joten sovelluksen avauduttua ensimmäisessä näkymässä vaaditaan syötettäväksi ilmoittautumisen yhteydessä käytetty sähköposti. Sähköpostin syöttämisen jälkeen listataan tapahtumat, joiden osallistujalistaan kyseinen sähköposti sisältyy ja näihin edelleen sisään pääseminen vaatii kullekin tapahtumalle ominaisen salasanan. Taustajärjestelmältä vaaditaan siis osallistujan tietojen sekä tapahtumien salasanojen varastoimista. Lisäksi vaaditaan logiikka, jolla

kummassakin autentikoinnin vaiheessa tarkistetaan, onko syötetyt tiedot oikeat ja täten varmistutaan siitä, onko käyttäjä oikeutettu jatkamaan seuraavaan näkymään. Käyttäjän pitää pysyä kirjautuneena ennalta määritellyn ajan. Turvallisuussyistä taustajärjestelmän pitäisi kyetä testaamaan käyttäjän kirjautumisen voimassaolo joka kerta kun mobiilisovellus kommunikoi taustajärjestelmän kanssa. Aika, jonka käyttäjä pysyy sisään kirjautuneena, pitäisi olla rajallinen, jotta vältytään turhilta turvallisuusriskeiltä. Tästä syystä session rajallisuuteen tarvitaan myös logiikka. Niin tapahtumien listaukseen kuin niihin sisälle navigoitaessa näytettävään dataan tarvitaan taustajärjestelmältä tapa toimittaa oikeat tiedot käyttöliittymälle.

2.2.2 Web-sovelluksen osalta tarvittavat ominaisuudet

Järjestelmän hallintaan käytettävän web-sovelluksen ominaisuuksiin kuuluu myös sisään kirjautuminen ja kunkin tapahtuman datan näyttäminen, joten niiltä osin taustajärjestelmän vaatimukset ovat samat kuin mobiilisovelluksen kannalta. Näiden lisäksi web-sovelluksesta käsin muokataan tapahtumien dataa, joten taustajärjestelmän täytyy kyetä ottamaan vastaan muokatut tiedot ja tallentamaan päivitetty data. Varastoitava data on pääasiassa merkkijonoja ja numeroita, mutta tapahtumien tiedoissa on myös mukana kuvia, joita taustajärjestelmän on pystyttävä tallentamaan.

2.3 Valitut teknologiat

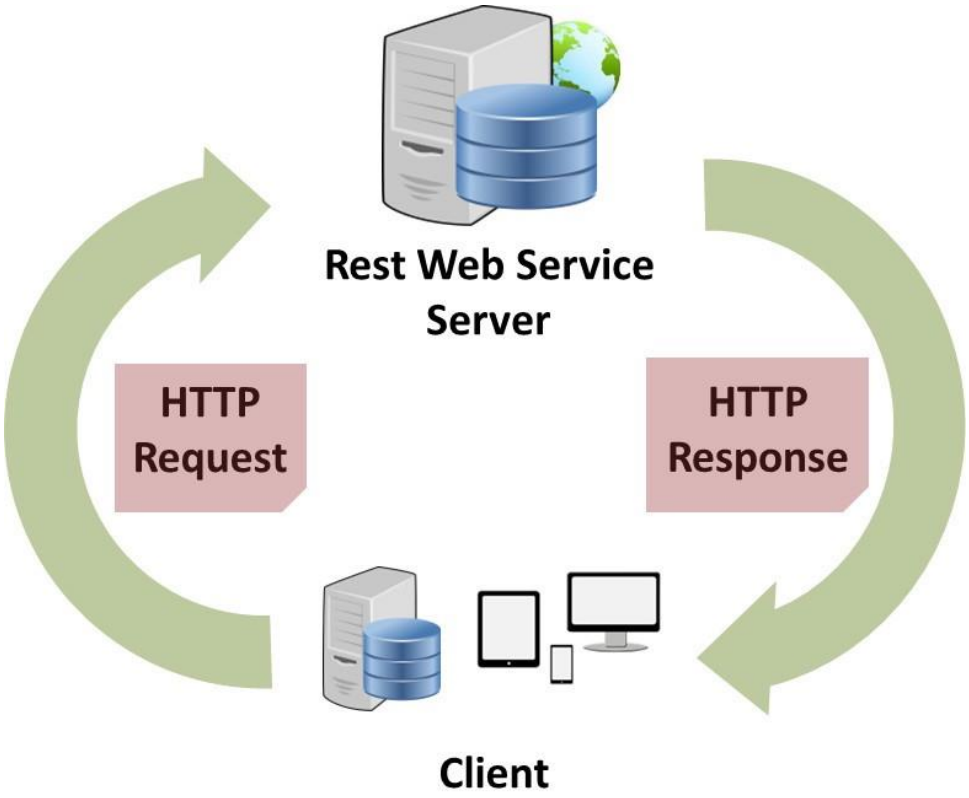
Taustajärjestelmän ohjelmoimisessa käytettiin JavaScript ohjelmointikieltä. Ratkaisuksi JavaScript koodin suorittamiseksi palvelinympäristössä valittiin Node.js JavaScript ajoympäristö. Sovellusten ja taustajärjestelmän välinen keskustelu toteutettiin REST-ohjelmistoarkkitehtuurin periaatteiden mukaisesti. Node.js ympäristössä käynnissä olevan palvelimen toteuttamiseen käytettiin Express sovelluskehystä. Datan varastoimiseen käytettiin MongoDB tietokantaa.

3 Teknologiat

3.1 REST-arkkitehtuuri

3.1.1 Yleistä

REST lyhenne tulee sanoista “Representational State Transfer” ja se on palvelujen kommunikointiin tarvittavien rajapintojen toteuttamiseen tarkoitettu arkkitehtuurimalli. REST tarjoaa normeja verkon tietokonejärjestelmien välille. REST:ssä kahden palvelun välinen tiedonsiirto pohjautuu HTTP-protokollaan ja sen ominaisuuksiin. REST-yhteensopiville järjestelmille, eli niin sanotuille RESTful järjestelmille on ominaista niiden tilattomuus ja se, kuinka asiakas- ja palvelinpuoli erotellaan niissä. (restfulapi.net 2020.) Alla on kuva (Kuva 2) havainnollistamassa REST-arkkitehtuurin toimintaa.



Kuva 2. REST arkkitehtuuri havainnollistettuna (ÖZLÜ 2020)

3.1.2 Asiakas-palvelin-malli

REST-arkkitehtuurimallissa on keskeisenä osana asiakas-palvelin-malli, jossa kaksi palvelua, asiakas ja palvelin, keskustelevat keskenään. Asiakas lähettää pyynnön, palvelin

käsittelee sen ja lähettää vastauksen. Palvelimen ja asiakkaan toteutukset tapahtuvat itsenäisesti, mistä seuraa, että molemman osapuolen koodia pystytään muokkaamaan ilman, että ne vaikuttaisivat toistensa toimintaan. Niin kauan kuin asiakas ja palvelin ovat tietoisia siitä muodosta, missä niiden välinen viestintä tapahtuu, voidaan ne pitää erillisinä. Tämä erillisyys mahdollistaa komponenttien joustavuuden ja skaalautuvuuden, sekä antaa niille mahdollisuuden kehittyä itsenäisesti. (restfulapi.net 2020.)

3.1.3 Tilattomuus

REST:n ominaisuuksiin kuuluu tilattomuus. Tilattomuudessa on kyse siitä, että asiakas ja palvelin eivät tiedä toistensa tilasta. Tämä aiheuttaa sen, että palvelimen pitäisi pystyä ymmärtämään saamansa viestit tietämättä edellisistä viesteistä. Palvelujen välisessä kommunikaatiossa tapahtuvat erilliset pyynnöt eivät tiedä toisistaan ja kaikki yhteen pyyntöön liittyvä tieto siirretään jokaisella pyynnöllä. (restfulapi.net.) Palvelin unohtaa kyselyn heti, kun se on suoritettu loppuun, mutta välimuistia voidaan toki käyttää, jotta turhalta toistamiselta vältytään.

3.1.4 Kerrostettu järjestelmä

REST:ssä asiakas sovellus ei tiedä, tai sillä ei ole merkitystä sovelluksen toiminnalle, onko se suorassa yhteydessä pääpalvelimeen, vai johonkin välittäjään matkan varrella (resfulapi.net). Tällaiset välityspalvelimet voivat tarjota esimerkiksi tason välimuistille tai autentikoinnille. Välityspalvelimen ei pitäisi vaikuttaa pyyntöön tai vastaukseen. Asiakas on tietämätön olemassa olevien välityspalvelimien määrästä sen ja pääpalvelimen välissä.

3.1.5 HTTP-protokolla ja sen metodit

HTTP-protokolla on sovellustason protokolla, jota käytetään tiedonsiirtoon tietoverkoissa. HTTP-protokollassa tieto liikkuu pyynnöillä ja vastauksilla asiakas-palvelin-mallin mukaisesti. REST:ssä pyynnöt kahden palvelun välillä tapahtuvat HTTP-protokollalla. Käytettävissä ovat HTTP-protokollan lukuisat metodit (Taulukko 1). Näillä metodeilla ja erilaisilla URI:lla kuvataan, millaisesta pyynnöstä on kysymys. (MDN web docs 2020.)

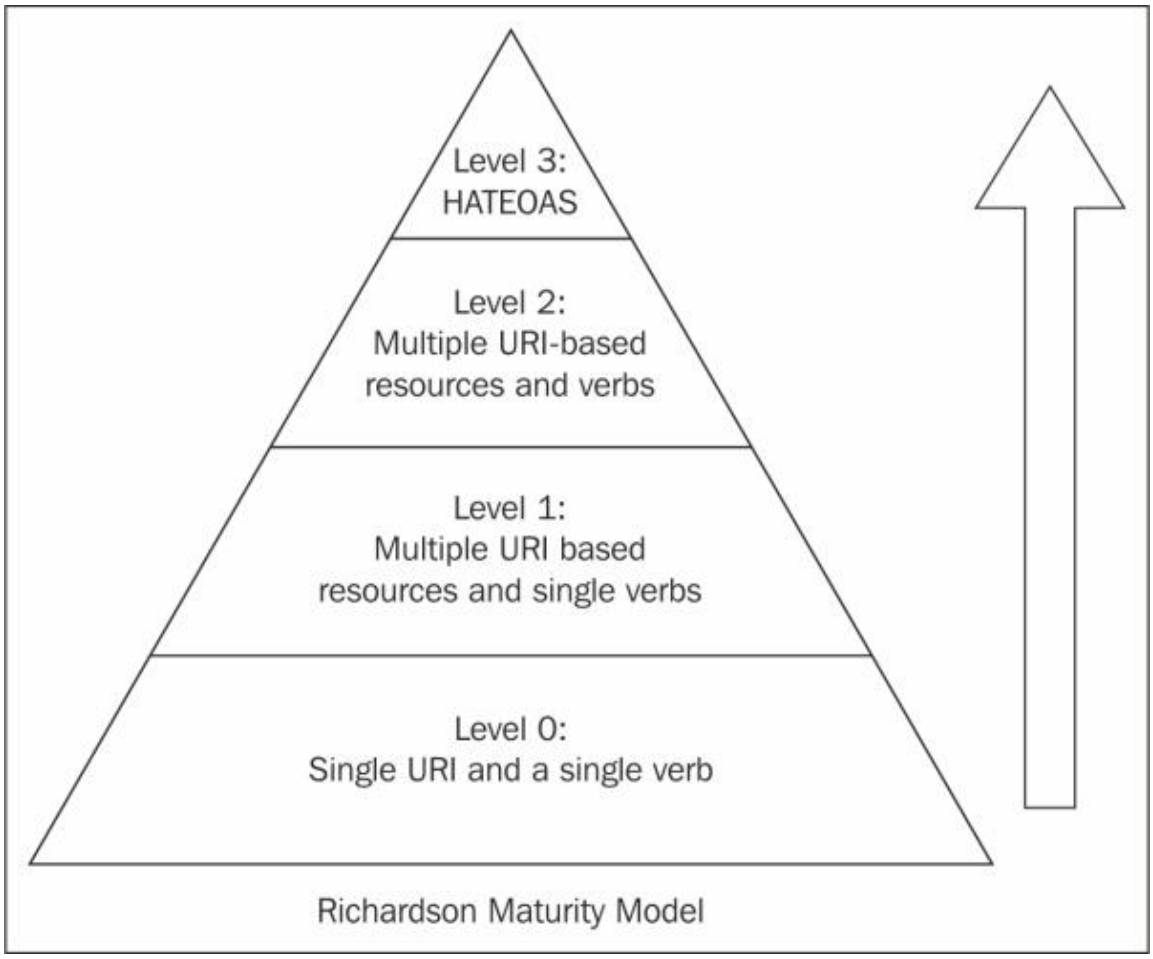
Metodi	Kuvaus
GET	GET-metodilla pyydetään tietynlaista resurssia kohteesta. GET-tyyppisien pyyntöjen pitäisi ainoastaan vastaanottaa dataa.

HEAD	HEAD-metodi on muuten samanlainen kuin GET-metodi, mutta HEAD-tyyppiset pyynnöt odottavat vastausta ilman vastauksen runkoa.
POST	POST-metodia käytetään lähettämään resurssi määriteltyyn määränpäähän.
PUT	PUT-metodi korvaa jo olemassa olevaa dataa mukanaan lähetetyllä datalla.
DELETE	DELETE-metodia käytetään poistamaan resurssi.
CONNECT	CONNECT-metodi perustaa tunnelin palvelimelle, joka identifioidaan kohteen resurssin mukaan.
OPTIONS	OPTIONS-metodia käytetään kuvailemaan kommunikaatioasetukset kohde resurssille.
TRACE	TRACE-metodi suorittaa viestisilmukantestin kohderesurssiin vievän polun varrella.
PATCH	PATCH-metodia käytetään muokkaamaan kohde resurssia osittaisesti.

Taulukko 1. HTTP-protokollan metodit

3.1.6 REST kypsyysmalli

Richardsonin kypsyysmalli on tapa antaa ohjelmointirajapinnalle luokka REST-arkkitehtuurin rajoituksien mukaan (Kuva 3). Mitä parempi ohjelmointirajapinta on näiden rajoitusten seuraamisessa, sitä korkeampi sen luokka on mallissa. Malli sisältää neljä tasoa nollasta kolmeen, jossa kolmas taso kuvastaa todellista RESTful ohjelmointirajapintaa. (restfulapi.net 2020.)



Kuva 3. Richardsonin kypsyys malli (restfulapi.net)

Taso nollaa kutsutaan nimellä Swamp of POX (Plain Old XML) ja siinä käytetään protokollaa, kuten esimerkiksi HTTP:tä, kuin kuljetukseen tarkoitettua protokollaa. Pyynnöt ja vastaukset tunneloidaan protokollan läpi käyttämättä sitä ilmaisemaan sovelluksen tilaa. URI:a käytetään yhtenä ja ainoana sisääntulopisteenä ja vain yksi protokollan metodi kuten POST on käytössä.

Taso yksi nimeltään Resources toteutuu tapauksessa, jossa ohjelmointirajapinta tunnistaa erilaiset resurssit. Tasossa on käytössä useampi URI, jossa eri resurssit ovat saatavilla eri URI:lla. Käytössä on edelleen vain yksi metodi kuten tasossa nolla.

Taso kaksi, eli HTTP verbs, osoittaa, että ohjelmointirajapinnan tulisi käyttää protokollan ominaisuuksia skaalautuvuuden ja epäonnistumisien hoitamiseksi. Tasossa kaksi tulisi käyttää protokollan eri metodeja niiden kuvausten mukaisissa asiayhteyksissä kuten POST-metodia kun viedään resurssia tai GET-metodia, kun haetaan resurssia. Vastauksissa tulisi myös käyttää monia vastaus koodeja ja asianmukaisissa tilanteissa.

Tasossa kolme nimeltään Hypermedia controls, otetaan käyttöön HATEOAS (Hypertext As The Engine Of Application State). Tämä REST-arkkitehtuurin komponentti erottaa sen

muista sovellusarkkitehtuureista. HATEOAS:ia hyödyntämällä asiakas on vuorovaikutuksessa verkkosovelluksen kanssa, jonka palvelin tarjoaa tietoa dynaamisesti hypermedian kautta. (restfulapi.net.) REST-asiakas tarvitsee hypermedian ymmärtämisen lisäksi vain vähän tai ei ollenkaan ennakkotietoa siitä, kuinka toimia palvelimen kanssa.

3.1.7 CORS

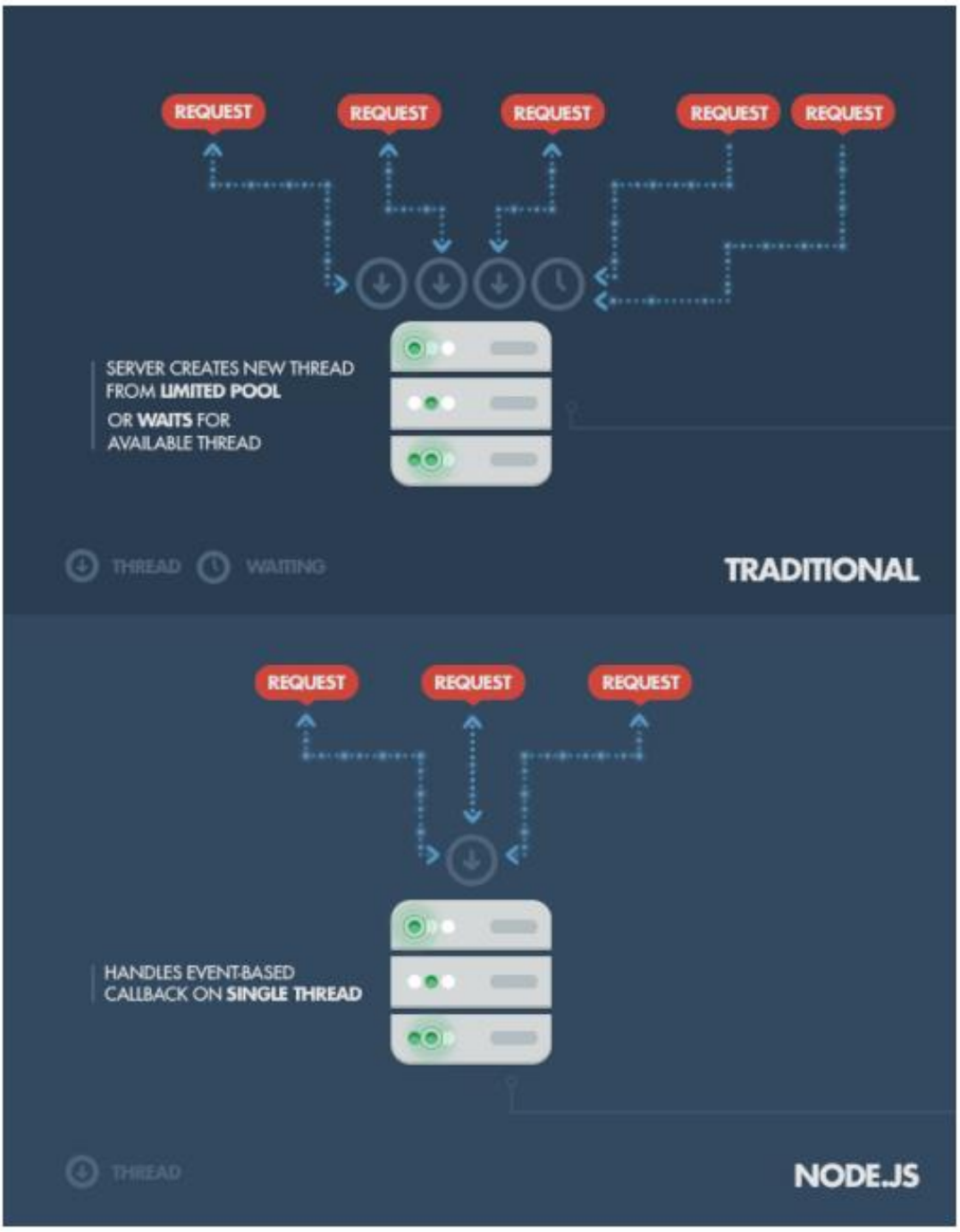
CORS on lyhenne sanoista Cross-Origin Resource Sharing. Se on mekanismi, joka käyttää HTTP-pyynnöissä lisäotsikoita, jotka mahdollistavat yhdestä lähteestä haetun sovelluksen päästä käsiksi toisella lähteellä olevan sovelluksen resursseihin. Sovellus suorittaa cross-origin HTTP-pyynnön tilanteessa, jossa se pyytää resurssia jostakin omastaan poikkeavasta alkuperästä, kuten esimerkiksi verkkotunnuksesta, protokollasta tai portista. Tämä tarkoittaa siis tilanteita, joissa URI eroaa joiltakin osin. CORS:n käyttö on yleistä web-sovelluksissa, koska selaimet eivät salli HTTP-pyyntöjä, jotka saavat alkuperänsä koodista, jos osoite, protokolla tai portti on alkuperästä eroava (MDN web docs 2020).

3.2 NODE.js

3.2.1 Yleistä

Node.js on web-teknologia, joka mahdollistaa JavaScriptin koodin suorittamisen palvelimella. Node.js:n käyttö on kasvanut palvelimien ohjelmoinnissa ja on tällä hetkellä toinen suosituimmista teknologioista palvelinpuolen kehityksessä PHP:n ohella (Patel 2019).

Node.js mahdollistaa palvelimien luonnin käyttämällä JavaScript ohjelmointikieltä ja kattavaa moduulikokoelmaa, joka mahdollistaa monimuotoisia toiminnallisuuksia. Node.js tukee natiivisti vain JavaScript ohjelmointikieltä, mutta JavaScriptiin kääntyviä kieliä kuten TypeScriptiä on mahdollista käyttää. Node.js:n pääkäyttötarkoitus on verkkosovelluksien rakentaminen, esimerkiksi web-palvelimet ovat tämänkaltaisia sovelluksia. Keskeyttämättömän I/O järjestelmän ansiosta Node.js:n ominaisuuksiin kuuluu se, että ohjelma ei pysähdy odottamaan levy- tai verkko-operaatioiden ajaksi niiden valmistumista vaan jatkaa muuta suorittamista. Kun operaatio valmistuu, ohjelma käsittelee siitä seuraavan tapahtuman. Yksikään suoritettava operaatio ei siis estä toisen suorittamista. Tämä on yksi suurimmista eroista Node.js:ssä verrattuna esimerkiksi toiseen yleiseen teknologiaan Apache palvelinohjelmaan, jota PHP käyttää. Node.js:ssä vain yhtä säiettä käytetään tukemaan suurta määrää yhteyksiä poiketen perinteisestä verkkopalvelutekniikasta. (Node.js 2020.) Alla on kuvassa (Kuva 4) havainnollistettuna eroavaisuus Node.js:n ja perinteisen verkkopalvelutekniikan välillä.

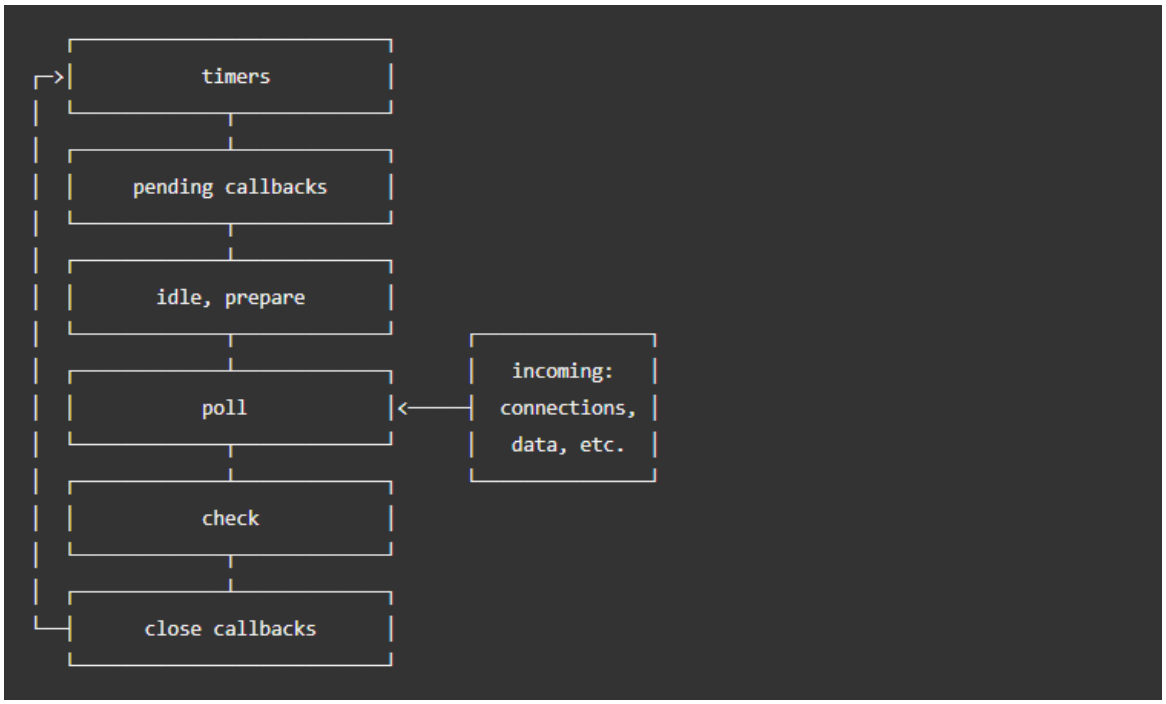


Kuva 4. Node.js:n ja perinteisen verkkopalvelutekniikan erot havainnollistettuna (Capan 2013)

3.2.2 Node.js tapahtumasilmukka

Tapahtumasilmukka mahdollistaa Node.js:n esteettömät operaatiot siitä huolimatta, että JavaScript on yksisäikeinen. Tämä tapahtuu siirtämällä operaatioita järjestelmän ytimelle

aina kun se on mahdollista. Modernien ytimien sisältäessä useampia säikeitä niiden on mahdollista suorittaa monia operaatioita taustalla, ja kun tällainen operaatio on suoritettu, ydin kertoo siitä Node.js:lle, jotta operaatioon kuuluva takaisinkutsu voidaan laittaa jonoon. Alla kuvassa (Kuva 5) on havainnollistettuna Node.js:n tapahtumasilmukan operaatioiden järjestys.



Kuva 5. Node.js tapahtumasilmukan operaatioiden järjestys (Node.js 2020)

Node.js:n käynnistyessä sen mukana käynnistyy myös tapahtumasilmukka, jota aletaan prosessoimaan yllä olevassa kuvassa näytetyssä järjestyksessä. Jokainen vaihe sisältää FIFO (First In First Out) jonon takaisinkutsuja. Tapahtumasilmukan jokaisessa vaiheessa suoritetaan kyseisen vaiheen kaikki operaatiot, jonka jälkeen suoritetaan jonon takaisinkutsut, kunnes niitä ei enää ole tai saavutetaan maksimimäärä takaisinkutsuja. Tämän jälkeen hypätään seuraavaan vaiheeseen, jossa sama toistuu.

Eri vaiheet tekevät seuraavanlaisia asioita. Timers vaiheessa suoritetaan ajastimien takaisinkutsut, jotka ovat peräisin setTimeout() ja setInterval() funktioista. Pending callbacks vaiheessa suoritetaan I/O takaisinkutsut, jotka lykätään seuraavalle silmukan iteraatiolle. Idle, prepare vaihetta käytetään vain sisäisesti. Poll vaihe ottaa vastaan uusia I/O tapahtumia, ja suorittaa I/O liitännäisiä takaisinkutsuja. Check suorittaa setImmediate() funktioiden takaisinkutsut. Close callbacks vaiheeseen kuuluu joitakin sulkemisien yhteydessä kutsuttavia takaisinkutsuja kuten socket.on(‘close’, …). Jokaisen tapahtumasyklin välissä

Node.js tarkastaa, onko se odottamassa asynkronisia I/O tai ajastimia ja sammuu jos yhtäkään ei ole (Node.js 2020).

3.2.3 Alusta-arkkitehtuuri

Node.js sovellus perustuu tapahtumapohjaiseen ohjelmointiin. Tämä mahdollistaa nopeiden web-palveluiden luomisen. Tapahtumapohjaisen ohjelmoinnin avulla callback-funktioita ja säikeitä käyttämällä voidaan luoda skaalautuvia palveluita. Node.js:ssä yhdistyy Unix verkko-ohjelmoinnin teho ja JavaScript ohjelmointikielen helppokäyttöisyys. Node.js:ssä käytetään Google V8 JavaScript moottoria (V8).

3.2.4 NPM

NPM lyhenne tulee sanoista Node Package Manager. Se on ohjelma, joka on luotu Node.js:n ohelle helpottamaan sen käyttöä. NPM koostuu komentorivillä käytettävästä asiakasohjelmistosta ja verkossa olevasta tietokannasta, joka sisältää Node.js:ssä käytettäviä paketteja. NPM:n kautta kyseisiä paketteja on mahdollista jakaa ja ladata käytettäväksi moduuleina. NPM ylläpitää paketteja, jotka ovat joko paikallisia riippuvuuksia jollakin projektilla tai globaaleja asennuksia. Kaikki projektin riippuvuutena olevat paketit voidaan asentaa yhdellä komennolla.

3.2.5 Node.js:n ajaminen palvelinympäristössä

Yksinkertaisenpiin käyttötarkoituksiin Node.js ympäristö ja sen palvelin voidaan käynnistää konsolin kautta Node.js:n omalla komennolla. Node.js:n päätiedosto ajetaan käynnistäen palvelimen kuuntelemaan kutsuja. Tilanteessa, jossa Node.js ympäristön ja sen palvelimen tarjoamat ominaisuudet pitää olla saatavilla täysipäiväisesti, Node.js ympäristö asetetaan jonkinlaiselle palvelimelle. PM2 on moduuli, joka on työkalu Node.js prosessien hallitsemiseen palvelinympäristössä. PM2:n kautta pystytään käynnistämään ja sulkemaan prosesseja sekä listaamaan niitä näyttäen kunkin prosessin tilan. PM2 mahdollistaa myös välittömän uudelleenkäynnistyksen tarvittaessa.

3.3 Express

3.3.1 Yleistä

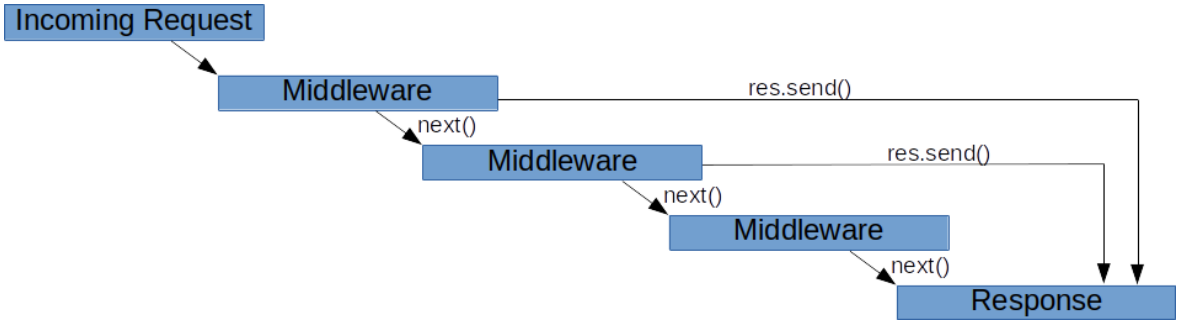
Express on Node.js:lle luotu yksinkertainen reititys- ja väliohjelmisto web-sovelluskehys, joka tarjoaa ominaisuuksia web-sovelluksen käytettäväksi. Tätä kehystä käytetään Node.js:n sisäänrakennetun web-palvelimen HTTP:n tilalla. Expressin ominaisuuksiin kuuluu esimerkiksi reititys, uudelleenohjaus ja välimuisti. Expressin avulla sovelluksen

taustajärjestelmä organisoidaan MVC arkkitehtuuriksi, joka tulee sanoista model-view-controller, suomeksi malli-näkymä-käsittelijä. Kyseisen arkkitehtuurin mukaan sovelluksen taustajärjestelmä jaetaan kolmeen osaan, joista malli vastaa järjestelmän tiedon tallentamisesta, ylläpidosta ja käsittelystä, näkymä määrittää käyttöliittymän ulkoasun ja käsittelijä vastaanottaa käyttäjältä tulevat käskyt muuttaen mallia ja näkymää käskyjen mukaisesti.

3.3.2 Väliohjelmistot

Express on itsessään minimalistinen sovelluskehys tarjoten hyvin vähän ominaisuuksia. Express sovellus on ytimessään sarja väliohjelmistotoimintojen kutsuja ja ominaisuudet Expressiin tulevat pääsääntöisesti juuri näistä väliohjelmistoista (middleware).

Väliohjelmistojen arkkitehtuuri on käytännössä funktio, joilla on pääsy pyyntö objektiin (req), vastaus objektiin (res) ja sen tehtävä on kutsua seuraavaa asennettua väliohjelmistoa next-parametrin avulla. Väliohjelmistotoiminnot kykenevät suorittamaan useita tehtäviä. Näihin lukeutuu koodin suorittaminen, HTTP-kutsun pyyntö- sekä vastausobjektin muuttaminen, pyyntö-vastaussyklin lopettaminen, seuraavan väliohjelmistotoiminnon kutsuminen. Tilanteessa, jossa väliohjelmistotoiminto ei päätä pyyntö-vastaussykliä, on kutsuttava seuraavaa väliohjelmistoa, jotta HTTP-kutsu ei jää kesken. Kuvassa (Kuva 6) on havainnollistettuna väliohjelmistojen toimintaperiaatetta.



Kuva 6. Väliohjelmistojen toimintaperiaate havainnollistettuna (Lee Brandt 2018)

Expressiä käyttävä sovellus voi käyttää kuudenlaista väliohjelmistoa. Nämä väliohjelmistotyypit ovat sovellustason- ja reititintason väliohjelmistot, virheiden käsittely väliohjelmistot, sisäänrakennetut väliohjelmistot ja kolmannen osapuolen väliohjelmistot. Sovellus- ja reititintason väliohjelmistoihin voi lisätä ehdollisen kiinnityspolun. Väliohjelmistoja voi ladata sarjassa, joka luo niistä oman alipinon suoritettavaksi.

Sovellustason väliohjelmisto on kiinnitetty sovellusobjektin instanssiin ja oletuksena se suoritetaan, joka kerta kun sovellus vastaanottaa pyynnön. Tämä kiinnitetään app.use()

funktiolla. Kiinnityspoluilla voidaan rajoittaa väliohjelmiston ajaminen tiettyihin reitittimen reitteihin. Rajoitus tehdään app.METHOD() funktiolla, jossa METHOD voi olla esimerkiksi GET tai POST. Reititin tason väliohjelmisto toimii samoin, mutta eroten edellisestä se on kiinnitetty Expressin reitittimen instanssiin. Virheiden käsittelyyn tarkoitetut väliohjelmistot eivät eroa muuten muista väliohjelmistoista kuin siten, että ne ottavat vastaan ylimääräisen argumentin virheen käsittelyä varten. Expressillä on muutama sisäänrakennettu väliohjelmisto, joka tarkoittaa sitä, että niitä ei tarvitse itse erikseen kirjoittaa. Lisäksi kolmannen osapuolen väliohjelmistot ovat valmiina käytettäväksi, kunhan ne ensin asennetaan. (Express.js.) Esimerkki sovellustason väliohjelmistosta on kuvassa (Kuva 7) havainnollistettuna koodiesimerkillä.

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

app.use(function (req, res, next) {
  console.log('Time:', Date.now())
  next()
})
```

Kuva 7. Sovellustason väliohjelmiston koodiesimerkki (Express.js)

3.4 Tietokanta

3.4.1 Yleistä

Tietokannat ovat kokoelmia varastoitua dataa organisoidussa muodossa. Nykypäivän tietokannat varastoivat tietoa tyypillisesti taulukkoihin, jotka koostuvat riveistä ja sarakkeista mahdollistaen tehokkaan tietokannan datan prosessoimisen ja kyselemisen. Tätä dataa pystytään hakemaan, hallitsemaan, muokkaamaan, päivittämään ja järjestämään joko jonkinlaisen tietokannan käyttöliittymän kautta tai tietokantakyselyillä jostakin sovelluksesta käsin. Suuri osa tietokannoista käyttää SQL (Standard Query Language) kyselykieltä tietokannan tietojen kyselemiseen, muokkaamiseen ja määrittelyyn (oracle.com 2020). Uusia ohjelmointikieliä tähän tarkoitukseen kehitetään ja SQL tietokantojen ohelle on nousemassa muita tietokantateknologioita kuten NoSQL (Not only SQL) tietokannat.

3.4.2 MongoDB

MongoDB on useamman alustan dokumenttikeskeinen tietokanta. MongoDB kuuluu niin sanottuihin NoSQL-tietokantoihin eli perinteisestä relaatiomallista poikkeaviin tietokantoihin. Tarkemmin luokassaan MongoDB määritellään dokumenttivarastoksi, jonka pääperiaatteena on se, että data kääritään dokumenttien sisään johonkin vakiomuotoiseen formaattiin. MongoDB:n tapauksessa tämä formaatti on BSON eli Binary JSON.

3.4.3 BSON

BSON lyhenne tulee sanoista Binary JSON. BSON on binäärinen muoto tietorakenteiden esittämiseen, joihin lukeutuu esimerkiksi assosiatiiviset taulukot eli avain-arvoparit ja kokonaisluvuilla indeksoidut taulukot. BSON sai alkunsa vuonna 2009 MongoDB:n omana datatyyppinä, mutta on nykyjään käytettävissä itsenäisesti sen ulkopuolella.

3.4.4 Kokoelmat

MongoDB ryhmittelee siihen tallennetut dokumentit kokoelmiin. Nämä kokoelmat ovat käyttötarkoitukseltaan verrattavissa relaatiotietokantojen tauluihin ja dokumentit olisivat samalla tavalla verrattavissa taulukon tietueisiin. Dokumentit ja kokoelmat eroavat kuitenkin relaatiotietokannan taulukoista ja tietueista siten, että kokoelman dokumenteilla voi olla täysin toisistaan eroavia kenttiä toisin kuin relaatiotietokannan tietueilla, jotka ovat kentiltään samanlaisia.

3.4.5 Mongoose

Mongoose on moduuli MongoDB tietokannan käyttämisen helpottamiseksi Node.js sovelluksessa. Mongoose tarjoaa suoraviivaisen ja helppokäyttöisen ratkaisun sovelluksen ja tietokannan yhteyden luomiseen, tietokantaan tallennettavien tietojen mallintamiseen, datatyyppien muuttamiseen toiseksi, validointiin ja tietokannan kyselyjen muodostamiseen.

3.4.6 Multer

Multer on Node.js väliohjelmisto, joka on tarkoitettu prosessoimaan multipart/form-data-datatyyppiä, jota lähetetään HTTP-kutsuissa. Tätä datatyyppiä käytetään kuljettamaan tiedostoja kuten kuvia ja videoita. Multer mahdollistaa tämän datan tallentamisen, suodattamisen, rajaamisen ja virheiden käsittelyn.

4 Taustajärjestelmän toteutus

4.1 Projektin kansiorakenne

Projektin juurikansio sisältää alikansiot controllers, models, middlewares, routes, node_modules ja public. Juurikansio sisältää myös tiedostot app.js, package.json sekä package_lock.json. Kansiot sisällytetään versionhallintaan ja laitetaan jakoon projektin Githubiin.

Mainituilla alikansioilla ja tiedostoilla on seuraavanlaiset merkitykset:

- Controllers kansio sisältää ohjaintiedostoja, joissa on määritelty taustajärjestelmän toiminnallisuutta mahdollistavia funktioita. Näitä funktioita ajetaan yleisimmin käyttöliittymän lähettämän pyynnön tuloksena.
- Models kansiossa ovat tiedostoissa mongoose moduulin tarjoamien skeemojen määrittelyt. Näissä skeemoissa määritellään tietokantaan tallennettavan datan rakenne JSON formaatissa.
- Middlewares kansion tiedostoissa on määriteltynä itse rakennettuja väliohjelmistoja.
- Routes kansio sisältä tiedostot, joissa määritellään Express palvelimen reitittimen reitit.
- Node_modules kansiossa on NPM:llä asennettujen moduulien lähdekoodi. Tämä kansio jätetään versionhallinnan ulkopuolelle suuren kokonsa takia.
- Public kansio sisältää julkiseen jakoon tarkoitetut tiedostot. Nämä ovat lähinnä pdf- ja kuvatiedostoja, joita esitetään käyttöliittymissä.
- App.js tiedoston ajamalla käynnistetään Node.js sovellus palvelimineen. Tämä käynnistystiedosto sisältää sovelluksen alkumäärittelyt, kuten tietokantayhteyden muodostamisen, väliohjelmistojen kiinnityksen sekä useimmat sovelluksen, tai sen osa-alueiden asetukset.
- Package.json tiedosto sisältää joitain projektin määrittelyjä sekä sen moduuli riippuvuudet.
- Package_lock.json tiedosto lukitsee moduuliriippuvuudet tiettyihin moduulien versioihin.

4.2 Palvelimen toteutus

Taustajärjestelmän palvelin toteutettiin Express sovelluskehyksellä. Palvelin on tehty HTTPS-protokollalla mahdollistaen suojatun tiedonsiirron. Sovelluksen käynnistystiedostosta löytyvät palvelimen alustuksen määrittelyt kuten sovelluksen instanssin kiinnitys palvelimeen sekä palvelimen kredentiaalit. Kredentiaaleihin kuuluu palvelimen rajapintaan käsiksi pääsemiseen käytettävän verkkotunnuksen yksityinen avain sekä sertifikaatti. Palvelin on luotu asetusten määrittelyn jälkeen createServer() funktiolla ja käynnistetään listen() funktiolla sovelluksen käynnistystiedoston ajon yhteydessä. Palvelin käyttää Expressin reititintä reititykseen. Reititin on luotu sen omassa tiedostossa määrittelyineen.

Palvelimessa on käytetty väliohjelmistoja (middleware) mahdollistamaan useita ominaisuuksia. Käytetyt väliohjelmistot ovat body-parser, cors, multer, checkAdminAuth ja checkEventAuth, joista kaksi jälkimmäistä on itse kehitettyjä.

Mainittujen väliohjelmistojen tarkoitus on seuraava:

- Body-parser on tarkoitettu jäsentämään HTTP-pyyntöjen rungon dataa. Tämän projektin kohdalla body-parser väliohjelmisto määriteltiin jäsentämään pyyntöjen runko JSON muotoon, koska tietokannan skeematkin ovat kyseisessä muodossa
- Cors väliohjelmisto mahdollistaa CORS ominaisuudet, jotta sovelluskokonaisuus toimii selaimissa, joissa CORS kykyisyyttä vaaditaan sovelluksilta.
- Multer väliohjelmistoa käytetään tallentamaan HTTP-pyynnöissä vastaanotetut multipart/form-data datatyypin tiedostot julkiseen jakoon tarkoitettujen tiedostojen kansioon.
- CheckAdminAuth on rakennettu tarkistamaan kirjautumisen session voimassaolo järjestelmänhallinnan web-sovellukselta tulevien pyyntöjen kohdalla.
- CheckEventAuth tehtiin suorittamaan sama tarkastus kuin edellinen, mutta mobiililaitteelta tuleville pyynnöille. Nämä kaksi rakennettiin eri väliohjelmistoiksi, koska nähtiin helpompana sisällyttää eri väliohjelmisto eri reiteissä tarpeen mukaan, kuin luoda vain yksi väliohjelmisto, jossa on rakennettuna logiikka katsomaan kumpaa tarkastusta tarvitaan.

4.3 Autentikoinnin toteutus

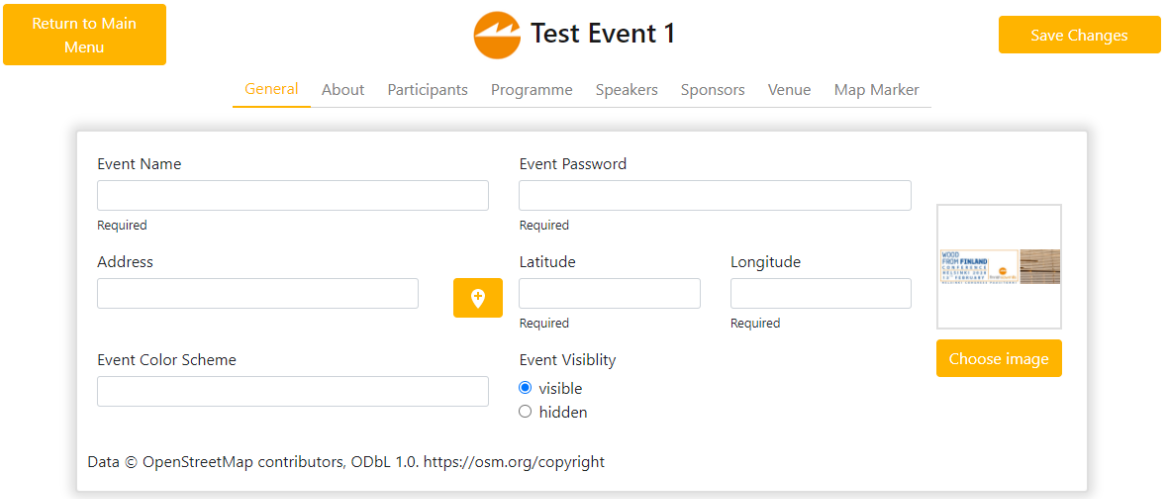
Autentikointi toteutettiin pääasiassa jsonwebtoken ja bcrypt moduuleilla. Autentikointiin tarvittavat salasanat tallennetaan tietokantaan salatussa muodossa. Salauksessa käytetään bcrypt moduulin hash() funktiota, joka suorittaa salauksen funktioon syötetyllä

määrällä suolauskertoja. Reitittimessä on luotuna reitit, sekä järjestelmänhallinnan web-sovellukselle, että mobiilisovellukselle, jotka saatuaan pyynnön käyttöliittymältä, suorittavat ohjaintiedostoista asianmukaiset funktiot. Näissä funktioissa haetaan käyttöliittymän mukaan joko tapahtumaa, tai järjestelmänhallitsijaa pyynnön määrittelyn mukaisesti. Tämän jälkeen verrataan vastaanotettua salasanaa tietokannasta löytyneeseen salattuun salasanaan brypt moduulin compare() funktiolla. Vertauksen täsmätessä luodaan json-webtoken moduulilla muutaman tunnin voimassa oleva token lähetettäväksi takaisin käyttöliittymälle. Tällä tokenilla ja sen rajoitetulla voimassaololla on toteutettu kirjautumisen session kesto.

4.4 Tapahtuman datan rakenne

Seuraavaksi käydään mobiilisovelluksessa näytettävien ja web-sovelluksessa hallittavien tapahtumien datan rakennetta läpi web-sovelluksen käyttöliittymän näkymien kautta. Datan rakenteet on määritelty mongoose moduulin skeema ominaisuudella models kansion tiedostoissa. Skeeman rakenne on JSON muodossa avain-arvopareina. Niissä määritelty rakenne ja datatyypit ovat samat, missä data tullaan tallentamaan tietokantaan. Datan oikeellisuuden validointi on myös toteutettu skeemojen kautta. Dataa, jota yritetään tallentaa skeeman rakenteesta poiketen, ei tulla tallentamaan. Virhetilanteessa data jää tallentamatta niiltä osin, missä se poikkeaa skeemasta. Mikäli virheellinen osa-alue on vaadittu, koko tallennusoperaatio epäonnistuu.

Jokaisella tapahtumalla on olemassa metadata tiedot, jotka kattavat yleistiedot tapahtumasta. Tietoihin kuuluu avain-arvoparit tapahtuman nimi, kuvan staattinen osoite, tapahtuman osoite, leveysaste, pituusaste, näkyvyys ja väriteema. Nämä arvot ovat metadata objektin sisällä. Kuvassa (Kuva 8) on kuva web-sovelluksen käyttöliittymän näkymästä, josta metadata objektin tietoja voidaan muokata. Kuvassa (Kuva 9) vastaavasti metadata objekti mongoose moduulin skeemassa.

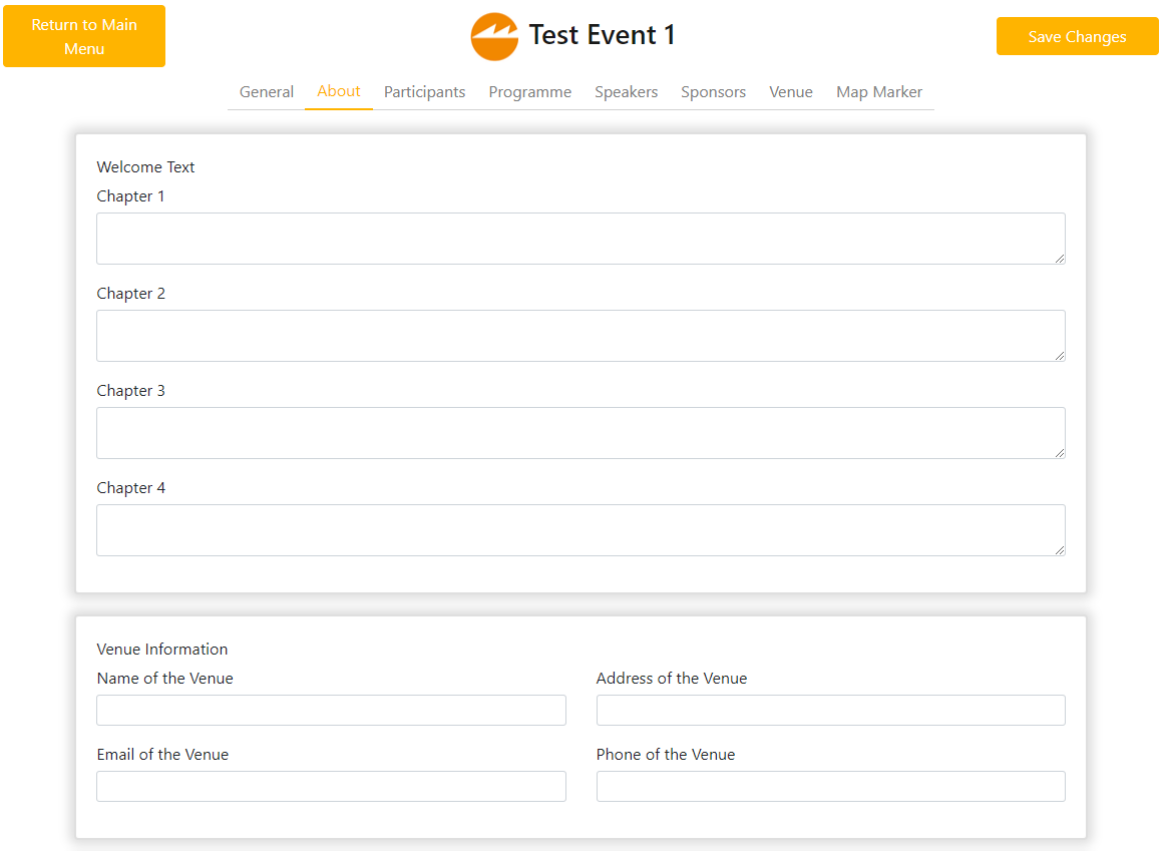


Kuva 8. Tapahtuman yleistietojen muokkausnäkymä

```
metadata : {
  eventName: String,
  eventImage: String,
  address: String,
  lat: String,
  long: String,
  visibility: String,
  colorScheme: String
},
```

Kuva 9. Tapahtuman yleistiedot mongoose skeemassa

Muille tapahtuman tiedoille on olemassa about objekti. Se sisältää dataa, joka haluttiin kategorisoida metadatan ulkopuolelle. Tämän objektin avain-arvopareihin kuuluu tapahtuman järjestäjien web-sivujen osoite, otsikko, neljä arvoa tekstikokonaisuuksille sekä kaksi arvoa lisäteksteille. About objektin sisällä on myös kaksi muuta objektia, jotka ovat tapahtumapaikan tiedot ja lisäinformaatiot. Tapahtumapaikan objektin tietoihin kuuluu nimi, osoite, tapahtumapaikan puhelinnumero ja sähköposti. Lisätietojen objekti sisältää tapahtuman web-sivun osoitteen, järjestäjän nimen ja sen sähköpostin. Kuvassa (Kuva 10) esitetään tämän kokonaisuuden muokkausnäkymä havainnollistamassa dataa. About kokonaisuuden skeema on kuvattuna kuvassa (Kuva 11).

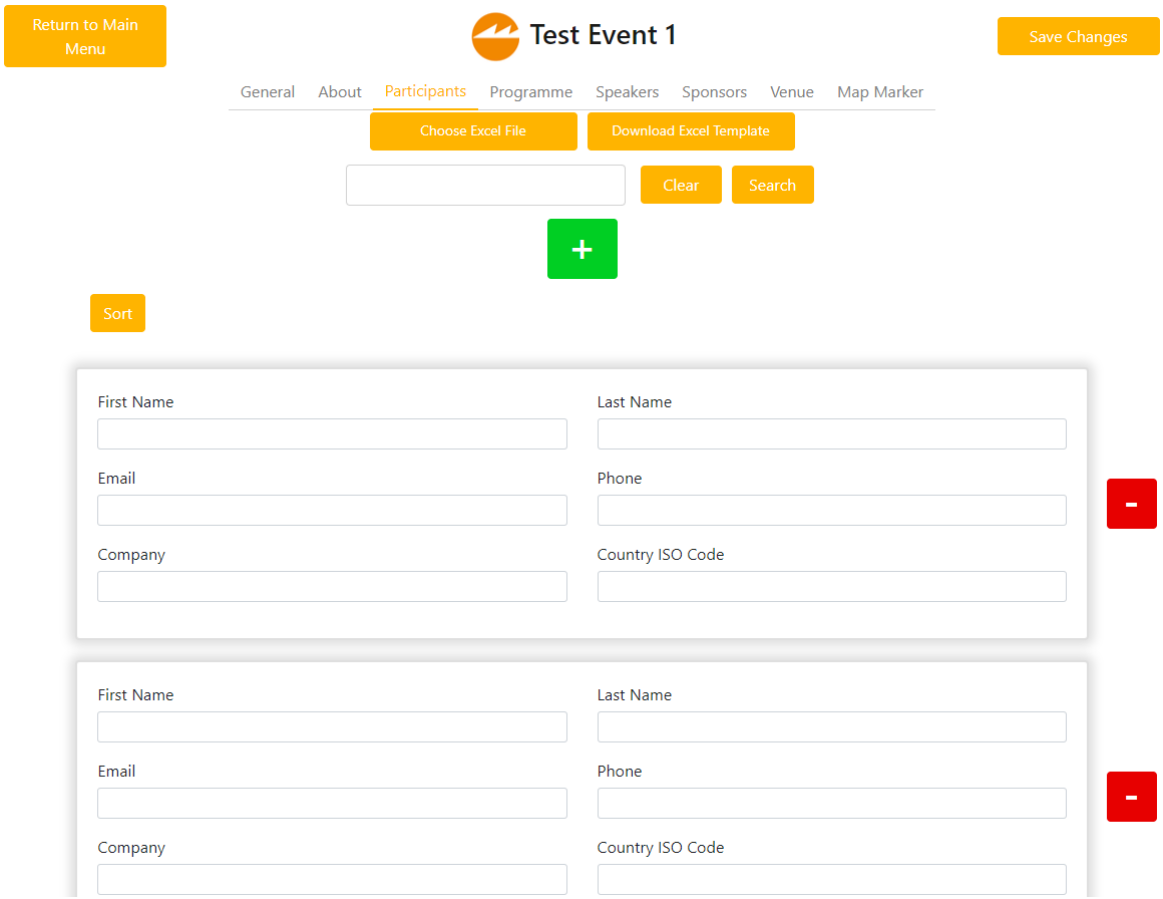


Kuva 10. About tietojen muokkausnäkymä

```
about : {
  eventWebUrl: String,
  eventPlace: {
      name: String,
      address: String,
      phone: String,
      email: String,
  },
  title: String,
  bodyText1: String,
  bodyText2: String,
  bodyText3: String,
  bodyText4: String,
  moreInformation: {
      eventWebsite: String,
      organizer: String,
      email: String
  },
  disclaimer1: String,
  disclaimer2: String
},
```

Kuva 11. About objektin rakenne mongoose skeemassa

Tapahtuman osallistujien tiedot on listattu objekteina participants taulukossa. Näiden objektien avain-arvoparit ovat osallistujan maa, etunimi, sukunimi, sähköposti, puhelinnumero ja yhtiö. Mobiilisovelluksen käyttäjien logiikka on toteutettu osallistujien kautta. Käyttäjiä ei erikseen ole olemassa siinä mielessä, että heillä ei ole omaa profiilia tai salasanaa, jolla he kirjautuvat sisään. Sen sijaan he kirjautuvat sisään tapahtumiin tapahtumakohtaisesti käyttämällä tapahtuman tiedoista löytyvällä sähköpostillaan ja tapahtuman tiedoissa määritellyllä tapahtuman omalla salasanalla. Kuvassa (Kuva 12) on havainnollistettuna osallistujien data muokkausnäkymän kautta. Kuvassa (Kuva 13) on vastaavasti havainnollistus osallistujien skeeman kannalta.

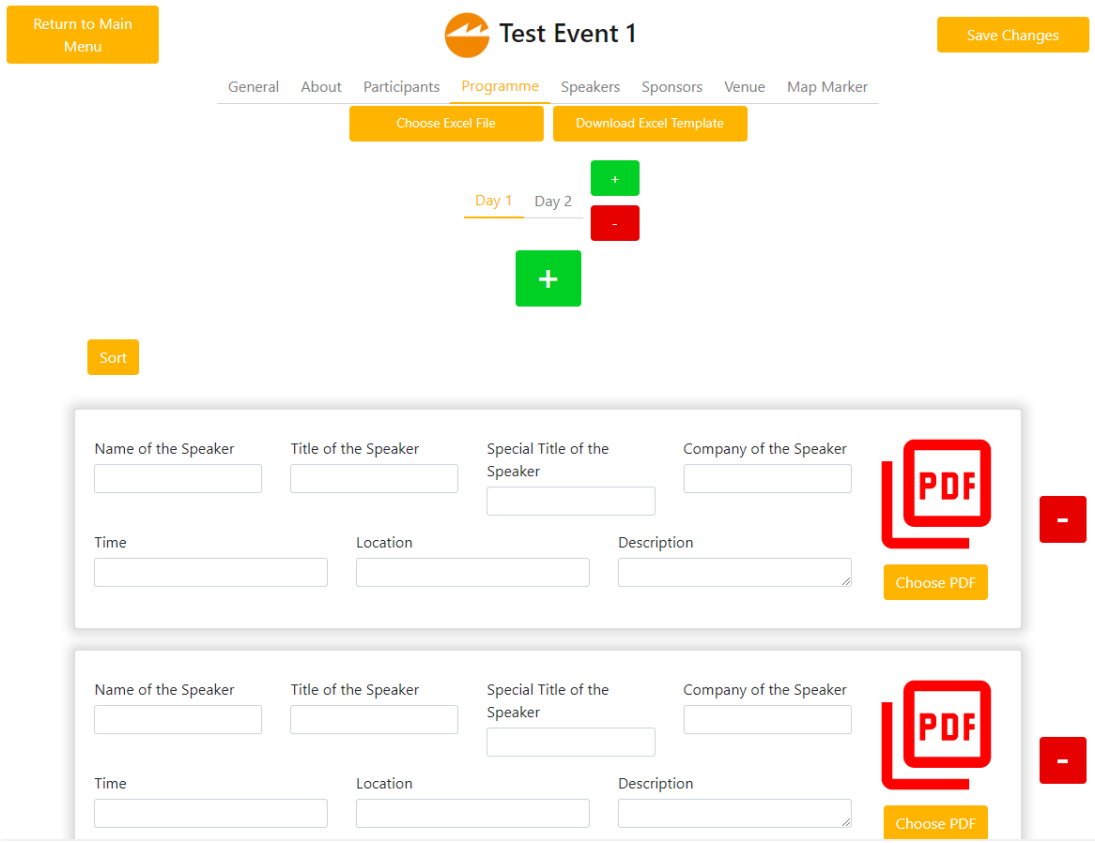


Kuva 12. Osallistujien tietojen muokkausnäkymä

```
participants : [
  {
    Country: String,
    FirstName: String,
    LastName: String,
    Email: String,
    Phone: String,
    Company: String
  }
],
```

Kuva 13. Participants taulukon rakenne mongoose skeemassa

Tapahtumilla on ohjelmaa useimmin usealle päivälle. Tarvittiin siis rakenne, jossa voidaan jaotella päivät ja niihin kuuluvat ohjelmat. Tapahtumien ohjelman datan rakenne on seuraava. Programme nimisestä taulukko löytyy avain-arvopari päivälle, sekä toinen taulukko, jossa on objekteina päivän ohjelmanumero. Ohjelmanumeron tiedoissa on tiedot ajasta, paikasta, kuvauksesta, puhujan nimestä, hänen nimikkeestään, erityisnimikkeestä, yhtiöstä sekä esityksen pdf tiedoston staattisen jaon osoite. Kuva (Kuva 14) havainnollistaa dataa käyttöliittymän avulla ja kuvassa (Kuva 15) on tapahtuman ohjelman rakenne skeemassa.

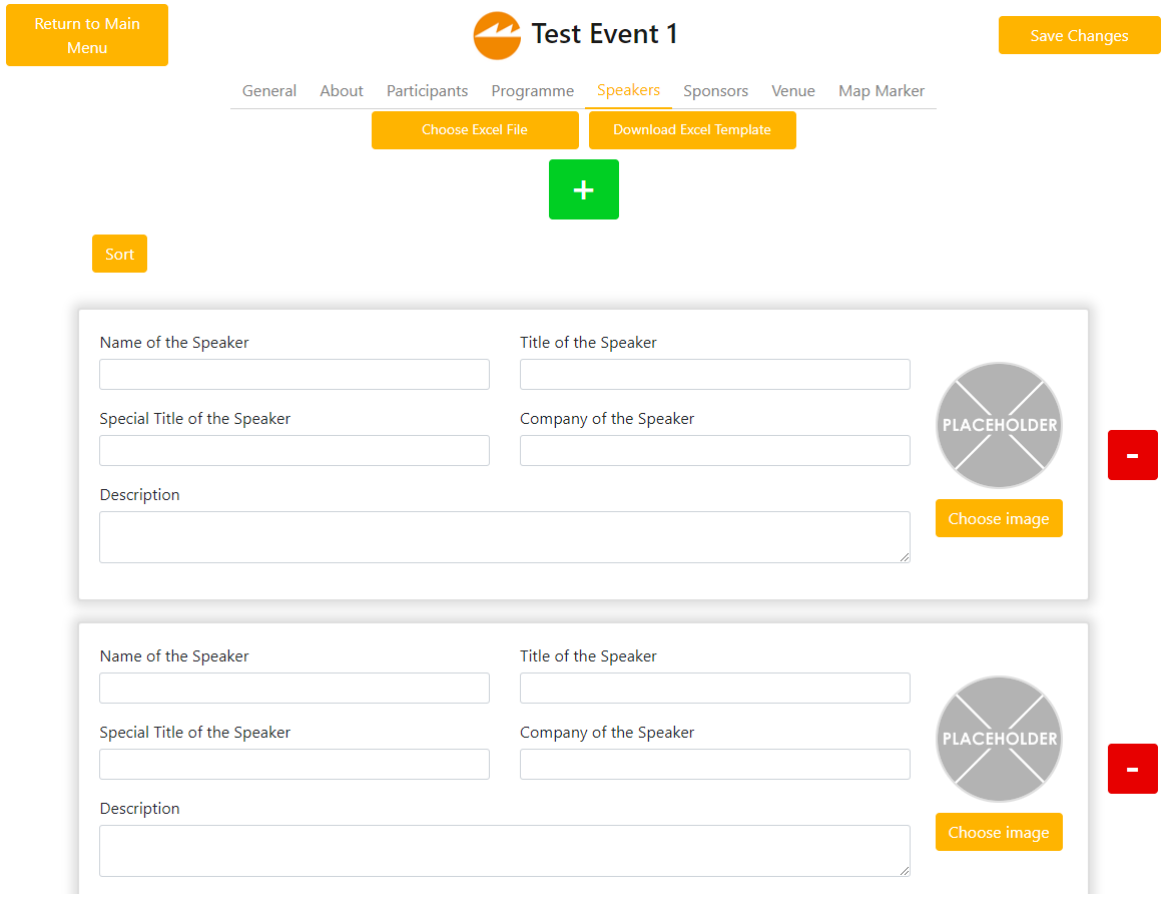


Kuva 14. Ohjelman muokkausnäkymä

```
programme: [
  {
    day: String,
    content: [
      {
        Time: String,
        Location: String,
        Description: String,
        NameOfSpeaker: String,
        TitleOfSpeaker: String,
        SpecialTitleOfSpeaker: String,
        CompanyOfSpeaker: String,
        Pdf: String
      }
    ]
  }
],
```

Kuva 15. Programme taulukon rakenne skeemassa.

Tapahtumassa esiintyville puhujille on oma speakers taulukkonsa, jonka objekteilla on seuraavat avain-arvoparit: Puhujan, nimi, nimike, erityisnimike, yhtiö, kuvaus ja kuva. Kuvassa (Kuva 16) on datan rakenne muokkausnäkymän kannalta ja kuvassa (Kuva 17) on rakenne skeemassa.

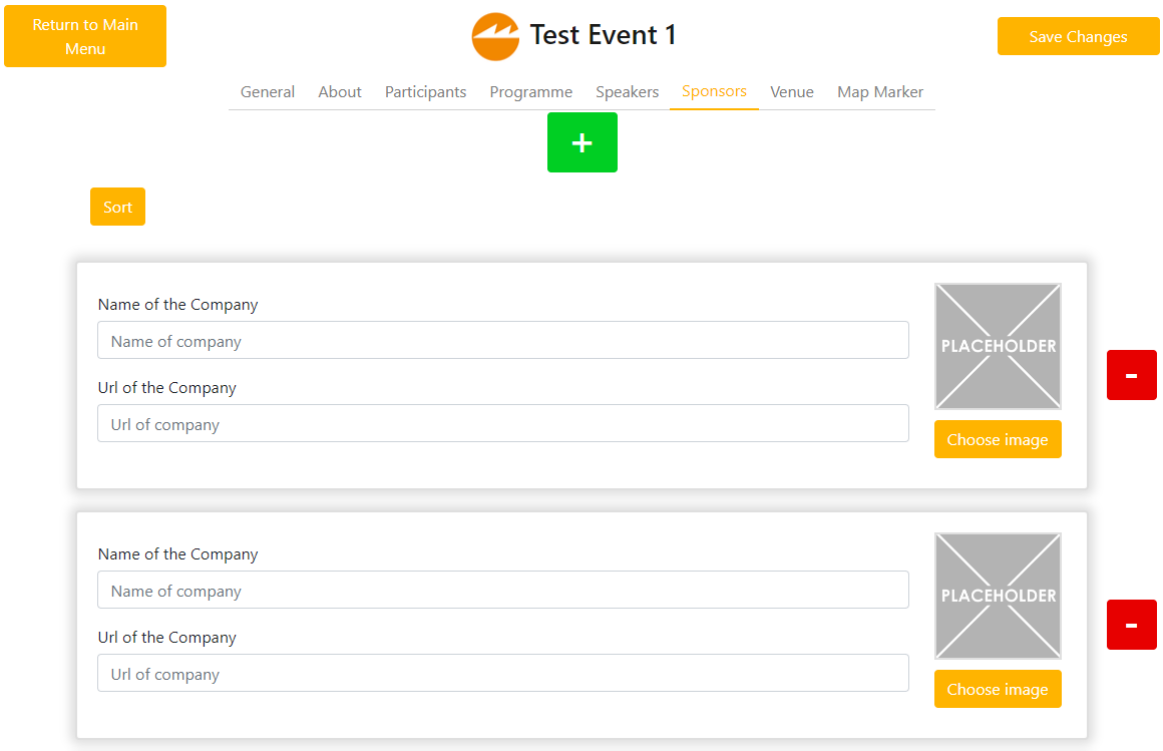


Kuva 16. Puhujien muokkausnäkymä

```
speakers : [
  {
    Speaker: String,
    Title: String,
    SpecialTitle: String,
    Company: String,
    Description: String,
    ImageID: String
  }
],
```

Kuva 17. Speakers taulukon rakenne skeemassa.

Tapahtumalla on myös sponsoreita, jotka listataan objekteina taulukossa sponsors. Objektit sisältävät tapahtumaa sponsoroivan yhtiön nimen, web-sivujen osoitteen sekä logon kuvan staattisen jaon osoitteen. Kuvassa (Kuva 18) on data muokkausnäkymän kautta ja kuvassa (Kuva 19) on rakenne skeemassa.

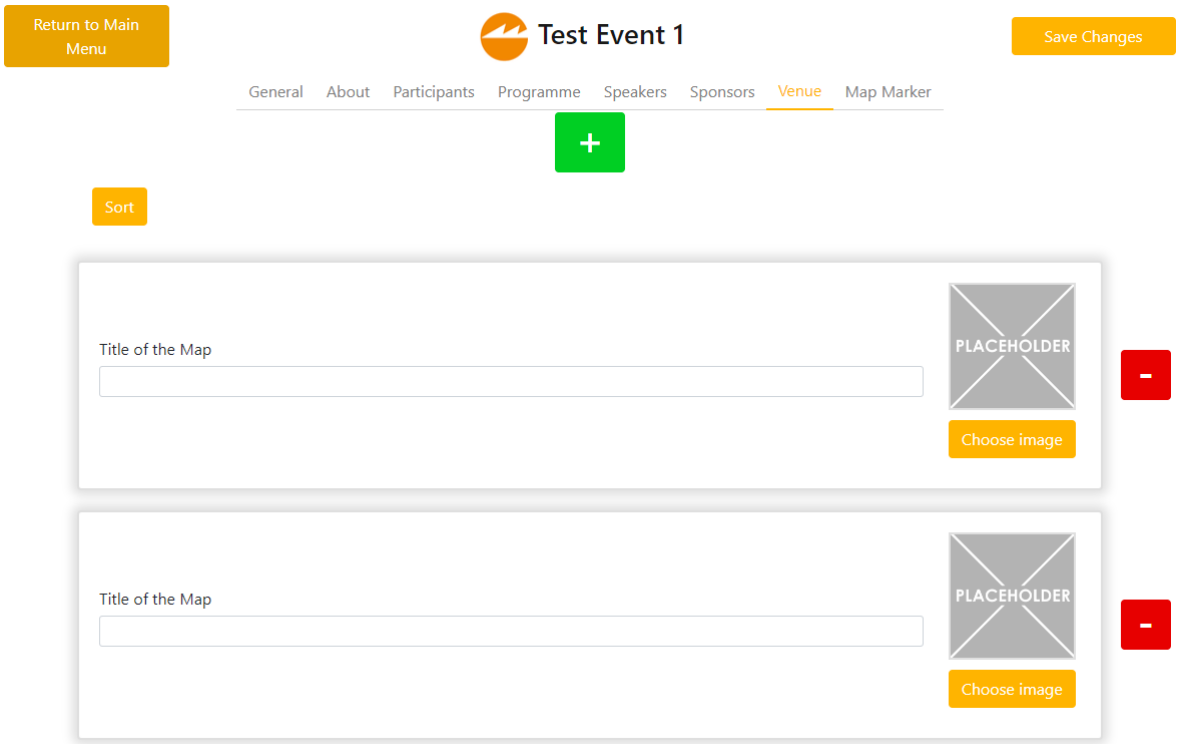


Kuva 18. Sponsorien muokkausnäkymä

```
sponsors : [
  {
    CompanyName: String,
    CompanyUrl: String,
    ImageID: String
  }
],
```

Kuva 19. Sponsors taulukon rakenne skeemassa

Tapahtuman karttaa varten tarvittiin taulukko mahdollisille tapahtumapaikan osa-alueille kuten kerroksille tai siiville. Venue taulukon objekteilla on yksinkertaisuudessaan arvoina otsikko sekä kartan kuvan staattisen jaon osoite. Kartan osan tyyppiä kuten kerrosta tai siipeä ei erikseen ole määritelty avain-arvoparilla vaan otsikointi riitti mobiilisovellukselle karttojen esittämisessä. Muokkausnäkymällä havainnollistettu datan rakenne on esitetty kuvassa (Kuva 20) sekä skeemassa oleva rakenne on kuvassa (Kuva 21).

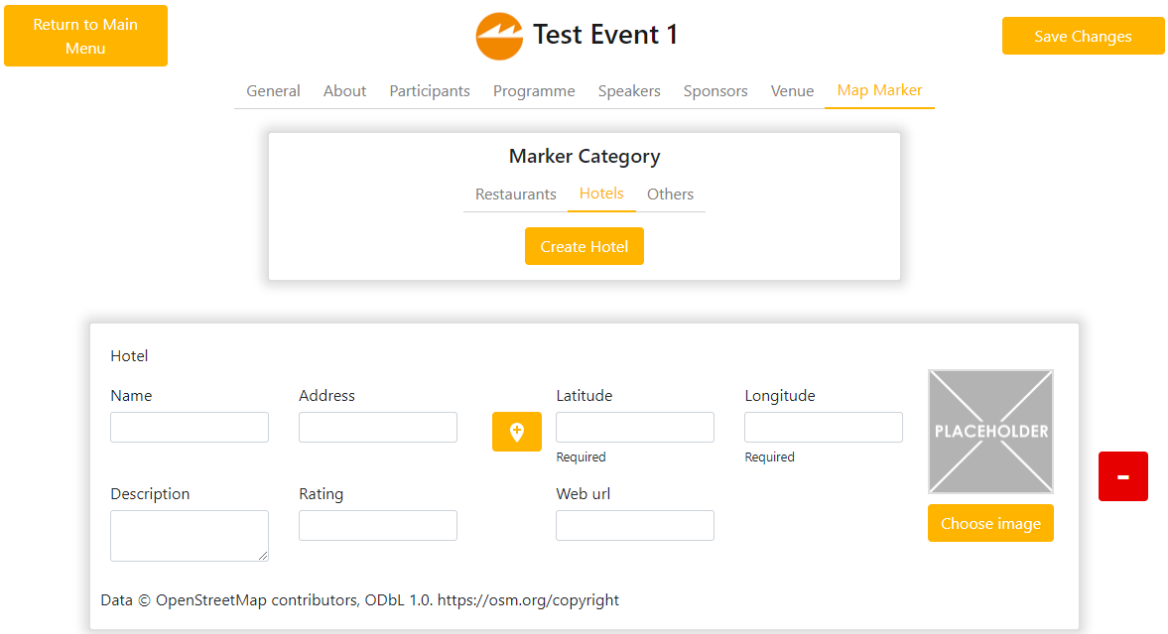


Kuva 20. Karttojen muokkausnäkymä

```
venue: [
  {
    title: String,
    image: String
  }
],
```

Kuva 21. Venue taulukon rakenne skeemassa.

Mobiilisovelluksessa oli myös tarkoitus näyttää karttapalvelun kautta tietoa lähialueesta esimerkiksi ravintoloiden ja hotellien osalta. Sijainteja haluttiin mahdolliseksi lisätä manuaalisesti karttaan järjestelmänhallinnan web-sovelluksesta, joten sijaintitiedot pitää tallentaa tietokantaan. MapData objekti sisältää kolme taulukkoa restaurants, hotels ja others. Näistä restaurants on nimensä mukaisesti tarkoitettu listaamaan ravintoloja, hotels hotelleja, ja others muiden kategorioiden sijainteja. Restaurants taulukosta löytyvät paikkatiedot, nimi, osoite, kuvaus, kategoria, arvostelu, web-osoite ja kuvan sijainti palvelimella. Hotels-taulukon objektit ovat muuten samanlaiset mutta kategoria arvo puuttuu. Others-taulukko seuraa myös muuten samaa kaavaa mutta arvostelu puuttuu. Kuvassa (Kuva 22) on esitetty karttatietojen muokkausnäkymä ja rakenne skeemassa on kuvassa (Kuva 23).



Kuva 22. Karttapalvelun sijaintimerkintöjen muokkausnäkymä

```
mapData: {
  restaurants: [
    {
      lat: String,
      long: String,
      name: String,
      address: String,
      description: String,
      category: String,
      rating: String,
      webURL: String,
      image: String
    }
  ],
  hotels: [
    {
      lat: String,
      long: String,
      name: String,
      address: String,
      description: String,
      rating: String,
      webURL: String,
      image: String
    }
  ],
  others: [
    {
      lat: String,
      long: String,
      name: String,
      address: String,
      description: String,
      category: String,
      webURL: String,
      image: String
    }
  ]
}
```

Kuva 23. MapData objektin rakenne skeemassa

4.5 Tietokannan CRUD operaatioiden toteutus

CRUD eli create, read, update ja delete operaatiot on toteutettu Expressin reitittimen kautta. Reitittimen reitit suorittavat funktioita saadessaan pyynnön, jonka data kulkeutuu ensin väliohjelmistojen läpi. Reitittimeen on tehty reitit tapahtuman tietojen luomista, hakemista, päivittämistä ja poistoa varten. Tietokantakyselyt tapahtuvat mongoosen niihin tarkoitetuilla funktioilla, joissa määritellään skeeman kautta mistä tietokannan dokumentista on kysymys, sekä tarkempi määrittely avain-arvoparien kautta.

Tapahtuman tietojen luomisen logiikassa tarkastetaan, että juuri luotavan tapahtuman nimi ei ole sama, kuin jo olemassa olevan tapahtuman nimi. Mikäli virhettä ei tapahdu jatketaan asettamalla tapahtuman tiedot mongoose moduulin skeeman mukaisesti saadusta pyynnöstä. Tästä syntyvä objekti tallennetaan mongoosen Model.save() funktiolla tietokantaan. Uuden tapahtuman luonnissa suoritetaan myös tapahtuman salasanan salaus ja tallennus autentikoinnin skeeman mukaisesti.

Tapahtumien tietojen hakemisessa mobiilisovelluksen käyttöliittymä pyytää tapahtumien perustietoja kirjautuneen henkilön sähköpostin mukaan. Sovelluksessa näytetään vain ne tapahtumat, joihin kyseinen henkilö on osallistunut, joten taustajärjestelmässä haku tapahtuu myös tämän perusteella ja palautetaan vain niiden tapahtumien perustiedot, joissa määritelty sähköposti löytyy participants taulukon objekteista. Käyttöliittymässä tapahtumien sisäisiin näkymiin navigoitaessa tapahtumien tiedot palautetaan pyydettäessä. Tietojen haku tietokannasta tapahtuu mongoosen Model.find() funktiolla.

Tapahtuman päivittämisen logiikassa haetaan tietokannasta jo olemassa olevan tapahtuman tiedot, ja mikäli tapahtuma löytyy, asetetaan uudet pyynnön mukana saapuneet tiedot skeeman mukaisesti vanhojen tietojen tilalle samalla Model.save() funktiolla, jolla myös luodaan uutta tietoa. Tilanteessa, jossa on olemassa jo edeltävää tietoa, funktio korvaa edelliset tiedot uusien tietojen luomisen sijaan. Jos tapahtuman autentikointiin liittyviä tietoja päivitettiin, on uusi salasana salattava kuten luonnissakin tehtiin, jonka jälkeen vanha korvataan.

Tapahtuman poistamisen logiikassa yksinkertaisuudessaan haetaan tapahtumaa, joka halutaan poistaa, ja jos pyynnön määrittelyn mukainen tapahtuma löydetään, poistetaan kaikki sen tiedot mongoosen Model.delete() funktiolla.

4.6 Tiedostojen lisääminen ja jakaminen

Tapahtumiin haluttujen kuva ja pdf tiedostojen esittäminen edellytti taustajärjestelmältä niiden tallentamista ja hakemista. Ratkaisuksi valittiin tiedostojen tallentaminen taustajärjestelmän public kansion alikansioihin tietokannan sijaan. Public kansion alikansioita ovat kansiot, jotka on nimetty kunkin tapahtuman Id:n mukaisesti. Kansiot varastoivat tiedostoja tapahtumittain.

Tiedostojen tallentaminen toteutettiin multer moduulilla. Tiedostojen tallennuksen reitti ajaa multer väliohjelmiston upload.array() funktion, joka vastaanottaa pyynnön multipart/formdata datatyypin tiedostotaulukon. Väliohjelmistossa multer.diskStorage() funktiolla määritellään mitä kansioita käytetään tiedostojen tietovarastona, sekä mitä tietovaraston kansiota käytetään kussakin pyynnössä vastaanotettujen tiedostojen tallentamiseen. Määriteltynä ovat edellisessä kappaleessa mainitut alikansiot, joihin vastaanotetut tiedostot tallennetaan pyynnön määrittelyn mukaisesti. Mikäli määriteltyjä kansioita ei ole olemassa, multer luo ne. Tallennettujen tiedostojen tiedostonimiksi asetettiin niiden alkuperäinen tiedostonimi.

Tiedostojen jako toteutettiin Expressin express.static() funktiolla, jolla voidaan laittaa tietyt kansiot jakoon staattisesta URL osoitteesta. Projektin public kansio jaettiin tällä tavalla, jotta käyttöliittymät pääsevät käsiksi tiedostoihin. Jotta käyttöliittymät ovat tietoisia siitä, mihin minkäkin kuvan kuuluu mennä, sekä mistä osoitteesta kukin tiedosto on löydettävissä, on tietokantaan lisättävä linkitys. Tietokantaan lisätään siis kunkin tiedoston osoite oikeaan kenttään tiedoston tallentamisen yhteydessä. Kuvien poistaminen tapahtuu tapahtuman poistamisen yhteydessä, jossa levyltä poistetaan public kansiosta poistettavaan tapahtumaan kuuluva alikansio kaikkine sisältöineen.

5 Yhteenveto

Työn tavoitteena oli kehittää Sahateollisuus RY:lle taustajärjestelmä tapahtumapalvelun sovelluskokonaisuuteen mahdollistamaan käyttöliittymien ominaisuuksia. Tutkimusongelmana oli kyseisen taustajärjestelmän toteuttaminen valituilla teknologioilla.

Työn tuloksena saatiin kehitettyä taustajärjestelmä käyttötarkoituksen mukaisesti valituilla teknologioilla. Kaikki vaaditut ominaisuudet saatiin toteutettua toimivilla ratkaisuilla. Taustajärjestelmä on annettu asiakkaalle ja on tällä hetkellä testikäytössä. Asiakkaan käyttökokemukset ovat olleet asiakaspalautteen mukaan positiivisia ja projektiin on oltu kokonaisuudessaan asiakkaan puolelta tyytyväisiä.

Kehittäjän näkökulmasta katsottuna taustajärjestelmän toteutuksessa on kuitenkin monta asiaa, jotka olisi voitu tehdä paremmin. Projektin reflektiossa havaittiin, että joitakin teknologioita olisi voitu vaihtaa toisiin ja valittuja teknologioita ei paikka paikoin käytetty parhailla mahdollisilla tavoilla tai niiden kaikkia mahdollisia ominaisuuksia, joita olisi voitu hyödyntää, ei hyödynnetty.

Taustajärjestelmän jatkokehitys on mahdollista. Tapahtuuko jatkokehitystä vai ei on täysin kiinni asiakkaasta. Sovelluskokonaisuus on rakennettu käytettäväksi tapahtumiin, joissa on tarkoitus olla fyysisesti mukana tapahtumapaikalla. Tästä herääkin kysymys vaikuttaako COVID-19 palvelun jatkamiseen. On täysin pimennossa, meneekö projekti jäihin ja mikäli näin tapahtuu, kuinka kauaksi aikaa. Tämä asettaakin varjon jatkokehityksen ylle. Tässä vaiheessa on vaikea sanoa, jatkokehitetäänkö taustajärjestelmää vai ei.

Mikäli jatkokehitystä tapahtuu, on asiakas maininnut joitakin mahdollisia ominaisuuksia, joita alettaisiin kehittää seuraavaksi. Näitä ominaisuuksia ovat olleet esimerkiksi jonkinlainen kontaktien lisääminen ja keskusteleminen muiden käyttäjien kanssa. Tämä tarkoittaisi asiakkaan mukaan yksityisviestien ja yleisen chatin luomista.

Vaikka taustajärjestelmässä löytyy paljon kehitettävää, saatiin se kuitenkin toteutettua vaatimukset täyttäen ja positiivisella asiakaspalautteella. Projektia voidaan siis sanoa onnistuneeksi sekä tavoitteiden kannalta että antoisana oppimiskokemuksena.

Lähteet

Brandt, L. 2020. Build and Understand Express Middleware through Examples. Viitattu 19.10.2020. Saatavilla https://developer.okta.com/blog/2018/09/13/build-and-understand-express-middleware-through-examples

Capan, T. 2013. Why the Hell Would You Use Node.js. Viitattu 17.10.2020. Saatavissa https://medium.com/the-node-js-collection/why-the-hell-would-you-use-node-js-4b053b94ab8e

Express.js, 2020. Using middleware. Viitattu 19.10.2020. Saatavissa https://expressjs.com/en/guide/using-middleware.html

MDN web docs, 2020. Viitattu 14.10.2020 Saatavissa https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

MDN web docs, 2020. Cross-Origin Resource Sharing (CORS). Viitattu 15.10.2020. Saatavissa https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

Nodejs.org, 2020. About Node.js. Viitattu 17.10.2020. Saatavissa https://nodejs.org/en/about/

Node.js 2020. The Node.js Event Loop, Timers, and process.nextTick(). Viitattu 18.10.2020. Saatavissa https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

ÖZLÜ, A. 2020. Mastering REST Architecture — REST Architecture Details. Viitattu 13.10.2020. Saatavissa https://medium.com/@ahmetozlu93/mastering-rest-architecture-rest-architecture-details-e47ec659f6bc

Patel, R. 2020. Node Js vs PHP: Comparing Stats, Features and Performance in 2020. Viitattu 17.10.2020. Saatavissa https://aglowiditsolutions.com/blog/node-js-vs-php/

Restfulapi.net, 2020. What is REST. Viitattu 13.10.2020. Saatavissa https://restfulapi.net/

Restfulapi.net, 2020. Client-server. Viitattu 14.10.2020. Saatavissa https://restfulapi.net/rest-architectural-constraints/

Restfulapi.net, 2020. Stateless. Viitattu 14.10.2020. Saatavissa https://restfulapi.net/statelessness/

Restfulapi.net, 2020. Layered system. Viitattu 14.10.2020. Saatavissa https://restfulapi.net/rest-architectural-constraints/

Restfulapi.net, 2020. Richardson Maturity Model. Viitattu 15.10.2020. Saatavissa https://restfulapi.net/richardson-maturity-model/

Restfulapi.net, 2020. HATEOAS Driven REST APIs. Viitattu 15.10.2020. Saatavissa https://restfulapi.net/hateoas/

Sahateollisuus RY, 2020. Viitattu 9.10.2020. Saatavissa https://sahateollisuus.com/

V8, 2020. What is V8. Viitattu 18.10.2020. Saatavissa https://v8.dev/

Oracle.com 2020. Database / What is a database? Viitattu 21.10.2020. Saatavissa https://www.oracle.com/database/what-is-database.html