OULUN AMMATTIKORKEAKOULU

Sakari Mursu

REST-TIETOKANTARAJAPINTA MOBIILISOVELLUKSELLE JA WEB-SIVUSTOLLE

REST-TIETOKANTARAJAPINTA MOBIILISOVELLUKSELLE JA WEB-SIVUSTOLLE

Sakari Mursu
Opinnäytetyö
Kevät 2016
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehitys

Tekijä(t): Sakari Mursu
Opinnäytetyön nimi: REST-tietokantarajapinta mobiilisovellukselle ja web-sivustolle
Työn ohjaaja(t): Juha Alakärppä
Työn valmistumislukukausi ja -vuosi: Kevät 2016 Sivumäärä: 32

Tässä työssä suunnitellaan ja toteutetaan tietokanta mobiilisovellukselle ja web-sivustolle. Tietokannan käyttöä varten kehitettiin yhtenäinen REST-tietokantarajapinta, jota voidaan käyttää eri alustoilta. Työ aloitettiin tyhjältä pohjalta ja tavoitteena oli toimiva lopputulos.

Työn toteutus alettiin tietokannan suunnittelemisella ja REST-rajapintaan tutustumisella. Suunniteltaessa tietokantaa täytyy ottaa huomioon REST-rajapinnan vaatimat määritykset. REST-rajapintaan tutustuminen vei aikaa. Rajapinnan dokumentointi kuuluu osaksi työtä.

Rajapinta kehitettiin Node.js-ympäristössä. Node.js-moduulien avulla rajapinta kehitetään tehokkaasti ja nopeasti. Työssä käytettiin restify-moduulia, jonka avulla kehitetään toimiva ja standardien mukainen REST-rajapinta. Tietokantaa käytetään knex.js-moduulilla ja tietokannan sisältöä hallitaan bookshelf-moduulilla. Bookshelf-moduuli helpottaa tietokannan käyttöä antamalla käyttäjän luoda erilaisia relaatiomalleja.

Työn lopputuloksena syntyi toimiva REST-tietokantarajapinta ja siihen liitetty tietokanta. Tietokanta sisältää mobiilisovelluksen ja web-sivuston sisällön. Tietokanta on nykyaikainen ja helposti laajennettavissa.

REST-tietokantarajapinnan kehittämiselle on useita eri tapoja. Työssä kehitetty rajapinta on toteutettu yhdellä mahdollisella tavalla. Node.js-ympäristö kasvattaa jatkuvasti suosiotaan, mutta myös muita kilpailijoita on jo tullut. Tähän työhön Node.js-ympäristö oli sopivin.

Asiasanat: REST, API, tietokantarajapinta, palvelinsovellus, rajapinta, MySQL, Node.js, internetpalvelut

ALKULAUSE

Tämä opinnäytetyö tehtiin Weela-projektiin, josta aloituksen jälkeen tuli Welapro Oy, Oulussa. Opinnäytetyö alkoi syksyllä 2015 ja valmistui keväällä 2016. Ennen opinnäytetyön tekoa tein projektissa 30 opintopisteen edestä harjoittelua, minkä vuoksi oli hienoa saada tehdä opinnäytetyö samassa paikassa. Pystyin jatkamaan harjoitteluiden aikana aloittamaani järjestelmää. Haluaisin kiittää Miikka Kurunlahtea, joka veti projektia eteenpäin ja oli yrityksessä yhteyshenkilönäni.

Oulussa 12.5.2016

Sakari Mursu

SISÄLLYS

TIIVISTELMÄ 3
ALKULAUSE 4
SISÄLLYS 5
SANASTO 7
1 JOHDANTO 8
2 REST-ARKKITEHTUURI 9
2.1 REST-arkkitehtuuri 9
2.2 Rajoitteet 9
2.2.1 Asiakas ja palvelin 9
2.2.2 Tilattomuus 9
2.2.3 Yhtenäinen rajapinta 10
2.2.4 Välimuistin käyttäminen 10
2.2.5 Kerroksittainen järjestelmä 11
2.2.6 Ladattava koodi 11
2.3 HTTP-protokolla 11
2.3.1 Metodit 12
2.3.2 CRUD-operaatiot 13
3 NODE.JS-AJOYMPÄRISTÖ 14
3.1 Node.js lyhyesti 14
3.2 Node Package Manager 15
3.3 Restify 16
4 TIETOKANTA 17
4.1 Relaatiotietokanta 17
4.2 Tietokannan suunnittelu 18
4.3 MySQL 18
5 REST-RAJAPINNAN TOTEUTUS 19
5.1 Node.js ja käytetyt moduulit 19
5.1.1 Restify 19
5.1.2 Bookshelf ja Knex 20

5.2 Tietoturva 21
5.2.1 Pyyntöjen todentaminen 21
5.2.2 Julkinen ja salainen avainpari 21
5.2.3 Allekirjoituksen luominen 22
5.2.4 Todennuksen lähettäminen otsakkeella 22
5.3 REST-päätepisteet 23
6 TIETOKANNAN TOTEUTUS 25
6.1 Suunnittelu ja rakenne 25
6.2 Lokalisointi 25
7 TESTAUS 27
7.1 REST-rajapinnan testaaminen 27
7.2 Tietoturvan testaaminen 28
7.3 Postman 28
8 YHTEENVETO 29
LÄHTEET 31

SANASTO

API	Ohjelmointirajapinta, jolla palvelua voidaan ohjata ohjelmallisesti
HTTP	Hypertext Transfer Protocol. Webissä sijaitsevien hypermedioiden välistä kommunikaatiota ohjaava protokolla.
I/O	Input/Output. Käytetään kuvaamaan tiedon lähettämistä tai vastaanottamista.
REST	Representational State Transfer. Sovellusarkkitehtuurityyli hajautetuille hypermediajärjestelmille.
Resurssi	REST-arkkitehtuurissa käytetty termi tietolähteestä.
Sovelma	Asiakaskoneessa selaimella ajettava ohjelma.
Skripti	Ajonaikana tulkattava ohjelma, joka ajetaan komentotulkin avulla.
URI	Merkkijono, jolla kerrotaan tietyn tiedon paikka. URI voidaan määritellä tiedon paikantimena, nimenä tai kumpanakin.
web	Maailmanlaajuinen tietojärjestelmä, joka koostuu HTTP-protokollan avulla keskenään kommunikoivista komponenteista.
www	Kokoelma hypermediaa, jonka osat ovat yhdistettynä toisiinsa hyperlinkkien avulla

1 JOHDANTO

Huhtikuussa 2015 aloitin Weela-projektissa työharjoittelun. Työharjoittelun aiheena oli mobiilisovelluksen kehittäminen uudenlaisen kuntolaitteen ohjausta varten. Harjoitteluiden edetessä ilmeni tarvetta tietokannalle ja tietokantarajapinnalle. Hyvin aikaisessa vaiheessa keskustelimme esimiehen kanssa mahdollisuudesta toteuttaa ja kehittää tietokanta sekä tietokannan käytölle tarkoitettu tietokantarajapinta opinnäytetyönä.

Työn tarkoituksena on suunnitella ja kehittää mobiilisovelluksen sekä websivuston käyttöön tarkoitettu tietokanta ja tietokannan käytön helpottamista varten yhtenäinen REST-rajapinta. REST-rajapinta on hajautetuissa hypermediajärjestelmissä käytettävä arkkitehtuurityyli. RESTin tarkoituksena on mahdollistaa eri osapuolten itsenäinen kehittäminen eli tietokanta ja rajapinta voidaan kehittää erikseen, omassa tahdissaan. (1.)

Suunniteltava tietokanta ja rajapinta tulevat uudenlaisen sähkömoottorilla toimivan kuntolaitteen ohjaukseen käytettävän mobiilisovelluksen sekä kuntolaitteen kotisivun sisällön käyttöön. Tietokannan ominaisuuksilta vaaditaan, että se on oltava hyvin skaalautuva, laajennettavissa ja nykyaikainen. Tietokantaa tullaan myöhemmin laajentamaan, kun sivuston ja mobiilisovelluksen ominaisuuksia laajennetaan.

Työ aloitetaan tyhjästä ja lopputuloksena tavoitteena on toimiva tietokanta, jossa on toimiva REST-rajapinta. Tietokanta ja rajapinta toteutetaan ajettavaksi tilaajan vuokraamalle palvelimelle.

2 REST-ARKKITEHTUURI

Web-rajapinnan toteuttamista varten on nykypäivänä käytettävissä useita eri web-teknologioita. Kullakin teknologialla on omat vahvuutensa ja heikkoutensa. Tässä luvussa kerrotaan opinnäytetyössäni käytetystä REST-arkkitehtuurimallista.

2.1 REST-arkkitehtuuri

REST on lyhenne sanoille REpresentional State Transfer. REST on arkkitehtuurimalli tietoverkkojen avulla toisiinsa yhdistettäville sovelluksille. Tiedonsiirto sovellusten välillä perustuu yleensä HTTP-protokollaan. RESTin kehityksen taustalla ideana on web-protokollien, kuten HTTP ja URI, kehityksen hallitseminen. (2.)

2.2 Rajoitteet

REST-arkkitehtuuri määrittää periaatteet, joita REST-pohjaisen palvelun tulee noudattaa. Rajoitteiden tarkoituksena on taata palvelun toimivuus eri resurssien välillä. (3, s. 78.)

2.2.1 Asiakas ja palvelin

REST-järjestelmässä on selkeä jako asiakkaaseen ja palvelimeen. Palvelimen tehtävänä on asiakasohjelman lähettämien viestien vastaanottaminen ja käsittely. Asiakasohjelman tehtävänä on viestien lähettäminen oikeassa muodossa ja vastausten käsittely tarpeidensa vaatimalla tavalla. Tällaisella selkeällä järjestelmän jakamisella parannetaan käyttöliittymän siirrettävyyttä erilaisille alustoille sekä koko järjestelmän skaalautuvuutta. Jako mahdollistaa myös asiakas- ja palvelinpuolen komponenttien kehittämistä omalla tahdilla. (3, s. 78.)

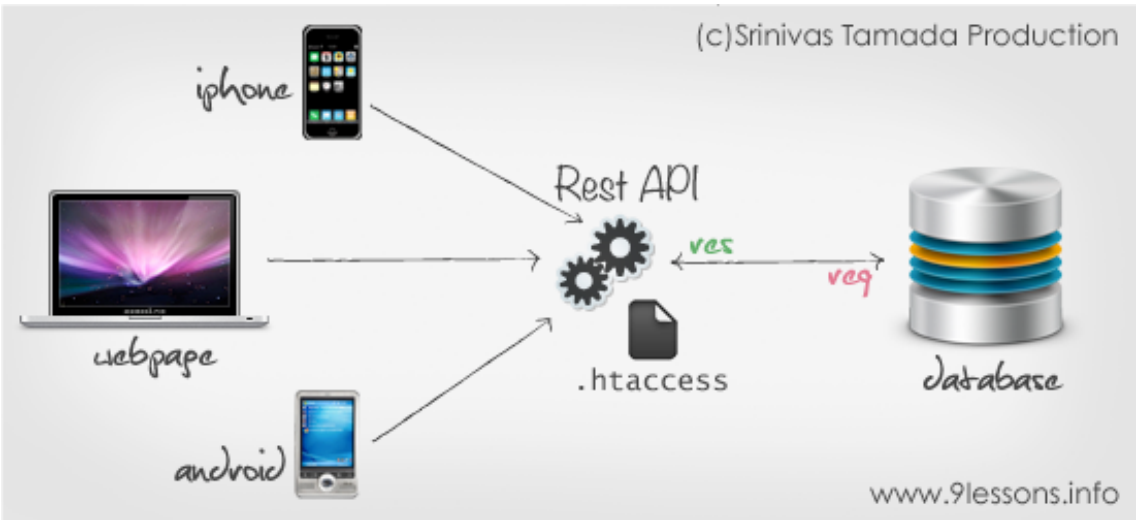
2.2.2 Tilattomuus

REST-järjestelmä on tilaton eli palvelimelle ei jää tietoa suoritetuista kyselyistä. Kyselyiden tulokset voidaan tallentaa välimuistiin turhan resurssien uudelleenkäytön vähentämiseksi. Jokainen kutsu käsitellään riippumattomana aiemmista

kyselyistä tai tuloksista. Jokaisen kyselyn tulee sisältää tarpeelliset tiedot operaation suorittamiseen. (3, s. 78–79.)

2.2.3 Yhtenäinen rajapinta

REST-rajapinnan avulla useat eri asiakasohjelmat eri alustoilta voivat käyttää samaa resurssia yhtenäisen rajapinnan kautta. Resurssien erottaminen erilleen rajapinnasta mahdollistaa yksittäisten komponenttien kehittämisen omassa tahdissaan erillään muista komponenteista. (Kuva 1.) (3, s. 81–82.)



KUVA 1. REST-rajapinta toimii yhdyskäytävänä resursseihin (7)

Yhtenäisen rajapinnan haittapuolena on suorituskyvyn heikkeneminen, koska komponenttien välillä kulkevan tiedon täytyy aina siirtyä samassa muodossa riippumatta siitä, mikä olisi paras muoto kullekin komponentille. Tiedon yhdenmukaisuutta rajoitetaan vielä neljällä lisärajoitteella: resurssien tunnistamisella, manipuloinnilla esitysten kautta, itseselitteisillä viesteillä ja hypermedian käytöllä tilakoneena. (3, s. 81–82.)

2.2.4 Välimuistin käyttäminen

Kyselyiden tulokset voidaan tallentaa välimuistiin, jos resurssi on sallittu tallennettavan välimuistiin. Näin uuden kyselyn saapuessa resurssi voidaan lähettää asiakkaalle ilman, että tieto täytyy hakea uudestaan palvelimelta. Tällä vähennetään turhaa liikennettä ja kyselyiden aiheuttamaa rasitusta palvelimella. Pal-

velimen lisäksi asiakasohjelma voi toteuttaa välimuistiominaisuuden ja siten vähentää turhaa resurssien haaskausta. (3, s. 79–81.)

2.2.5 Kerroksittainen järjestelmä

REST-järjestelmän tulee olla sellainen, että osapuolten välille voidaan asettaa palomuureja ja välityspalvelimia ilman, että asiakkaan ja palvelimen välisiä rajapintoja täytyy muokata muutosten tapahtuessa. Tämä tarkoittaa, että tietokerros ei aseta palautettavan tietoon muotoiluja, vaan välittää tiedon raakana ilman muotoiluja ja muokkauksia seuraavalle kerrokselle. Seuraavassa kerroksessa tietoon voidaan tehdä muotoilut. Sen jälkeen tieto voidaan lähettää edelleen seuraavalle kerrokselle. (3, s. 82–84.)

2.2.6 Ladattava koodi

Ladattavan koodin avulla asiakasohjelman toiminallisuutta voidaan laajentaa lataamalla koodia skirptien ja sovelmien avulla. Tämän ominaisuuden avulla asiakasohjelmista voidaan tehdä yksinkertaisempia ja kevyempiä, koska valmiiksi määriteltyjä ominaisuuksia on vähemmän. Toimintojen lataaminen parantaa järjestelmän laajennettavuutta. Tämä rajoite on kuitenkin vapaaehtoinen. (3, s. 84.)

2.3 HTTP-protokolla

HTTP-protokolla on sovellustason protokolla. Sen käyttötarkoitus on yleisen yhteistyöhön perustuvan hypermediatietojärjestelmien väliseen kommunikointiin. HTTP-protokollaa on käytetty WWW:n (World Wide Web) yhteydessä vuodesta 1990 asti. Nykyaikainen HTTP/1.1-protokolla sallii kommunikoinnissa käytettävän myös MIME-tyyppistä tietoa ja pyyntö sekä vastaus semantiikkaa. MIME-tyyppinen tieto voi sisältää metadataa, eli liitännäistietoa. (8.)

HTTP-protokolla perustuu pyyntöihin ja vastauksiin. Asiakasohjelma lähettää palvelimelle pyynnön, joka sisältää pyydettävän metodin, protokollan version ja pyynnön URI:in (Uniform Resource Identifier). Näiden tietojen lisäksi pyyntöön lisätään MIME-tyyppiseen viestiin sisällettynä asiakasohjelman tiedot ja mahdollisen lähetettävän viestin sisällön. Palvelin vastaa pyyntöön lähettämällä asia-

kasohjelmalle vastauksen onnistumisesta tai epäonnistumisesta. Palvelimen lähettämässä vastauksessa on myös mukana MIME-tietoa palvelimesta ja metadatasta sekä mahdollinen viesti, jonka palvelin lähettää asiakasohjelmalle vastauksen yhteydessä. (8.)

REST-arkkitehtuurimallia noudattavat sovellukset kommunikoivat HTTP-protokollan välityksellä. Tästä syystä REST-järjestelmä voi käyttää hyväksi HTTP-protokollan tarjoamia metodeja (kuten GET, POST, PUT, DELETE jne.) Näiden metodien avulla voi REST-järjestelmä käyttää CRUD-operaatioita (Create, Read, Update, Delete). (2.)

2.3.1 Metodit

Kun HTTP-protokollan avulla tehdään pyyntö, lähetetään pyynnön mukana metodi, jota halutaan käyttää. Määritelty HTTP-protokollan metodi määrittelee, kuinka palvelin käsittelee pyynnön. (8.)

GET

GET-metodilla haetaan resurssi palvelimelta. Haettavaa resurssia ei muuteta, vaan haettava resurssi haetaan sellaisenaan, kun se palvelimelta löytyy. Haettava resurssi määritellään käytetyssä URI:ssä. (8.)

POST

POST-metodilla luodaan uusi resurssi palvelimelle. Uusi resurssi luodaan tiedosta, joka lähetetään POST-metodin yhteydessä. Käytetyssä URI:ssä määritellään, minne tieto lähetetään palvelimella. (8.)

PUT

PUT-metodilla päivitetään palvelimella oleva yksittäinen resurssi. PUT-metodilla ei ole tarkoitus luoda uutta tietoa, mutta se on mahdollista. Lähetettävän resurssin pitäisi olla muuten identtinen versio palvelimella sijaitsevasta resurssista, paitsi muokatun tiedon kohdalta. Jos käytetty URI ei osoita palvelimella olevaan resurssiin, luo palvelin kyseiseen URI:hin uuden resurssin. Tässä tapauksessa palvelin ilmoittaa asiakasohjelmalle, että pyyntö loi uuden resurssin palvelimelle eikä muokannut valmiiksi olemassa olevaa resurssia. Lyhyesti sanottuna PUT-metodilla voidaan päivittää olemassa olevaa tietoa. REST-arkkitehtuurissa ei suositella PUT-metodin käyttöä uuden resurssin luomiseen tai osittaiseen päivittämiseen. Näitä toimintoja tulisi käyttää POST- ja PATCH-metodeilla. (8.)

DELETE

DELETE-metodilla pyydetään palvelinta poistamaan URI:in osoittama resurssi. Asiakasohjelma ei kuitenkaan voi varmuudella tietää poistettiinko resurssi. Palvelin voidaan ohjelmoida suorittamaan jokin muu toiminto, kun DELETE-metodia käytetään. (8.)

2.3.2 CRUD-operaatiot

CRUD tulee sanoista Create, Read, Update ja Delete. Se esittää neljää relaatiotietokantasovellusten perusfunktiota. Jokainen metodi voidaan yhdistää SQL-kielen komentoon tai HTTP-protokollan metodiin. CRUDia käytetään yleensä REST-rajapinnan yhteydessä tiedonkäsittelyyn. (9, s. 381–382.)

CRUD-operaatioita käytetään REST-arkkitehtuurissa yhdessä tietokannan kanssa. Perusfunktion nimi kertoo itsessään, millaisen toiminnon funktio tekee. Create-funktion avulla luodaan uusi resurssi tietokantaan ja vastaavasti Read-funktiolla voidaan lukea tietokannassa olevia resursseja. Update-funktiolla tietokannassa oleva resurssi päivitetään, ja tietyissä tietokanta ympäristöissä Update-funktio luo uuden resurssin tietokantaan, jos päivitettävää tietoa ei löydy. Delete-funktiolla poistetaan resurssi. (9, s. 381–382.)

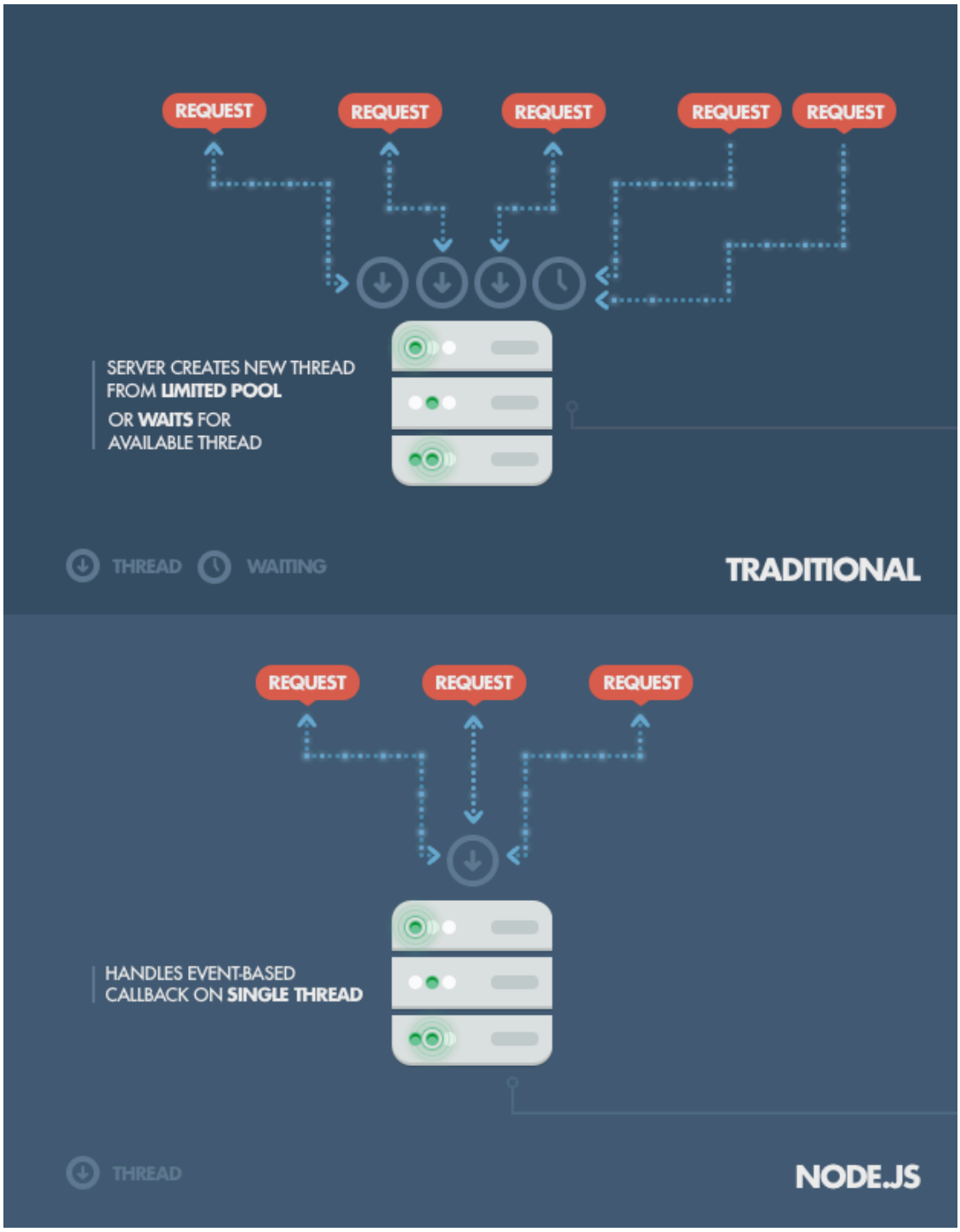
3 NODE.JS-AJOYMPÄRISTÖ

Node.js on Ryan Dahlin vuoden 2009 lopulla julkistama web-teknologia. Yleisesti JavaScriptiä on käytetty vain selaimessa, mutta Node.js toi JavaScriptin myös palvelinpuolelle. Nykyään JavaScriptin käyttö palvelimissa on kasvanut huomattavasti ja siitä on tullut yksi suosituimmista ohjelmointikielistä. (10.)

3.1 Node.js lyhyesti

Node.js käyttää Googlen kehittämää V8 JavaScript moottoria ja järjestelmää nimeltä event loop. Tämä mahdollistaa sen, että Node.js pystyy lähettämään ja käsittelemään tietoa hyvin nopeasti. V8:n avulla Node.js ymmärtää JavaScript-ohjelmointikieltä. Ymmärtämällä JavaScript-ohjelmointikieltä Node.js:n ei tarvitse ensin kääntää koodia ymmärtääkseen sitä, vaan se voi käyttää JavaScriptiä suoraan. Tästä syystä Node.js on todella nopea. JavaScriptin käyttö palvelinpuolella ohjelmointikielenä tarkoittaa myös sitä, että kehityksen aikana ei tarvitse kirjoittaa erillisiä ohjelmakoodeja palvelimelle ja asiakasohjelmalle, mikä mahdollistaa entistä nopeamman ohjelmistokehityksen. (11.)

Pääideana Node.js:ssä on olla esteetön ja tapahtumapohjainen I/O-järjestelmä. Esteetön I/O-järjestelmä tarkoittaa, että mikään suoritettava käsky ei estä toisen käskyn ajoa. Tällaisesta arkkitehtuurista johtuen Node.js pysyy tehokkaana data-intensiivisissä ohjelmissa. Verrattuna perinteiseen verkkopalvelutekniikkaan, missä jokainen yhteys saa uuden säikeen käyttäen paljon RAM-muistia, Node.js käyttää vain yhtä säiettä tukemaan useita, jopa kymmeniä tuhansia, yhteyksiä. Tästä johtuen Node.js on erittäin hyvin skaalautuva. Kuvassa 2 näkyy perinteisen ja Node.js:n verkkopalvelutekniikan ero. (12.)



KUVA 2. Perinteisten ja Node.js-yhteyksien ero (12)

3.2 Node Package Manager

NPM eli Node Package Manager on ohjelma, jonka avulla Node.js:n käyttö on helpompaa. NPM:llä on kaksi käyttötarkoitusta. Se toimii verkossa säilytyspaikkana julkistetuille avoimenlähdekoodin Node.js-projekteille. Toiseksi se on ko-

mentorivityökalu kyseisten projektien eli pakettien asentamisessa, versionhallinnassa ja riippuvuussuhteiden hallinnassa. Julkaistuja paketteja löytyy valtava määrä. NPM:n avulla julkaistuja paketteja voi etsiä ja halutessaan asentaa yhdellä komennolla. (13.)

Asennettava paketti voi käyttää hyväkseen muita julkaistuja paketteja, tällaisessa tilanteessa NPM pitää asennettujen pakettien riippuvuussuhteet kunnossa ja asentaa kaikki tarvittavat paketit automaattisesti. Tämän avulla Node.js-pakettien hakeminen, asentaminen ja päivittäminen on todella helppoa. (13.)

3.3 Restify

Restify on ladattava sovelluskehys. Restifyä käytetään rakennettaessa REST-palvelua, joka seuraa REST-arkkitehtuurin asettamia määritelmiä. Se pohjautuu Express.js-sovelluskehykseen, mutta on käytännössä tarkoitettu vain API:a käyttävien web-sovellusten ohjelmointiin Node.js-ympäristössä. Express.js:ää sen sijaan käytetään web-sivujen julkaisuun. (14.)

4 TIETOKANTA

Tietokannan yleisin määritelmä on kokoelma loogisesti yhteenkuuluvaa tietoa, joka on järjestelty siten, että sitä on helppo ja vaivaton lukea. Tietokannan sisältämien tietojen on oltava luettavissa, päivitettävissä ja muokattavissa tarvittavin tavoin. Tietokannat voidaan jaotella niiden sisältämien tietojen perusteella. Yleensä tietokannat ovat täysin teksti- ja numeropohjaisia taulukoita. Tietokanta voi sisältää myös erityyppistä tietoa, kuten kuvia ja olioita. Erityyppisten tietojen yhdisteleminen tietokannassa on myös mahdollista. (15, s. 4–5.)

Tietokannan hallinta tapahtuu hallintajärjestelmällä, jonka avulla voidaan syöttää tietokantakäskyjä esimerkiksi SQL-kielellä. Ilman tietohallintajärjestelmää tietokantojen käyttäminen on monimutkaista, aikaa vievää ja vaikeaa. (15, s. 4–5.)

4.1 Relaatiotietokanta

Relaatiotietokannaksi kutsutaan tietokantaa, joka koostuu toisiinsa yhdistetyistä tiedoista. Jokaisen tiedon välillä on relaatio, eli yhteys, johonkin toiseen tietoon. Relaatiomallin perusideana on tallentaa yksittäinen tieto vain kerran. Jos tietoa pitää toistaa, luodaan yhteys olemassa olevaan tietoon. Tällöin tiedon turha toisto saadaan poistettua. Tämä parantaa tietokannan suorituskykyä nopeuttamalla tiedonhakua ja pienentämällä tietokannan kokoa. (15, s. 7–8.)

Relaatiotietokanta koostuu taulukoista. Taulukoiden välillä on erilaisia viitteitä, joita voidaan kutsua myös avaimiksi. Viitteiden avulla voidaan loogisesti nähdä mitkä tietokannan taulut ovat relaatiossa toisiinsa. Viitteiden avulla tietojen päivittäminen, hakeminen ja yhdisteleminen on monipuolista, mikä mahdollistaa monimutkaisten hakujen tekemisen. Hyvin suunniteltu relaatiotietokanta on yksinkertainen käyttää ja tehokas. (15, s. 7–8.)

Tietokannan muodostavat taulukot ovat kuin taulukkolaskennan taulut. Rivejä kutsutaan tietueiksi ja sarakkeita kentiksi. Yksi tietue sisältää yhden tiedon jokaisesta kentästä. Kenttien sisältämä tietotyyppi määritellään tietokannassa,

eikä kenttä hyväksy muun tyyppistä tietoa. Kenttä voidaan jättää tyhjäksi, jos asetukset sen sallivat. (15, s 8–9.)

4.2 Tietokannan suunnittelu

Tietokannan suunnittelulla on erittäin tärkeä rooli nykyaikaisten sovellusten operoinnin ja kehitysprosessien onnistumisessa. Huonosti suunniteltu tietokanta estää sovelluksen onnistumisen. Hyvin suunniteltu tietokanta nopeuttaa ohjelmointia ja luo vahvan perusrakenteen sovellukselle. Monimutkaisia ja laajoja tietokantoja suunniteltaessa tulee ottaa huomioon mahdollinen tietokannan laajentaminen, vaatimusmäärittelyt ja myös tietokannan fyysinen suunnittelu. Tietokanta mallinnetaan jollakin kuvaustekniikalla, esimerkiksi UML-kielellä. (15, s. 20.)

Hyvin suunnitellun tietokannan ominaisuuksia ovat selkeys ja ymmärrettävyys. Tietokannan rakenne on yksinkertainen ja tietokannasta on helppo hakea tietoa. Kaikki kyselyissä tarvittavat tiedot löytyvät loogisilta paikoilta ja tietorakenteet ovat selkeitä. (15, s. 21.)

4.3 MySQL

Oraclen kehittämä avoimen lähdekoodin relaatiotietokanta on tarjolla lähes kaikilla palvelintarjoajilla ja ilmainen käyttää GPL-lisenssillä (GNU General Public License). MySQL on erittäin suosittu tietokanta web-sovelluksissa sekä palveluissa. Esimerkiksi YouTube, Wikipedia, Facebook ja Yahoo käyttävät MySQL-relaatiotietokantoja. (16.)

5 REST-RAJAPINNAN TOTEUTUS

Opinnäytetyössä ohjelmoitiin REST-rajapinta MySQL-tietokannalle. Rajapintaa käytetään mobiilisovelluksella ja web-sivustolla. Esimerkiksi mobiilisovellus voi hakea rajapinnan avulla käyttäjätiedot tietokannasta. Mobiilisovellus lähettää rajapinnalle pyynnön käyttäen tiettyä URI:tä (Uniform Resource Identifier). Vastauksena mobiilisovelluksen pyyntöön rajapinta lähettää mobiilisovellukselle käyttäjätiedot. Opinnäytetyössä toteutettiin myös REST-rajapinnan käyttämä MySQL-tietokanta.

5.1 Node.js ja käytetyt moduulit

REST-rajapinta toteutettiin käyttäen Node.js-ympäristöä. Node.js on avoimelähdekoodin järjestelmäriippumaton ympäristö palvelimilla ajettavien websovellusten kehittämiseen. Opinnäytetyössä käytettiin Node.js:n versiota 2.14.7. Palvelinsovelluksen rajapinnan ohjelmoinnissa käytettiin apuna restify-moduulia.

5.1.1 Restify

Restify on tarkoitettu REST-rajapinnan ohjelmoimiseen ja se sisältää useita apuja helposti ylläpidettävän REST-rajapinnan luomiseen. Tässä työssä käytettiin restifyn versiota 4.0.3.

Restify-moduulin käyttö määritellään palvelimen alustuksen yhteydessä. Aluksi määritellään palvelimen tiedot ja asetukset. Tämän jälkeen otetaan käyttöön restifyn käyttämät moduulit ja lopuksi käynnistetään palvelin. (Kuva 3.)

```
1  // Creating the server with given configs.
2  var server = restify.createServer({
3    version: config.version,
4    name: config.name,
5    log: log
6  });
7
8  // Define which restify modules to use.
9  server.use(restify.sanitizePath());
10 server.use(restify.gzipResponse());
11 server.use(restify.queryParser());
12 server.use(restify.bodyParser());
13 server.use(restify.requestLogger());
14
15 ...
16
17 // Start server with the given configuration.
18 server.listen(config.port, config.address, function() {
19   console.log('Listening %s on %s:%s (%s)', server.name, server.address().address,
20     server.address().port, config.environment);
21 });
```

KUVA 3. Restify-moodulin alustaminen ja käyttöönotto server.js-tiedostossa.

5.1.2 Bookshelf ja Knex

Node.js:n kanssa voi käyttää erilaisia moduuleja, jotka helpottavat Node.js ympäristössä sovellusten ohjelmointia. Opinnäytetyössäni käytin bookshelf-moduulia tietokannan objektien relaation kartoitukseen (ORM). ORM:lla tarkoitetaan normaalisti yhteen sopimattoman tiedon muuttamista yhteensopivaksi. Bookshelf-moduulin avulla loin jokaisesta tietokannasta haettavasta tiedosta mallin, jonka avulla moduuli hakee mallin määrittämät tiedot tietokannasta. Bookshelf-moduulista käytin versiota 0.9.1.

Tietokannan ja palvelinsovelluksen väliseen yhteydenpitoon käytin knex.js-moduulia. Knex.js:n avulla voidaan luoda yhteyksiä tietokantaan ja sitä voidaan käyttää SQL-kyselykielen avulla tehtäviin tietokantahakuihin. Bookshelf-moduuli on rakennettu käytettäväksi knex.js-moduulin kanssa. Bookshelf-moduulin alustuksessa täytyy mukaan syöttää knex-objekti (Kuva 4).

```
/*
 * Initialises knex and bookshelf for making connection to database
 */
var config = require('../config/database.json');
var knex = require('knex')(config);
var bookshelf = require('bookshelf')(knex);

exports.bookshelf = bookshelf;
```

KUVA 4. Knex.js- ja bookshelf-moduulien alustaminen käytettäväksi yhdessä

5.2 Tietoturva

Tietoturva oli yksi suurimmista asioista, joka täytyi ottaa huomioon ja johon kului suurin osa opinnäytetyöhön varatusta ajasta. Tutkin useita erilaisia pyyntöjen todentamiseen käytettyjä keinoja, aina yksinkertaisista salasana ja käyttäjätunnut -yhdistelmistä kolmannen osapuolen palveluihin. Lopulta päädyin käyttämään Amazon Web Services -palvelun (AWS) esittelemää tapaa. AWS:n julkinen dokumentointi ei paljasta yksityiskohtia, kuinka todentaminen tapahtuu, mutta se antoi minulle idean, kuinka todennuksen voisi tehdä. Käyttämällä apuna AWS:n julkista dokumentointia pystyin luomaan Weelalle oman todennusjärjestelmän. AWS:n julkinen dokumentaatio löytyy osoitteesta http://aws.amazon.com.

5.2.1 Pyyntöjen todentaminen

Lyhyesti selitettynä asiakasohjelman pyynnöt valtuutetaan siten, että asiakasohjelma luo määritetyllä tavalla allekirjoituksen, jonka se lähettää pyynnön mukana palvelimelle. Ennen vastaanotetun pyynnön suorittamista palvelin luo allekirjoituksen pyynnön mukana tulleesta datasta samalla määritetyllä tavalla kuin asiakasohjelmakin. Lopuksi allekirjoituksia verrataan, ja jos ne täsmäävät, todetaan, että pyyntö voidaan käsitellä.

5.2.2 Julkinen ja salainen avainpari

Pyyntöjen todennuksessa käytetään avainparia. Avainpari koostuu julkisesta- ja salaisesta avaimesta (WLASecretKey ja WLAPublicKey). Jokaisella käyttäjällä on oma avainparinsa, joita voidaan käyttää heidän pyyntöjensä todentamiseen.

On tärkeää tietoturvallisuuden kannalta, että salaista avainta ei missään tapauksessa lähetetä verkon yli. Tällöin kolmannen osapuolen on mahdollista napata salainen avain ja käyttää sitä todentamaan oman haitallisen pyyntönsä. Verkon yli voidaan lähettää julkinen avain, sillä siitä ei ole mitään hyötyä, jos ei tiedetä myös avainparin salaista avainta. Julkisella avaimella ilmoitetaan palvelimelle, kenen käyttäjän salaista avainta käytettiin pyynnön allekirjoituksen luomiseen. Palvelin käyttää julkista avainta vain löytääkseen tietokannasta kyseisen avainparin salaisen avaimen.

Jos salainen avain on vaarannettu tai murrettu sen avainpari täytyy välittömästi poistaa käytöstä. Poistamalla avain käytöstä estetään kyseistä avainparia käytettyjen pyyntöjen todentaminen. Täten avainparia ei voida käyttää enää ollenkaan.

5.2.3 Allekirjoituksen luominen

Allekirjoitus luodaan käyttämällä HMAC-SHA256-algoritmia. HMAC-SHA256 on Internet Engineering Task Forcen määrittämä RFC 4868 -algoritmi. Sitä käytetään datan alkuperän ja yhtenäisyyden todentamiseen (17). Kyseinen algoritmi ottaa vastaan kaksi merkkijonoa, avaimen ja viestin. Weela REST -järjestelmän todentamisessa käytetään avaimena salaista avainta (WLASecretKey) ja viestinä UTF-8 -koodattua StringToSign-merkkijonoa. StringToSign on lähetettävästä datasta koottu merkkijono

Käyttämällä HMACH-SHA256 -algoritmia salaisella avaimella (WLASecretKey) ja allekirjoituksella (StringToSign) luodaan allekirjoitus, joka on Base64-pohjainen merkkijono. Tärkeä asia huomioida on, että kaikki todennuksessa luodut merkkijonot ovat Base64-pohjaisia merkkijonoja, jotta virheellisiä todennuksia ei aiheutuisi merkkijonojen muotoilusta.

5.2.4 Todennuksen lähettäminen otsakkeella

Kun HTTP-protokollalla lähetetään palvelimelle pyyntö, pyynnön mukana lähetetään myös otsakkeita. Otsakkeet ovat HTTP-protokollan avulla lähetettävien viestien alussa olevia tunnisteita. Otsakkeet antavat tietoa dokumentista, palvelimesta, lähetysajankohdasta yms. (18.)

Pyynnön mukana todentamiseen tarvittava julkinen avain ja allekirjoitus lähetetään Authorization-otsakkeen sisällä. Authorization-otsakkeessa lähetetään samalla myös todennuskaavan tunniste, jotta palvelin osaa käyttää luomaani Weela-todennuskaavaa. Authorization-otsake muotoillaan seuraavasti:

Authorization: WEELA WLAPublicKey:Signature

WLAPublicKeyn tilalle laitetaan käyttäjän julkinen avain ja Signaturen tilalle lähetettävä allekirjoitus. Jokainen pyyntö tulee todentaa tällä tavalla, käyttäen Authorization-otsaketta.

5.3 REST-päätepisteet

REST-rajapintaan on määritelty päätepisteitä (end-point), jotka esittävät web-selaimella käytettäviä osoitteita (URL). Päätepisteiden avulla mobiilisovellus voi käyttää haluttua toimintoa. Mobiilisovellus voi esimerkiksi hakea tietokannasta harjoituslistan käyttämällä päätepistettä, jonka toimintona on palauttaa harjoituslista. Määrittelin harjoituksiin liittyvien toimintojen päätepisteet taulukon 1. mukaisesti.

TAULUKKO 1. REST-päätepisteet

Metodi	**Päätepiste**	**Toiminto**
GET	/exercises	Palauttaa kaikki tietokannassa olevat harjoitukset.
GET	/exercises/:exercise_id	Palauttaa täsmäävän harjoituksen tyhjän kokoelman sisällä
POST	/exercises	Luo uuden harjoituksen lähetetystä datasta.
POST	/exercise/update	Päivittää täsmäävän harjoituksen kirjoittamalla sen päälle lähetetyillä tiedoilla
POST	/exercise/patch	Päivittää täsmäävän harjoituksen vain niillä tiedoilla, jotka on lähetetty
POST	/exercise/delete/:id	Poistaa täsmäävän harjoituksen

Tietokannasta voidaan hakea lista harjoituksista lähettämällä kuvan 5 mukainen pyyntö palvelimelle. Kun haetaan tiettyä harjoitusta, käytetään URI:tä /exercises/:exercise_id. URI:in jälkimmäinen osa exercise_id määrittelee haettavan harjoituksen tunnisteen. Kuvassa 5 tunnisteena on 1. Pyyntö tehdään käyttämällä yksittäisen harjoituksen hakemiseen tarkoitettua URI:tä. Pyynnön mukaan laitetaan todennukseen tarvittavat otsakkeet (todennus, kieli ja päivämäärä). Näiden tietojen palvelin päättää suoritetaanko pyyntö. Jos todennus on hyväksytty palvelin listaa harjoituksen, jonka tunniste on 1.

```
GET /exercises
Date: Fri, 20 Nov 2015 12:31:10 GMT
Authorization: WEELA 3ecc735b1cd33050c5cc:sk8BH+La7eJpDzJebKPrPdaSmq9hO366kpKMOZ6q89Y=
Language: fi
```

KUVA 5. Yksittäisen harjoituksen noutamiseen käytettävä pyyntö.

Jokainen päätepiste toimii samalla periaatteella kuin ylempi esimerkki. Kun tallennetaan tietoa tietokantaan rajapinnan avulla, lähetetään pyynnön mukana todennukseen tarvittavien tietojen lisäksi myös JSON-muodossa oleva objekti. Tämä JSON-objekti sisältää tietokantaan tallennettavan tiedon. Tallennettava tieto määräytyy sen mukaan mitä tallennetaan. Esimerkiksi uuden käyttäjän luomisessa lähetetään eri tietoa kuin harjoitusta lisättäessä. Päivitettäessä tietoa tietokannassa lähetetään päivitettävä tieto sekä päivitettävän tiedon tunniste.

6 TIETOKANNAN TOTEUTUS

Tietokantaan oli tarkoitus alun perin luoda valmiiksi taulut ja tietueet verkkosivuston sisällölle. Suunnitelma kuitenkin muuttui opinnäytetyön alkuvaiheessa ja verkkosivun sisältöä ei vielä tietokantaan laitettu. Opinnäytetyössäni tietokanta suunniteltiin tukevan vain opinnäytetyön aloituspalaverissä määriteltyjä mobiilisovelluksen käyttöön tarvittavia tietoja.

6.1 Suunnittelu ja rakenne

Tietokannan suunnittelu ja rakentaminen lähti liikkeelle aloituspalaverista, jossa määriteltiin yhdessä yrityksen edustajien kanssa mitä tietokantaan tulisi tallentaa. Päädyimme rakentamaan tietokannan sellaiseksi, että se tukisi olemassa olevan mobiilisovelluksen sisältöä.

Työn lopulla tauluja oli muodostunut 23 kappaletta. Taulujen määrä nousi ennakoitua suuremmaksi, mistä johtui lokalisointiin tarvittavista tauluista. Jokainen tietokannan taulu sisältää tietoa vain yhdestä kohteesta. Esimerkiksi käyttäjätaulu sisältää pelkästään käyttäjän tietoa ja käyttäjään liittyvät muut tiedot, kuten asetukset, ovat omissa tauluissaan.

Suunnittelin taulut siten, että keräsin ensin tallennettavat tiedot muistioon ja vaiheittain normalisoin tiedot niin, että ne voidaan tallentaa seuraten ehjää tallennusta. Normalisoimalla tietokannan rakenteen varmistin sen, että tiedon saaminen tietokannasta on tehokasta ja samaa ei tallennettaisi useaan kertaan.

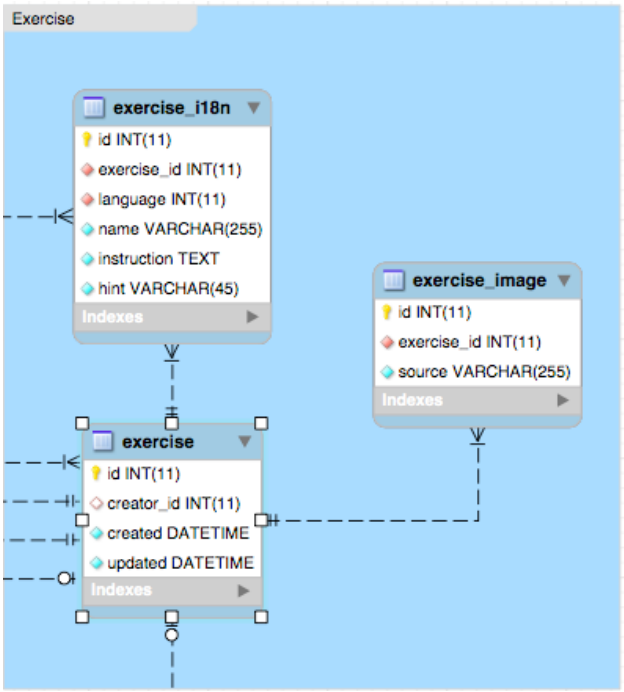
6.2 Lokalisointi

Lokalisoinnista käytetään termiä I18N, joka tulee lokalisoinnin englannin kielisestä termistä internationalization. Sanassa internationalization on i:n ja n:n välillä 18 kirjainta. (19.)

Koska mobiilisovelluksen ja verkkosivuston on suunniteltu tukevan useaa eri kieltä, tulevaisuudessa täytyi tietokannan myös tukea useita eri kieliä. Toteutin lokalisoinnin lisäämällä jokaiselle taululle, johon tallennetaan käännettävää tietoa, lisätaulun lokalisointia varten. Tämä lisätaulu sisältää kaikki tietueet, jotka

voidaan kääntää, ja itse päätaulu sisältää kaiken sellaisen, mitä ei voida kääntää.

Esimerkiksi harjoitukset tallennetaan tietokantaan kuvan 6 mukaisesti. Harjoitusten päätaulu on nimeltään exercise. Tähän tauluun tallennetaan harjoituksiin liittyvää yleistä tietoa, jota ei ole tarvetta lokalisoida. Kaikki käännettävät tiedot laitetaan exercise_i18n-tauluun. Tähän tauluun myös tallennetaan tunniste, joka määrittelee käännöksen kielen. Tietokannasta voidaan sitten hakea harjoitusten tiedot määrittelemällä hakuparametreihin suodattimeksi kyseinen kielen tunniste. Kaikki tietokannan taulut, joiden tiedot täytyy lokalisoida käyttävät tätä samaa periaatetta.



KUVA 6. MySQLWorkbench kuvankaappaus harjoitus-tauluista.

Lokalisoinnin kieli määritellään jokaisessa käännettävässä taulussa viiteavaimella. Uusien kielien lisääminen tietokantaan onnistuu helposti lisäämällä tuettava kieli language-tauluun, jonka jälkeen lokalisointitauluihin voidaan lisätä uusi käännös kyseisellä kielellä.

7 TESTAUS

Testauksella varmistetaan, että REST-rajapintasovellus toimii määritetyllä tavalla ja on toimiva. Opinnäytetyössäni en ehtinyt toteuttaa täyttä testausta, vaan sovellusta ehdittiin testata vain paikallisesti. Verkon yli julkaistua rajapintaa en ehtinyt testata opinnäytetyölle varatun ajan loputtua kesken sekä palvelimen tietokantaongelmien vuoksi. Palvelimen tietokanta ei ollut päivitetty yhteensopivaan versioon. Yksi suuri testattava alue sovelluksesta on tietoturva. Kolmannen osapuolen pääsyn estäminen järjestelmään on erittäin tärkeää. Tietoturvaa testasin kehityksen yhteydessä paikallisesti hyvinkin paljon ja paikalliset testaukset osoittivat, että tietoturva toimii odotetusti.

7.1 REST-rajapinnan testaaminen

REST-rajapintaa testataan luomalla useita erilaisia pyyntöjä, jotka sitten lähetetään palvelimelle. Testattavien pyyntöjen muuttujia muutetaan ja laitetaan tahallaan vääränlaisiksi. Tällä tavalla testataan, että rajapinta ei hyväksy vääränlaista tietoa tai suorita pyyntöä jossa todentaminen ei ole oikein. Pyynnöt voidaan suunnitella vaikka tekstitiedostoon kuvan 7 mukaisesti. Sen jälkeen ne voidaan suorittaa palvelimella esimerkiksi käyttämällä Postman-sovellusta.

```
GET /users/1
Date: Fri, 20 Nov 2015 12:31:10 GMT
Authorization: WEELA 3ecc735b1cd33050c5cc:1MRKlKSpQQ18FH1lW2lWvpYglzBDKrp6X1coRKoVECo=
Language: fi
```

KUVA 7. Toimiva palvelimelle lähetettävä pyyntö käyttäjätietojen lukemiseen.

Pyyntöjen muuttujia on hyvä antaa väärässä muodossa, jättää kokonaan pois ja kokeilla eri arvoja. Kuvassa 8 näkyy oikeanlaisessa muodossa oleva pyyntö uuden harjoituksen luomiseen tietokantaan. Kyseistä pyyntöä voidaan testata esimerkiksi jättämällä name-muuttuja kokonaan pois tai muuttamalla Content-type -muuttuja eri muotoon. Muuttamalla Content-type -muuttujaa testataan hyväksyykö rajapinta vääränlaista sisältötyyppiä ja jättämällä muuttuja kokonaan pois nähdään toimiiko rajapinnan viestin sisällön tarkistaminen.

```
POST /exercises
Date: Fri, 20 Nov 2015 12:31:10 GMT
Authorization: WEELA 3ecc735b1cd33050c5cc:1MRKlKSpQQ18FH1lW2lWvpYglzBDKrp6X1coRKoVECo=
Language: fi
Content-Type: application/json
{
  "exercise_id": 7,
  "language": 1,
  "name": "Uusi harjoitus",
  "instruction": "Tee sitä ja tätä",
  "hint": "Älä tee tuota!",
  "creator_id": 1
}
```

KUVA 8. Toimiva uuden harjoituksen luomiseen käytettävä pyyntö.

7.2 Tietoturvan testaaminen

Tietoturvaa testataan pyyntöjen todentamiseen tarvittavia muuttujia muokkailemalla ja yrittämällä saada rajapinta hyväksymään vääränlaiset pyynnöt. Testauksessa muutetaan pyyntöjen aikaleimoja, allekirjoituksia ja julkisia avaimia. Luomalla lähetettävästä tiedosta oikea allekirjoitus ja muuttamalla sen jälkeen lähetettävää tietoa voidaan testata tunnistaako rajapinta muokattua tietoa. Toimivaa pyyntöä kannattaa testata myös lähettämällä se kolmannelta osapuolelta alkuperäisen lähettäjän sijaan.

7.3 Postman

Postman on sovellus, jonka avulla voidaan suorittaa pyyntöjä palvelimelle. Sen käyttötarkoitus on rakentaa, testata ja dokumentoida rajapintoja. Sen käyttö on nopeampaa kuin testien ohjelmoiminen. Postmanissa on monia ominaisuuksia, jotka nopeuttavat testaamista. Pyynnöt tallentuvat historiaan, ja niitä voidaan käyttää helposti uudestaan. Uusien pyyntöjen luominen on nopeaa erilaisten auttavien toimintojen avulla, lisäksi käyttöliittymä on selkeä ja nopea oppia. Postmanilla voidaan myös ajaa erilaisia scriptejä, jotka voivat tehdä monimutkaisempia testauksia. Todentamiseen Postmanissa löytyy sisäänrakennettuja avustavia metodeja. Testauksen automatisointi on helpompaa. Postman on saatavissa osoitteessa http://www.getpostman.com/.

8 YHTEENVETO

Työn tarkoituksena oli suunnitella sekä toteuttaa REST-tietokantarajapinta ja relaatiotietokanta. Tietokannan ja rajapinnan käyttötarkoituksena oli mobiilisovelluksen ja web-sivuston sisällön tallentaminen yhtenäiseen tietokantaan. Tietokanta sekä rajapinta tulivat käytettäväksi usealla eri alustalla. Työ aloitettiin tyhjältä pohjalta ja tavoitteena oli toimiva lopputulos. Tavoitteena oli myös saada palvelinsovellus sellaiseen kuntoon, että kehitystä voidaan jatkaa helposti.

Työ alkoi tietokannan suunnitelulla ja REST-rajapintaan tutustumisella. Alun hitauden jälkeen rajapinnan kehittäminen nopeutui. Työtä hidasti tietoturvan tärkeys. Tietoturvaan jouduin panostamaan todella paljon aikaa ja se vei enemmän työtunteja kuin osasin aluksi odottaa. Tehdyn työn dokumentointiin kului myös aikaa, mutta jälkikäteen ilmeni kuinka tärkeä hyvä ja perusteellinen dokumentointi on, etenkin kehityksen jatkamisessa ja opinnäytetyön kirjoittamisessa.

Rajapinta kehitettiin Node.js-ympäristössä. Node.js-moduulien avulla rajapinnan kehittäminen oli nopeaa ja vaivatonta. Työssä käytetty restify-moduuli oli hyvä ratkaisu REST-rajapinnan luomiseen. Sen sisältämät avut ja kirjastot vähensivät turhaa ohjelmointia. Valmiita kirjastoja ja metodeja käyttämällä toteutus nopeutui. Tietokannan käyttämiseen käytettiin knex.js-moduulia ja tietokannan sisällön hallitaan bookshelf-moduulilla. Bookshelf-moduuli helpotti tietokannan käyttöä mahdollistamalla erilaisten relaatiomallien luomisen ja käytön.

Lopputuloksena syntyi annettujen vaatimusten mukainen REST-tietokantarajapinta sekä siihen liitetty tietokanta. Tietokanta sisältää mobiilisovelluksen käyttöön tarkoitetut taulut. Alkuperäisten suunnitelmien mukaisia tauluja web-sivuston käyttöön ei lisätty. Rajapinta sekä tietokanta ovat nykyaikaisia ja helposti laajennettavissa sekä kehitettävissä. Alkuperäisestä tavoitteesta poiketen web-sivuston käyttöön tarkoitetut toiminnot ja tietokannan taulut jäivät pois. Aika ja resurssit eivät riittäneet niiden lisäämiseen.

REST-tietokantarajapinnan kehittämiselle on useita eri tapoja. Käytetty rajapinta on toteutettu yhdellä mahdollisella tavalla. Palvelinympäristön valitseminen täy-

tyy tehdä aina tarpeen ja kohteen mukaan. Tässä työssä Node.js-ympäristö oli sopivin. Node.js kasvattaa jatkuvasti suosiotaan, mutta myös muita kilpailijoita on tullut. Dokumentaatio on todella hyvä, mikä helpottaa kehittämistä.

Rajapinnan jatkokehityksessä tietoturvaa voidaan parantaa kehittämällä Steam-pelialustan sisäänkirjautumisen tapainen järjestelmä. Kaksivaiheinen käyttäjän todentaminen sisään kirjautuessa nostaisi sovelluksen turvallisuutta. Sisäänkirjautuessa palvelin lähettäisi käyttäjän sähköpostiin lyhyen koodin, jonka käyttäjä syöttää sovellukseen sisäänkirjautumisen yhteydessä. Tästä koodista ja käyttäjän julkisesta avaimesta mobiilisovelluksessa sekä palvelimella luotaisiin käyttäjälle salainen avain määritetyllä tavalla. Muita jatkokehityskohteita sisään kirjautumisen ohella ovat web-sivustoon liittyvien metodien kehittäminen ja tietokannan laajentaminen sisältämään myös web-sivuston käyttöön tarkoitettuja tauluja.

LÄHTEET

1. Rodriguez, A. 2015. RESTful Web services: The basics. IBM developerWorks. Saatavissa: http://www.ibm.com/developerworks/webservices/library/ws-restful/ws-restful-pdf.pdf. Hakupäivä 30.1.2016.

2. Elkstein, M. 2008. Learn REST: A Tutorial. Saatavissa: http://rest.elkstein.org/. Hakupäivä 5.2.2016.

3. Fielding, R.T. 2000. Architectural Styles and the Design of Network-based Software. Väitöskirja. University of California. Saatavissa: http://jpkc.fudan.edu.cn/picture/article/216/35/4b/22598d594e3d93239700ce79bce1/7ed3ec2a-03c2-49cb-8bf8-5a90ea42f523.pdf. Hakupäivä: 5.2.2016.

4. Richardson, L. – Ruby, S. 2007. RESTful Web Services. O'Reilly. Saatavissa: http://www.crummy.com/writing/RESTful-Web-Services/RESTful_Web_Services.pdf. Hakupäivä: 5.2.2016.

5. Fielding, R.T. – Taylor, R.N. 2000. Principled design of the modern Web architecture. Saatavissa: https://www.ics.uci.edu/~taylor/documents/2002-REST-TOIT.pdf. Hakupäivä 17.2.2016.

6. W3C Help and FAQ 2009. What is the difference between the Web and the Internet? W3C. Saatavissa: http://www.w3.org/Help/#webinternet. Hakupäivä: 17.2.2016.

7. Kumar Sekar, A. 2012. Create RESTful Services API in PHP. 9Lessons. Saatavissa: http://www.9lessons.info/2012/05/create-restful-services-api-in-php.html. Hakupäivä 17.2.2016.

8. Fielding, R. – Gettys, J. – Mogul, J.C. – Nielsen, H.F. – Masinter, L. – Leach P.J. – Berners-Lee, T. 1999. Hypertext Transfer Protocol – HTTP/1.1. IETF. RFC 2616. Saatavissa: https://tools.ietf.org/html/rfc2616. Hakupäivä 16.3.2016.

9. Martin, J. 1983. Managing the Data Base Environment. Pearson Education.

10. Guillermo, R. 2012. Smashing Node.js JavaScript Everywhere. Saatavissa: http://aod.zyklon-b.tk/library/web-development/%7BEN%7D_%5BRauch%5D_Smashing_Node.js.pdf. Hakupäivä 1.1.2016.

11. Orsini, L. 2013. What you need to know about Node.js. Readwrite. Saatavissa: http://readwrite.com/2013/11/07/what-you-need-to-know-about-nodejs/. Hakupäivä 22.3.2016.

12. Tomislav, C. Why The Hell Would I Use Node.js? A Case-by-Case Tutorial. Toptal. Saatavissa: https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js. Hakupäivä 22.3.2016.

13. Reed, N. 2011. What is npm? Nodejitsu. Saatavissa: https://docs.nodejitsu.com/articles/getting-started/npm/what-is-npm. Hakupäivä 22.3.2016.

14. Restify Documentation. Restify. Saatavissa: http://restify.com/. Hakupäivä 22.3.2016.

15. Hovi, A. – Huotari, J. 2005. Tietokantojen suunnittelu ja indeksointi. Docendo Finland Oy.

16. MySQL 2012. Wikipedia. Saatavissa: https://en.wikipedia.org/wiki/MySQL. Hakupäivä 18.3.2016.

17. Kelly, S. – Frankel, S. 2007. RFC4868. Aruba Networks. Saatavissa: https://www.ietf.org/rfc/rfc4868.txt. Hakupäivä 31.3.2016.

18. Crocker, D. 1982. RFC 0822. University of Delaware. Saatavissa: https://www.ietf.org/rfc/rfc0822.txt. Hakupäivä 31.3.2016.

19. Origin Of The Abbreviation I18n. Tex Texin. Saatavissa: http://www.i18nguy.com/origini18n.html. Hakupäivä 10.4.2016.