Älykkyyden lisääminen kieltenopiskelupalveluun

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

HAMK Riihimäki, tieto- ja viestintätekniikka

Kevät, 2019

Niklas Nikanti

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikka
Riihimäki

Tekijä Niklas Nikanti **Vuosi** 2017

Työn nimi Älykkyyden lisääminen kieltenopiskelupalveluun

Työn ohjaaja/t Petri Kuittinen, Niina Valtaranta

TIIVISTELMÄ

Tämän opinnäytetyön aiheena oli lisätä älykkyyttä kieltenoppimispalveluun nimeltä Reactored. Se on hämeenlinnalaisen startup-yrityksen, Rouhia Oy:n kehittämä sovellus, jossa käyttäjät voivat opiskella uusia kieliä. Reactoredissa käyttäjät voivat opiskella kieliä haluamallaan tavalla. Esimerkiksi pelillisellä tyylillä, jossa käyttäjän ei välttämättä tarvitse osata kirjoittaa mitään, vaan hän voi esimerkiksi valita kuvista. Taitojen karttuessa hän voi edetä vaativimpiin tehtävämuotoihin. Käyttäjät voivat myös itse luoda oppimateriaalia ja jakaa sitä muiden käyttöön. Tämä tekee siitä oivallisen työkalun opettajien käyttöön kielten opetuksessa. Reactored on suunnattu pääasiallisesti koulujen ja muiden opetusorganisaatioiden käyttöön, mutta myös yksityiskäyttäjät voivat käyttää sitä.

Varsinainen opinnäytetyön toimeksianto oli luoda Reactorediin järjestelmä, joka mahdollistaa tehtävien helpomman luomisen. Järjestelmän tavoitteena on automatisoida tehtävän vastausten kääntäminen, jolloin oppimateriaalin tekijän ei välttämättä tarvitse osata kieltä, jolle hän tekee tehtäviä. Tällöin käyttäjät voivat esimerkiksi tehdä itselleen opiskelumateriaalia heitä itseään kiinnostavista aiheista. Tämä myös helpottaa tehtäviä tekevien opettajien taakkaa, kun heidän ei tarvitse välttämättä käsin kirjoittaa vastausta jokaiseen tehtyyn tehtävään, vaan järjestelmä hoitaa sen automaattisesti.

Työn aikana tuli vastaan muutamia ongelmia ja haasteita, jotka hidastivat projektin valmistumista, mutta lopputuloksena saatiin luotua järjestelmä, joka mahdollistaa tehtävien vastausten automaattiset käännökset. Vaikka se poikkeaakin hieman alkuperäisestä suunnitelmasta, oli opinnäytetyön toimeksiannon antaja tyytyväinen lopputulokseen. Järjestelmässä on kuitenkin vielä parannettavaa ja ideoita jatkokehitystä varten.

Avainsanat AngularJS, JavaScript, PostgreSQL, Sails.js, web-sovelluskehitys

Sivut 25 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technology
Riihimäki

Author Niklas Nikanti **Year** 2017

Subject Adding intelligence to a language learning service

Supervisors Petri Kuittinen, Niina Valtaranta

ABSTRACT

The goal of this thesis project was to add intelligence to a language learning service called Reactored. It is a web app, where the users can learn new languages in their preferred style. It was developed by a startup company from Hämeenlinna called Rouhia Oy. Learners can for example study in a gamified style so they do not have to write anything but can instead choose from images given to them. When they become more skilled, they can advance to more challenging studying modes. The users can also create study material by themselves and share it for others to use. This makes the software an excellent tool for teachers to use in language teaching. Reactored is primarily aimed at schools and other teaching organizations but private users can also get access to it.

The actual assignment in the thesis project was to create a system that makes it easier to create new tasks for a lesson. The goal of the system is to automate translating the answers so that the creator does not necessarily have to know the language in order to create new tasks for it. This makes it possible for example for users to create study material for themselves in topics that interest them. This also eases the burden of the teachers that create new material because they do not necessarily have to write out every single translation by hand because the system will deal with this automatically.

I encountered some problems that slowed down the progress along the way of making the system. But in the end, I managed to overcome them and create a system that enables automatic translations for creating lesson tasks. Even though it differs a bit from the original plan, the client was quite happy with the end result. There are still some things in the system to improve on and new ideas for further development.

Keywords AngularJS, JavaScript, PostgreSQL, Sails.js, web development

Pages 25 pages

KÄSITTEISTÖ

JavaScript	Kevyt, dynaaminen, tulkattu, prototyyppipohjainen ohjelmointikieli. Se kehitettiin alun perin luomaan lisätoiminnallisuutta verkkosivuille käyttöliittymäpuolella, mutta ajan myötä sen ominaisuudet ovat kasvaneet. Nykyään sitä voidaan ajaa myös serveripuolella esimerkiksi Node.js alustalla.
HTML	Hypertext Markup Language. Hierarkkinen kuvauskieli, jolla merkitään verkkosivun käyttöliittymän rakennetta sekä sisältöä. Sen avulla voi esimerkiksi merkitä minkä elementin sisällä nappula sijaitsee ja mitä siinä lukee.
CSS	Cascading Style Sheets. Tyyliohjekieli, jonka avulla HTML-sivuilla voidaan muuttaa tai lisätä elementtien tyylejä, kuten nappien fonttia, muotoa ja värejä.
HTTP	Hypertext Transport Protocol. Protokolla, jonka avulla siirretään tai vaihdetaan hypertekstiä (hypertext), joka on WWW:n eli World Wide Webin perustoimintaperiaate.
CRUD	Create Read Update Delete. Pysyvän tiedontallennuksen perusoperaatiot tiedon tallentamiseen eli luo, lue, päivitä ja poista. SQL-tietokantakyselyssä nämä ovat: INSERT, SELECT, UPDATE ja DELETE. HTTP:ssa nämä ovat POST/PUT, GET, POST/PUT/PATCH ja DELETE.
MVC	Model Viewer Controller. Ohjelmistoarkkitehtuuri käyttöliittymien luontiin. Se jakaa sovelluksen kolmeen osaan: Malliin (model), näkymään (view) ja käsittelijään (controller). Malli hallitsee sovelluksen dataa ja logiikkaa, näkymä näyttää sovelluksen käyttäjälle perustuen mallin tietoihin ja käsittelijä päivittää mallin tietoja, kun sovelluksen tila muuttuu esimerkiksi käyttäjän kirjoittaessa lomaketta.
Scaffolding	Tekniikka, joka määrittää sovelluksen kehyksen sen käyttämien tekniikoiden ehdoilla. MVC-sovellusviitekehykset käyttävät tätä tekniikkaa luodakseen uuden sovelluksen kehityksen aloituksen yhteydessä perusrungon sovellukselle, jotta sen kehittämistä ei tarvitse aloittaa aina tyhjästä. Sen tarkoitus on siis helpottaa kehittäjien työtä ja säästää aikaa.

RESTful API	Representational stateless transfer application programming interface. Ohjelmointirajapinta, joka hyödyntää HTTP-protokollaa tiedonsiirtoon internetissä. Se pyrkii parantamaan suorituskykyä, skaalautuvuutta, yksikertaisuutta, muokattavuutta ja luotettavuutta. Se on tilaton eli se ei itsessään ylläpidä tietoa tiedonsiirtotapahtumista, vaan se on sitä käyttävän sovelluksen vastuulla.
ORM	Object-relational mapping. Ohjelmointitekniikka, jossa muutetaan tietoa kahden yhteensopimattoman tyyppijärjestelmän välillä käyttäen olio-ohjelmointikieliä. Yleisesti ottaen ORMia käytetään muuttamaan relaatiotietokannan tietoa oliomuotoon ja päinvastoin.
Typescript	Microsoftin kehittämä JavaScriptin ylijoukko. Se on vaihtoehtoisesti tyypitetty ja staattinen kieli, joka kompiloituu takaisin JavaScriptiksi. Sen tarkoitus on helpottaa JavaScript-kehittäjien työtä mahdollistamalla virheiden etsinnän jo koodausvaiheessa. Esimerkiksi Googlen kehittämä Angular-viitekehys käyttää tätä kehityskielenään.
ORDBMS	Object-relational database management system. Oliorelaatiotietokantahallintajärjestelmä eli tietokantajärjestelmä, jossa tieto on tallennettu relaatiomuodossa ja sitä voidaan manipuloida kyselyillä, mutta tämän lisäksi se tukee käyttäjän määrittelemiä tietotyyppejä, periytyvyyttä ja olion kaltaista käytöstä. Se käytännössä yhdistää relaatiotietokantojen ja oliotietokantojen parhaat puolet.

SISÄLLYS

JOHDANTO

Tämän opinnäytetyön aihe liittyy osittain minua kiinnostavaan aiheeseen: tekoälyyn, joka on nyt ja tulevaisuudessa nopeasti kehittyvä ala. Opinnäytetyön tavoitteena on niin sanotusti lisätä älykkyyttä kieltenopiskelupalveluun nimeltä Reactored. Sen on kehittänyt Rouhia Oy, joka on pieni hämeenlinnalainen startup-yritys, jossa siis itsekin tällä hetkellä työskentelen. Reactoredissa käyttäjät voivat luoda tehtäviä omaan ja muiden käyttöön, mutta se on pääasiallisesti suunnattu koulujen ja muiden opetusorganisaatioiden käyttöön.

Opinnäytetyön aiheena on luoda järjestelmä, joka oppii käyttäjien luomista tehtävistä täydentämään jatkossa luodut tehtävät automaattisesti. Esimerkiksi, jos 90% käyttäjistä on luonut samaan lauseeseen saman vastauksen, niin järjestelmä hyväksyy sen ja jatkossa käyttäjien luomiin tehtäviin, jossa on tämä sama lause, käännös lisätään automaattisesti vastaukseksi. Tehtäviä tekevät käyttäjät voivat tarvittaessa poistaa automaattisesti lisätyt vastaukset tehtävästä, jolloin järjestelmä muistaa sen ja tarvittaessa poistaa sen automaattisesti lisättävistä vastauksista, jos tarpeeksi monta käyttäjää on poistanut sen. Järjestelmään tulisi myös mahdollisuus raportoida mahdollisista huonoista käännöksistä, jolloin niitä ei näkyisi, jos niitä on raportoinut tarpeeksi monta käyttäjää. Käyttäjät voisivat siis luoda uusia tehtäviä, vaikka he eivät aluksi tietäisikään käännöstä kysymykselle. Tavoitteena on siis helpottaa palvelua käyttävien käyttäjien tehtävien luontia sovelluksessa.

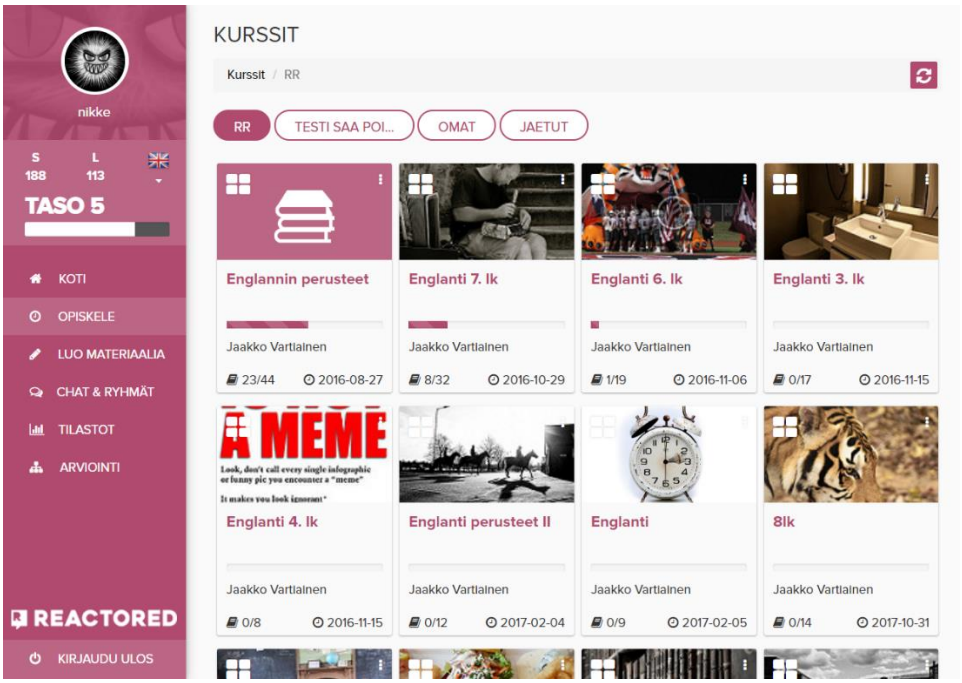
Työhön liittyy lisäksi kokeiden tehtävien tarkistuksen parantaminen. Nykyisellään kokeiden tehtävät tarkistetaan oikein tai väärin -metodilla. Tavoitteena on lisätä hienojakoisempi tarkistustapa prosentuaalisella -metodilla. Tehtävässä voi esimerkiksi olla kaikki sanat oikein, mutta sanajärjestys on väärä. Tällöin olisi väärin antaa tehtävästä nolla pistettä esimerkiksi 75% sijaan. Tehtävän palautuksen yhteydessä tehtävän tekijä saisi myös heti palautteen, joka kertoisi, mikä meni väärin ja näyttäisi myös oikean vastauksen.

PALVELUN KUVAUS JA TAUSTA

1.1 Tietoa palvelusta

Reactored on kieltenopiskelupalvelu, jossa käyttäjät voivat luoda oppimateriaalia omaan ja muiden käyttöön. Oppimateriaali on jaoteltu kursseihin, jotka käsittävät jonkin aihepiirin (Kuva 1). Kurssien sisällä on oppitunteja, joissa käsitellään jotain tiettyä aihepiiriin liittyvää asiaa. Oppitunnit sisältävät siihen liittyviä tehtäviä ja teoriaa sekä kontekstin. Esimerkiksi kotieläimet -oppitunti sisältää tehtäviä liittyen kotieläimiin. Tämä oppitunti on puolestaan kurssin sisällä, joka käsittää laajemman kokonaisuuden, johon se kuuluu. Kurssi voi olla esimerkiksi 7. luokan englanti, joka on suunnattu peruskoulun 7. luokan oppilaiden käyttöön. Opetusmateriaalin voi rajoittaa vain organisaation käyttöön, jolloin organisaation ulkopuoliset eivät näe materiaalia. Esimerkiksi opettaja voi luoda englannin oppitunnin vain oman koulunsa tai tietyn ryhmän käyttöön tai jakaa sen vain tietylle oppilaalle, joka tarvitsee erityisopetusta tietyssä asiassa.

Palvelussa on myös mahdollista luoda eri tyyppisiä tehtäviä. Näitä ovat muun muassa käännöstehtävä, jossa annetaan sana tai lause käännettäväksi puhumalla tai kirjoittamalla se vastauskenttään ja kuvatehtävä, jossa valitaan annettu sana tehtävässä näkyvistä kuvista. Lisäksi käyttäjä voi oppituntia aloittaessaan valita haluaako hän tehdä tehtäviä pelillisellä tyylillä vai enemmän kirjoittamalla, jolloin oppitunnin tehtävät annetaan käyttäjälle hänen haluamassa muodossa.



Kuva 1. Reactoredin opiskele näkymässä näkyvät kurssit, joita käyttäjä voi opiskella.

Palvelun opetusmetodi antaa oppijalle välitöntä palautetta visuaalisin ja äänen keinoin. Esimerkiksi, kun tekstikenttään kirjoitettu sana ei enää vastaa vastausta, niin se muuttuu punaiseksi ja kuuluu äänimerkki, joka huomauttaa käyttäjää väärin menevästä vastauksesta. Palvelu seuraa ja tallentaa käyttäjän oppimia sanoja sekä lauseita ja niistä kertyviä tasopisteitä. Ne näkyvät käyttäjän henkilökohtaisissa tilastoissa, jossa niitä voi kuunnella ja kerrata, ja niitä voi hyödyntää palvelun keskustelualueella, jossa niitä ehdotetaan ennakoivasti, jos käyttäjän kirjoittama sana tai lause vastaa niitä.

1.2 **Palvelun tekniikat**

Reactored -kieltenopiskelupalvelu on web-pohjainen sovellus, joka toimii verkkoselaimessa. Täten siis palvelua voi käyttää kaikissa laitteissa, joissa on moderni, HTML5:sta tukeva verkkoselain, kuten puhelimissa, tableteissa, pöytätietokoneissa, kannettavissa tietokoneissa ja vaikkapa älytelevisioissa. Palvelun taustapääte (back end) on toteutettu käyttäen Node.js-ajoympäristöä (runtime environment), Sails.js-viitekehystä (framework) sekä PostgreSQL-tietokantajärjestelmää. Palvelun etupääte (front end) on toteutettu AngularJS-viitekehystä käyttäen sekä Bootstrap-kirjastoa hyödyntäen. Koska sovelluksen molemmissa päissä käytetään JavaScriptiä, voidaan siinä täysivaltaisesti hyödyntää JavaScriptin uusimpia ominaisuuksia.

Node.js valittiin palvelun serveripuolen alustaksi, koska se on nopea, suosittu ja se skaalautuu hyvin käyttäjämäärän kasvulle. Tämän lisäksi se käyttää JavaScriptiä ohjelmointikielenään, mikä mahdollistaa helpomman full-stack-kehityksen, koska samaa kieltä voidaan käyttää sovelluksen taustapuolella sekä käyttöliittymässä, joten sama kehittäjä voi toteuttaa helpommin sovelluksen molemmat puolet. (Texeira, 2012, 3.)

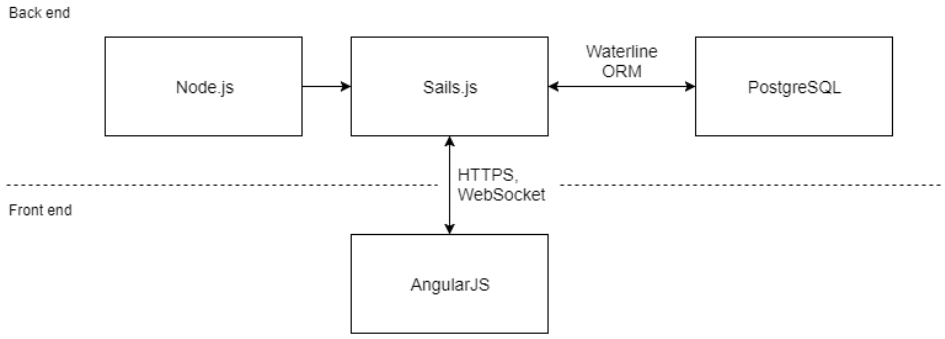
Sails.js puolestaan lisää Node.js:n toiminnallisuutta reaaliaikaisten sovellusten luonnissa, koska se sisältää Socket.IO-verkkopistoke (web socket) -kirjaston päälle rakennetun Sails.IO-kirjaston, jolla se toteuttaa selaimen ja serverin välisen yhteyden. Se siis mahdollistaa reaaliaikaisen chatin sekä pelillisten tehtävien luonnin palveluun. Toinen Sails.js:n tai minkä tahansa sovellusviitekehyksen käytön etu on koodauksen abstrahointi viitekehyksen periaatteiden mukaisesti ja turhan, niin sanotun vakiokoodin (boiler plate) määrän pienentäminen. Tällöin sovellusta ei tarvitse kirjoittaa alusta asti, vaan voidaan käyttää enemmän aikaa varsinaisten ominaisuuksien kehittämiseen.

Sovelluksen tietokannaksi valittiin PostgreSQL, koska se on todella laaja tietokantajärjestelmä ja siinä on enemmän toiminnallisuutta esimerkiksi verrattuna MySQL:än ja se noudattaa SQL-standardeja paremmin kuin edellä mainittu. Lisäksi sitä kehitetään jatkuvasti, se on hyvin dokumentoitu ja suosittu, joten lähes jokaiseen sen kanssa vastaantulevaan ongelmaan on olemassa valmiiksi dokumentoitu ratkaisu, jota voi soveltaa.

Kehityksessä pyritään myös hyödyntämään JavaScriptin uusimpia kehitystä helpottavia ominaisuuksia. Näitä ovat esimerkiksi nuolifunktiot (arrow function), asynkroniset funktiot (asynchronous function), await-avainsana (keyword), konstantit (constant) ja skoopatut muuttujat (scoped variable). Nuolifunktiot sisältävät lyhyemmän syntaksin ja funktion this-avainsana ei viittaa itse funktion skooppiin, vaan sen ulkopuolella olevaan isäntäskooppiin (parent scope). Asynkroniset funktiot ja await-avainsana puolestaan helpottavat JavaScriptin asynkronisesta luonteesta johtuvaa yleistä ongelmaa, jossa asynkronisuuden saavuttamiseksi tarvitaan merkittävä määrä takaisinkutsufunktioita. Sitä kutsutaan yleisesti nimellä takaisinkutsuhelvetti (callback hell). Näiden avulla asynkronisesta koodista voidaan tehdä synkronisen näköistä, jolloin koodin rakenne yksinkertaistuu ja se on helpommin ymmärrettävissä. Konstantit ja skoopatut muuttujat puolestaan selkeyttävät JavaScriptissä ollutta muuttujien skooppausongelmaa, jossa skooppien sisällä olevat var-avainsanalla merkityt muuttujat vuosivat niiden ulkopuolella oleviin skooppeihin. Edellä mainitut ominaisuudet tulivat ECMAScriptin versioiden ES6 ja ES2017 mukana. (ECMA-international, 2015)

Sovelluksen etupäätepuolella puolestaan käytetään Googlen kehittämää AngularJS-viitekehystä, joka mahdollistaa käyttöliittymien nopean ja kivuttoman kehityksen. AngularJS valittiin käyttöön sovelluksen aiempien kehittäjien toimesta sen tuttavallisuuden, vakauden, kypsyyden, suosion ja laajan dokumentaation takia. Sen kehitys oli siis Reactored-sovelluksen kehitystä aloittaessa pidemmällä kehityksessä kuin sen muut pääasialliset kilpailijat: React ja Vue.js. Tämä tarkoitti sitä, että siihen ei todennäköisesti sovelluksen kehityksen edetessä tullut merkittäviä päivityksiä, jotka rikkoisivat taaksepäin yhteensopivuuden aiempien versioiden kanssa. AngularJS:n lisäksi sovelluksen asiakaspuolella toimii joukko muita pienempiä avustavia kirjastoja kuten aiemmin mainittu Bootstrap. Myöskin ollessaan ison tunnetun yhtiön kehittämä viitekehys, sen tuki olisi taattu pitkälle tulevaisuuteen, ilman pelkoa siitä, että sen kehittäjä hylkäisi sen yhtäkkiä.

AngularJS ottaa yhteyden palvelimelle verkkopistokkeiden avulla käyttäen Angular-Sails -kirjastoa, joka keskustelee Sails.IO-kirjaston kanssa. Sails.IO-kirjastossa on kaksi osaa: asiakkaan (client) kirjasto ja palvelinpuolen (server) kirjasto. Serveripuolen Sails.js-viitekehystä puolestaan ajetaan Node.js-ajoympäristön päällä. Sails.js keskustelee PostgreSQL-tietokannan kanssa sen sisäänrakennetun Waterline ORM:in avulla. (Kuva 2)



Kuva 2. Reactored on rakennettu hyödyntäen Sails.js:ä, AngularJS:ä ja PostgreSQL:ä.

1.3 **Node.js**

Node.js on JavaScript-ajoympäristö, joka toimii serveripuolella. Sillä voi siis ajaa JavaScriptiä, joka on historiallisesti ollut selainpuolen skriptauskieli, serverillä. Se on rakennettu Googlen Chrome-selaimen V8-JavaScript-moottorin päälle, mikä tekee siitä erittäin suorituskykyisen. Node.js on asynkroninen ajoympäristö, joka käyttää yhtä säiettä (thread) toisin kuin nykyään yleinen malli, jossa käytetään käyttöjärjestelmän säikeitä, jotta voidaan saavuttaa monien yhteyksien yhtäaikaisuus (concurrency). Tämän ansiosta Node.js välttää järjestelmän prosessien lukittautumisen (deadlock) ja mahdollistaa skaalautuvien järjestelmien helpomman kehittämisen. Vaikka se onkin suunniteltu ilman säikeistystä (threading), se ei tarkoita, etteikö se voi hyödyntää prosessorin ylimääräisiä säikeitä, sillä ylimääräisiä lapsiprosesseja (child process) voidaan luoda käyttäen child_process.fork() API:a. Kommunikointi näiden prosessien välillä on tehty helpoksi ja lisäksi cluster-moduulilla voidaan jakaa verkkopistokkeita prosessien välillä ja mahdollistaa kuormantasaus (load balancing) säikeiden välillä. (Node.js, n.d-a) (Node.js, n.d-b)

Node.js hyödyntää niin sanottua tapahtumasilmukkaa (event loop) kirjaston (library) sijasta ajon aikaisten (runtime) operaatioiden suorittamiseen. Node.js:ssa HTTP on ensimmäisen luokan kansalainen (first class citizen), tehden siitä hyvin soveltuvan verkkokirjaston (web library) tai viitekehysten (framework) pohjaksi. (Node.js, n.d-a) (Node.js, n.d-b)

1.4 **Sails.js**

Sails.js on Node.js-ympäristön kanssa toimiva ja sen pohjalle luotu MVC-viitekehys. Sillä voidaan luoda valmiita yritystason Node.js-sovelluksia. Sails.js:n kanssa voi käyttää lähes mitä tahansa tietokantaa, koska se käyttää mukana tulevaa Waterline ORM:ia. Siihen on saatavilla valmiita sovittimia (adapter) kaikkiin yleisimpiin tietokantajärjestelmiin kuten MySQL, PostgreSQL ja MongoDB. Lisäksi se yhdenmukaistaa kaikkien sovelluksessa käytettyjen tietokantojen käsittelyn antaen mahdollisuuden käyttää sa-

moja CRUD-metodeja mille tahansa sen tukemalle tietokantajärjestelmälle. Waterline sisältää myös tietokannan taulujen mallien (model) validoinnin, jota käytetään oletuksena tietokannan CRUD-operaatioissa. Sails.js:n mukana tulee myös kehittämistä helpottavia scaffolding-työkaluja, kuten automaattinen RESTful API:n luonti niin sanottujen pohjapiirrosreittien (blueprint route) avulla ja Socket.IO-viitekehys, joka mahdollistaa reaaliaikaisten sovellusten luonnin verkkopistokkeita hyödyntämällä. Se sisältää myös monia tuotantotason automaatiotyökaluja kuten CSS- ja JavaScript-minimoinnin (minification). Lisäksi se sisältää myös perustason turvallisuuden ja tunnistautumisen, jos sovellus vaatii sitä ja käyttäjä ei tarvitse järeämpää turvallisuutta sovellukseensa. (Sails.js, n.d) (Krol 2014, 455.)

1.5 **PostgreSQL**

PostgreSQL on oliorelaatiotietokantahallintajärjestelmä (ORDBMS), joka korostaa laajennettavuutta ja standardien, kuten ANSI-SQL:2008, noudattamista. Se toimii kaikilla yleisimmillä käyttöjärjestelmillä kuten Linux, UNIX ja Windows. Se tukee yleisimpiä tietotyyppejä, kuten kokonaisluku (integer), totuusarvo (boolean), merkkijono (varchar), päivämäärä (date) ja aikaleima (timestamp). Se tukee myös suurien binääriobjektien, kuten kuvien, äänen tai videon tallentamista. Siinä on natiivi ohjelmointirajapinta monelle kielelle, mukaan lukien C/C++, Java, .Net, Perl, Python ja Ruby.

Koska PostgreSQL on yritystason tietokantajärjestelmä, se sisältää monia hienostuneita ominaisuuksia, kuten Multi-Version Concurrency Control (MVCC) -toiminto, joka sallii yhtäaikaiset operaatiot tietokannassa, tietokannan palautus tiettyyn ajanpisteeseen, tauluavaruudet (tablespaces) sekä edellä kirjaamisen (write ahead logging) vikasietoisuuden parantamiseksi. Tämän lisäksi se skaalautuu hyvin suurelle tietomäärälle ja yhtäaikaisten käyttäjien määrälle. Se tukee myös kansanvälisiä merkistöjä, Unicodea, eri lokaalien lajittelua, merkkikokoriippuvaisuutta ja formatointia (PostgreSQL, n.d)

1.6 **AngularJS**

AngularJS on Googlen kehittämä MVC-sovelluskehitysviitekehys käyttöliittymäpuolelle websovellusten luontiin. Sen tarkoitus on nopeuttaa ja helpottaa dynaamisten websovellusten luomista. AngularJS:lla websovellusta kehitettäessä HTML-koodin sekaan voidaan lisätä koodia, jolla sidotaan (data binding) elementtien sisältö koodiin. Kontrollerit (controller) ovat vastuussa HTML-sisällön päivittämisestä dynaamisesti. Ne voivat olla määritetty tiettyyn reittiin (route), joka on sidottu tiettyyn URL-osoitteeseen sovelluksen sisällä (Kuva 3) tai direktiivin (directive), joka on toisen reitin HTML-mallin (template) sisällä. Direktiivit sidotaan JavaScript koodissa määriteltyyn kontrolleriin ja merkitään mihin elementtiin HTML-mallissa

ne halutaan sitoa. Tämä voi tapahtua käyttäjän määrittämänä joko elementin nimen, attribuutin, luokan nimen tai kommentin perusteella. Tämän lisäksi on olemassa myös tehtaita (factory) ja palveluita (service). Palveluita voidaan käyttää tiedon hakemiseen tietokannasta, ja tehtaat yleensä varastoivat tätä tietoa, joten sitä ei tarvitse hakea erikseen joka kerta eri kontrollereiden välillä, mikäli samaa tehdasta käytetään useassa kontrollerissa. Ne ovat kerran ladattavia funktioita (singleton) eli ne ladataan ainoastaan silloin, kun niitä käytetään ensimmäisen kerran ja sen jälkeen sama instanssi siitä on käytössä kaikkialla sovelluksessa. Tehtaiden ja palveluiden avulla voidaan hallita jotain tiettyä osaa kontrollerissa, kuten esimerkiksi sovelluksen käyttäjän kielivalinnan ja käyttäjäkuvan lataamista, muuttamista ja tallentamista. Ne eivät ole suoraan tekemisissä sovelluksen HTML-puolen kanssa, vaan ne ovat määritelty kontrollerin sisällä JavaScript-koodissa ja niitä voi olla useita samanaikaisesti. HTML-puolella kontrollerin sisällä olevat elementit ovat yleensä kaikki samassa skoopissa (scope). Kontrollereiden sisällä voi olla myös muita kontrollereita tai direktiivejä, joilla on eristetty skooppi (isolated scope), jolloin sen sisällä tapahtuvat muutokset eivät suoraan vaikuta ylempänä oleviin skooppeihin, ellei käyttäjä erikseen määrittele näin. (AngularJS, n.d)

```
angular
    .module("reactored")
    .config([
        "$routeProvider",
        "$locationProvider",
        function($routeProvider, $locationProvider) {
            $routeProvider
                .when("/", {
                    controller: "mainController",
                    controllerAs: "ctrl",
                    templateUrl: "/templates/mainLanding.html"
                })
                .when("/settings", {
                    controller: "mainSettingsController",
                    templateUrl: "/templates/mainSettings.html"
                })
                .when("/study/:tab?/:orgId?", {
                    controller: "mainStudyController",
                    controllerAs: "ctrl",
                    templateUrl: "/templates/mainStudy.html"
                })
```

Kuva 3. AngularJS:n reittien määrittely tapahtuu moduulin config-metodin avulla.

AngularJS eli Angularin versio 1 poikkeaa huomattavasti sen versiosta 2 ja sitä myöhemmistä versioista, jotka kantavat nimeä Angular. Tämän takia sovelluksen siirtäminen AngularJS:sta Angulariin vaatii lähes kaikissa tapauksissa koodin uudelleenkirjoittamista. Angular hylkäsi kontrollerit sekä direktiivit ja korvasi ne komponenteilla (component), joissa on huomattavasti yksinkertaisempi rakenne ja vähemmän koodia, mikä nopeutti sovellusten kehitystä. Lisäksi kehityskieli vaihtui JavaScriptistä Microsoftin kehittämään TypeScriptiin, joka on JavaScriptin ylijoukko (superset) eli se kompiloituu takaisin JavaScriptiksi. (Devblast, 3.10.2016)

PALVELUN KEHITYS

1.7 **Kehitysympäristö**

Koska kyseessä on jo suhteellisen pitkään kehityksessä ollut sovellus, käytetään sen kehittämisessä käytettyä kehitysympäristöä ja siinä mukana olevia kehitystyökaluja, ellei kehityksen aikana ilmene tarvetta muille työkaluille.

Kyseessä on webselaimella toimiva websovellus, joten käyttöliittymäpuolella käytössä ovat varmasti HTML5-tekniikat. Näihin lukeutuvat HTML, CSS ja JavaScript. Myös näitä tekniikoita hyödyntävät kirjastot, kuten JQuery ja Bootstrap ovat hyödyllisiä sovelluksen käyttöliittymän kehittämisen kannalta. Itse asiassa AngularJS:ssä on sisäänrakennettu ja kevennetty versio JQuerystä nimeltä JQLite, jota se hyödyntää toiminnoissaan (AngularJS, n.d). JQueryn hyöty on siinä, että se normalisoi DOMin eli dokumenttiolio-mallin manipuloinnin lähes millä tahansa verkkoselaimella hyödyntäen JavaScriptiä (JQuery, n.d). Bootstrap on Twitterin luoma kirjasto, jota hyödynnetään responsiivisuuden saavuttamiseksi, jotta sovellus toimii hyvin jokaisella näyttökoolla ja erityisesti mobiililaitteilla (Bootstrap, n.d). Serveripuolella puolestaan käytetään ajoympäristönä Sails.js:ia, joka on siis Node.js:n päälle rakennettu viitekehys web-sovellusten rakentamista varten. Sails.js:sta on käytössä versio 0.12. Node.js:sta on puolestaan käytössä pitkän tuen versio (LTS) 8.9.1. Tietokantajärjestelmänä sovelluksessa toimii PostgreSQL, josta on paikallisesti asennettu tällä hetkellä tuorein versio eli versio 10.

Tietokannan tarkasteluun, hallintaan, testaamiseen ja muokkaamiseen käytetään HeidiSQL-tietokantasovellusta, josta löytyy PostgreSQL-tuki. HeidiSQL:n tuki PostgreSQL:lle on kuitenkin vielä hieman aukkonainen, joten joissain tapauksissa joudutaan turvautumaan selainpohjaiseen pgAdmin-sovellukseen, jonka käyttöliittymä ja käyttäjäkokemus ei ole yhtä intuitiivinen kuin HeidiSQL:n, mutta sen tuki PostgreSQL:n ominaisuuksille on monipuolisempi.

Sovelluksen versionhallintaan puolestaan käytetään suomalaisen Linuxin kehittäjän Linus Torvaldsin kehittämää Git-versionhallintajärjestelmää ja sillä on Githubissa oma yksityinen repositorio, johon pääsee käsiksi vain kyseisen organisaation jäsenet. Tätä opinnäytetyö-projektia varten luotiin Gitin uusi haara (branch), jossa sitä voidaan kehittää rauhassa sekoittamatta sitä sovelluksen pääkehityshaaraan (master) ennen kuin se on testattu toimivaksi ja täysin valmiiksi yleiseen käyttöön. (Chacon 2014, 1.2.)

Itse koodin kirjoittamiseen käytetään Sublime Text-tekstieditoria. Tarkemmin ottaen Sublimesta on käytössä kirjoittamishetkellä uusin versio eli 3. Siinä on täysi syntaksituki JavaScriptille. Lisäksi se on nopea ja kevyt järjestelmäresursseille eli se ei vaadi tehokasta tietokonetta sen käyttöä varten.

Siitä löytyy myös monipuolisia projektinhallinnan kannalta olennaisia ominaisuuksia kuten työskentelytilan tallentaminen omaan projektiin, joka säilyttää kaikki avatut tiedostot ja kansiot muistissa, mikä tekee projektin jatkamisesta nopeampaa ja helpompaa. Koska Node.js-serveri käynnistetään komentoriviltä, tarvitsee käytössä olla jokin komentoriviterminaali, josta se käynnistetään. Tässä tapauksessa käytetään ConEmu-nimistä komentorivikonsoleiden hallintaohjelmaa, joka toimii Windowsilla. Sen avulla voi käyttää mitä tahansa käyttöjärjestelmään asennettua komentorivikonsolia. Tässä tapauksessa ConEmun kanssa käytetään Git Bash-komentorivikehoitetta, joka tulee Git-asennuksen mukana ja toimii sen kanssa erittäin hyvin.

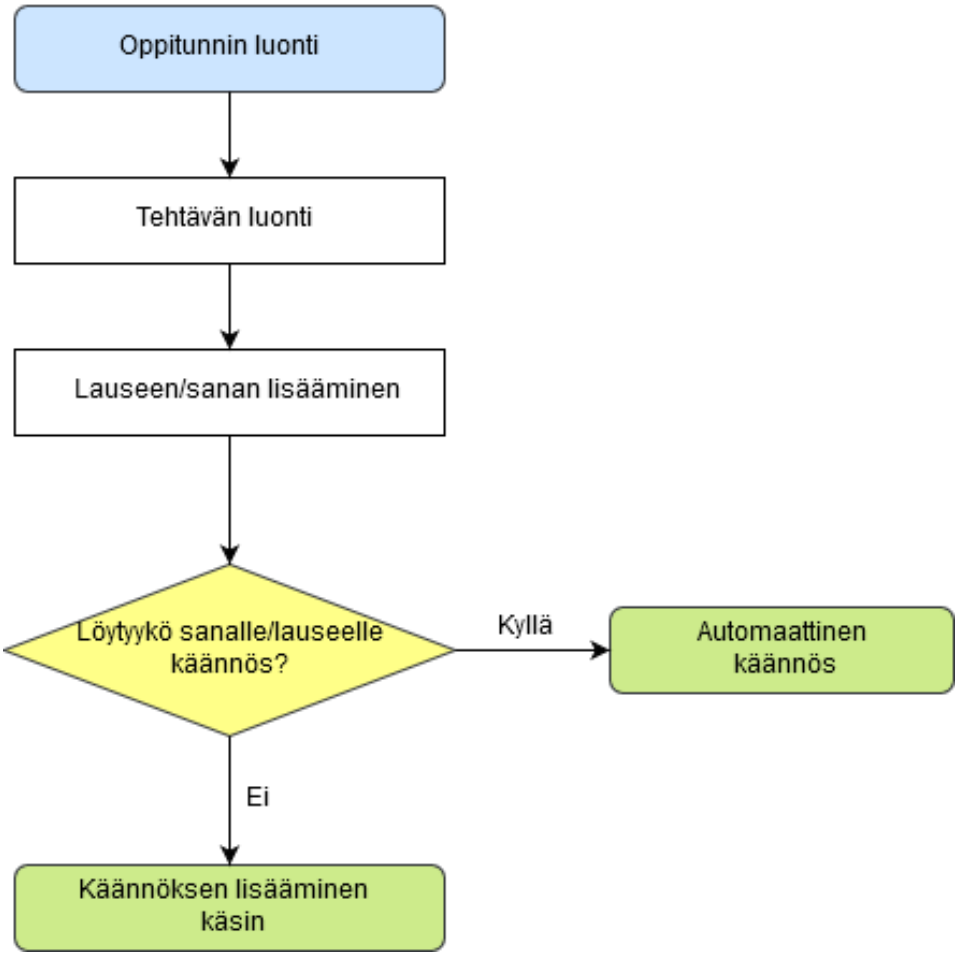
1.8 Palvelimen käynnistäminen

Sails.js:n taustalla oleva Node.js-serveri käynnistetään ajamalla ”npm run start” -komento, joka suorittaa Node.js:n avulla projektin juurikansiossa olevan app.js-tiedoston, joka puolestaan ajaa komennon Sails.js-serverin käynnistämiseen: ”sails.lift()”. Tämä komento on määritelty Node.js:n paketinhallintajärjestelmän, npm:n kautta. Tarkemmin ottaen tämä komento on npm:n luomassa package.json-tiedostossa, joka on projektikansion juuressa. Sails.js ilmoittaa komentorivillä, kun se on käynnistynyt (Kuva 4). Kun serveri on käynnistynyt, mennään selaimella kehitysympäristöön määriteltyyn osoitteeseen ja porttiin, jotka ovat tässä tapauksessa: ‘http://localhost:1337/login’, jolloin pitäisi näkyä palvelun kirjautumissivu.

Kuva 4. Sails.js-palvelin on käynnistynyt ja valmis käytettäväksi.

1.9 Toimintalogiikka

Sanojen kääntämiseen voidaan hyödyntää olemassa olevaa sanakirjaa. Lauseiden kääntäminen nojautuu tietokannassa olemassa olevien käännösten hyödyntämiseen käännösehdotuksissa. Käyttäjän luodessa oppitunnille uuteen tehtävään kysymyksen, hän voi etsiä siihen käännöksen painamalla hakunappia vastauksen vieressä. Tämä käynnistää sovelluksen tietokannasta haun, joka etsii sopivia käännöksiä kysymykselle. Tämän jälkeen avautuu ponnahdusikkuna, johon tulee näkyviin käännöksiä kyseiselle sanalle tai lauseelle, mikäli kyseiselle kysymys-vastaus-parille löytyy käännöksiä valitulla opiskelukieli-opiskeltava kieli -parilla. Jos kysymykselle löytyi käännöksiä, näytetään ne listana ponnahdusikkunassa. Käännöstä painamalla ikkuna sulkeutuu ja käännös lisätään kyseisen vastauksen kohdalle. Tämän jälkeen vastaus tallennetaan tietokantaan kyseisen tehtävän vastauksiin ja tulevaisuudessa sitä ehdotetaan vastauksena tällä kieliparilla vastaaville kysymyksille. (Kuva 5)



Kuva 5. Jos järjestelmä löytää käännöksen kysymykselle, se näytetään käännösjärjestelmän automaattisissa käännöksissä.

1.10 **Käännösten hakeminen**

Käännökset haetaan palvelimelta Sails.js:n GET-pyynnön avulla. AngularJS:ssä on käytössä Angular-Sails niminen kirjasto, jonka avulla pyyntöjen lähettäminen Sails.js-palvelimelle on helppoa, sillä siinä on valmiina pyyntömetodit kaikille Sails.js:n tukemille pyyntötyypeille. Sails.js kommunikoi käyttäjän selaimen kanssa verkkopistokkeiden avulla eli se ei käytä perinteisiä HTTP-metodeja, mutta sen vastaavat pyynnöt toimivat loogisesti varsin samalla tavalla. Angular-Sails keskustelee Sails.IO-nimisen kirjaston kanssa, joka on rakennettu Socket.IO-nimisen kirjaston päälle, jolla se avaa verkkopistokeyhteyden palvelimelle. Palvelimen vastaanotettua pyyntö käännösten hakua varten määriteltyyn päätepisteeseen (end point), se noutaa pyynnön hakuparametrien mukaan tietokannasta tietystä taulusta määritellyt tiedot ja lähettää ne takaisin vastauksena alkuperäiseen pyyntöön.

Hakufunktion kuuntelijalle palvelimella on asetettava lupa, jotta se toimii. Sails.js:ssä tämä sijaitsee policies.js-nimisessä tiedostossa (Kuva 6). Se toimii valkolista-periaatteella (white list) eli oletuksena kaikki saapuvat pyynnöt kontrollereihin ja näiden funktioihin, jotka eivät ole määritelty siihen, estetään. Kontrollerit ovat siis vastuussa sovelluksen päätepisteistä ja oletuksena ne ovat nimetty päätepisteen ja Controller-päätteen mukaisesti. Esimerkiksi chatController-niminen kontrolleri on vastuussa kaikista ”/chat”-päätepisteeseen tulevista pyynnöistä ja tämän kontrollerin metodit toimivat päätepisteen hakuparametreina. Policies.js-tiedosto noudattaa Node.js:n JavaScript-objekti-muotoisuutta (Sails, n.d). Kun halutaan sallia jokin pyyntö serverille, siihen asetetaan ensin kontrollerin nimi objektin ominaisuudeksi (property) ja tämän objektin sisälle lisätään niiden kontrollereiden metodien nimet, joiden pyynnöt halutaan sallia. Näiden arvot ovat taulukoita (array), joiden sisällä ovat säännöt, joiden mukaan pyynnöt sallitaan. Nämä säännöt ovat Node.js-muotoisia niin sanottuja middleware-funktioita, jotka ajetaan joka kerta, kun kyseiseen päätepisteeseen lähetetään pyyntö. Yksinkertaisin sääntö on vain sallia pyyntö ilman mitään lisäehtoa, niin sanottu whitelist-sääntö. Yleisin sääntö on kuitenkin tarkistaa, että käyttäjä on autentikoitu sovellukseen eli onko hän kirjautunut sisään oikealla tai olemassa olevalla käyttäjätunnuksella. Sails.js käyttää tähän tarkistetukseen sisäisesti HTTP-evästettä (cookie), jolla se varmentaa käyttäjän session aitouden. Policies.js-tiedostoon muutoksia tehdessä ne eivät astu heti voimaan, vaan vasta serverin seuraavalla käynnistyksellä.

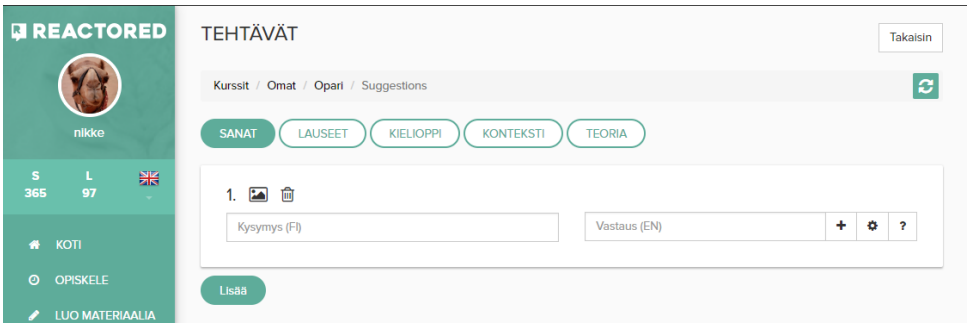
```
331     },
332     Study_itemController: {
333         destroy: ["passport", "sessionAuth", "hasLicense"],
334         find: ["passport", "sessionAuth"],
335         findOne: ["passport", "sessionAuth"],
336         on: ["passport", "sessionAuth", "hasLicense"],
337         create: ["passport", "sessionAuth", "hasLicense"],
338         update: ["passport", "sessionAuth", "hasLicense"],
339         createOrUpdateStudyItem: ["passport", "sessionAuth", "hasLicense"], // Todo check
340         deleteStudyItem: ["passport", "sessionAuth", "hasLicense"],
341         getStudySetItemsWithGrammars: ["passport", "sessionAuth", "hasLicense"], // TODO
342         searchTranslation: ["passport", "sessionAuth", "hasLicense"]
343     },
```

Kuva 6. Policies.js-tiedostossa määritetään sovelluksen sallitut päätepisteet sekä niiden säännöt ja ehdot.

Tässä tapauksessa haku parametreina Sails.js:n GET-pyyntöön lähetetään kyseessä oleva oppitunti sekä tehtävä. Tämän jälkeen etsitään tietokannasta tehtäviä, joilla on vastaava kysymys kuin lähetetyllä tehtävällä. Kun nämä tehtävät ovat löytyneet, etsitään mihin oppituntiin kyseiset tehtävät kuuluvat. Seuraavaksi haetaan oppitunteja, joilla on sama opiskelukieli - opiskeltava kieli -pari kuin oppitunnilla, jonka tehtävään käännös halutaan. Viimeiseksi yhdistetään löydetyt oppitunnit ja tehtävät. Löydetyistä tehtävistä otetaan vastaukset ja ne palautetaan palvelimelta takaisin JSON-muotoisena listana käyttäjän selaimelle, jossa ne voidaan näyttää.

1.11 **Käyttöliittymän luonti**

Ensimmäisenä vaihtoehtona on tehtävien luontinäkymässä vastauksen kohdalle luoda nappi, josta aukeaa modaali, jossa on lista mahdollisista käännöksistä kyseiselle sanalle tai lauseelle (Kuva 7).



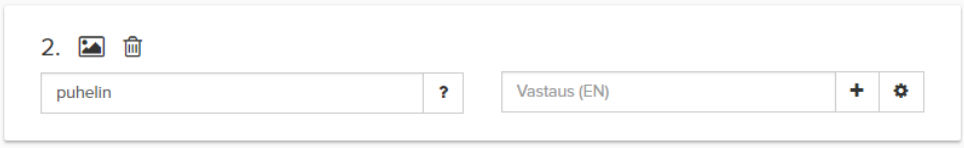
Kuva 7. Käännösehdotusnappi näytetään tehtävän vastauksen kohdalla.

Modaalissa näytettävät käännökset ovat allekkain ja niitä painamalla modaali sulkeutuu ja käännös siirtyy tehtävän vastauskenttään sen vastauksen kohdalle, josta nappia painettiin (Kuva 8). Tämän jälkeen kyseinen vastaus tallennetaan tietokantaan.

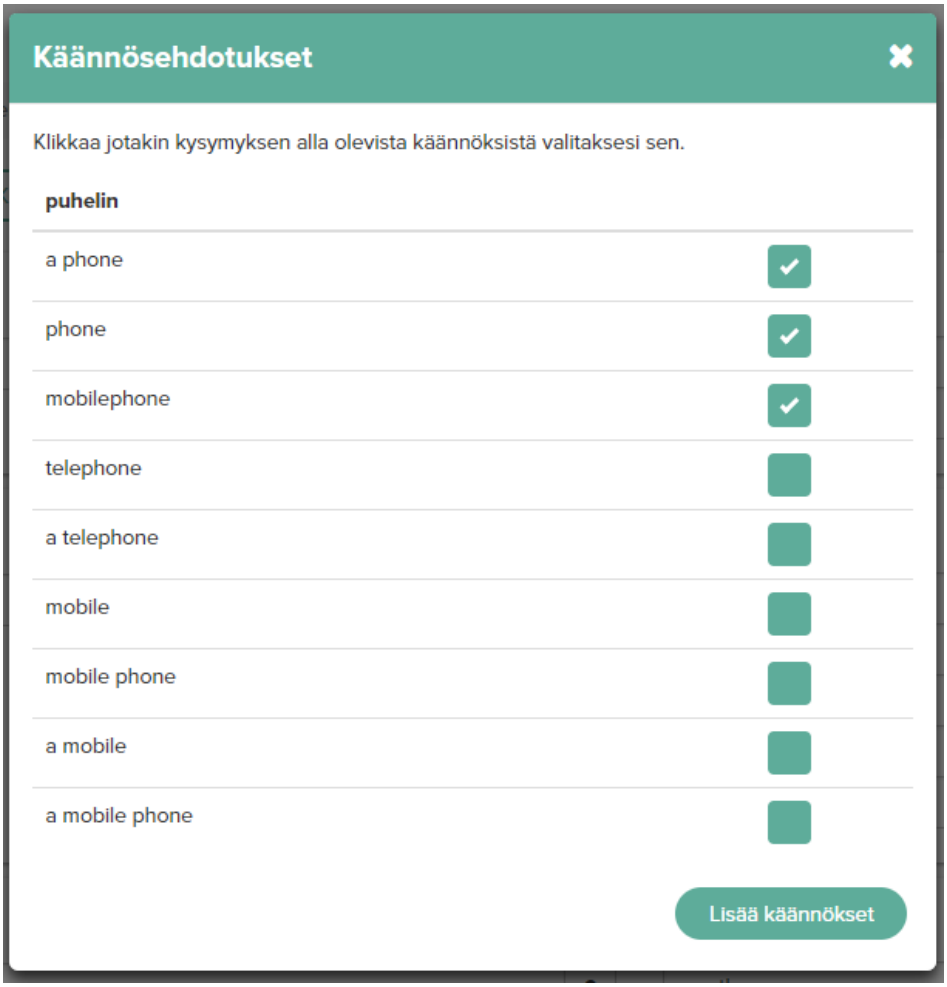
Kuva 8. Käännösehdotukset näkyvät modaalissa allekkain.

Toinen vaihtoehto olisi luoda nappi vastauksen sijasta kysymyksen kohdalle (Kuva 9) (Kuva 10). Siitä painamalla aukeaisi modaali, jossa olisi käännösten perässä valintaruutu (checkbox), josta valittaisiin kyseinen käännös tehtävälle ja käännöslistan alla olisi nappi, josta hyväksyttäisiin valitut käännökset tehtävälle (Kuva 11).

Kuva 9. Käännösehdotusnappi, joka on kysymyksen kohdalla, HTML-koodissa.



Kuva 10. Käännösehdotusnappi, joka näytetään tehtävän kysymyksen kohdalla.



Kuva 11. Käännösehdotusmodaali, jossa voidaan valita monta käännöstä kerralla.

Tämän jälkeen kaikki valitut käännökset luodaan tehtävälle ja ne tallennettaan tietokantaan. Tässä tapauksessa kuitenkin aluksi päädyttiin ensimmäiseen vaihtoehtoon eli luodaan napit jokaisen vastausvaihtoehdon kohdalle. Kuitenkin käytännön testausten jälkeen päädyttiin siirtämään nappi kysymyksen kohdalle, koska se yksinkertaisti käyttöliittymää ja nopeutti käännösten lisäämistä, sillä tällä tavoin voitiin lisätä monta käännöstä yhdellä kerralla samaan kysymykseen.

1.12 Tiedon esittäminen ja tallentaminen

Haetut tiedot esitetään sovelluksessa hyödyntäen HTML, CSS ja JavaScript -tekniikoita. Koska sovellus käyttää AngularJS-viitekehystä, niin täten käytetään sen tarjoamia tekniikoita tiedon esittämiseen. Tässä projektissa on käytössä AngularJS:in saatavilla oleva UI Bootstrap -kirjasto, joka helpottaa käyttöliittymien luontia huomattavasti. Bootstrap:n avulla saadaan käyttöliittymästä helposti hyvännäköinen ja helppokäyttöinen. Tiedot esitetään sovelluksen päälle avautuvassa modaali-ikkunassa, koska se soveltuu hyvin

tämän kaltaisen tiedon esittämiseen. Koska tiedot esitetään popup-ikkunassa, käytetään siinä hyväksi Twitter Bootstrap -kirjaston sisäänrakennettua popup-modaali-ikkunaa (Kuva 11).

Tietojen esittämistä varten luodaan sovelluksen päälle avautuva modaali-ikkuna hyödyntäen Angular-Bootstrap-kirjastoa, joka on siis niin sanottu wrapperi-kirjasto Bootstrap kirjaston ympärillä, jotta se toimisi saumattomasti AngularJS:n kanssa. Bootstap-modaalin käyttöä varten sovelluksen kontrolleriin injektoidaan (dependency injection) Angular-Bootstrapin $uibModal-moduuli. Uusi modaali luodaan ja näytetään sen open-metodilla. Metodi ottaa vastaan yhden parametrin, joka on objekti, jossa määritellään modaali-ikkunan käyttämä HTML-mallitiedosto (templateUrl), kontrolleri (controller), koko (size) ja sen kontrolleriin injektoitavat paikalliset parametrit (resolve). Tämä metodi palauttaa luodun modaalin, johon luodaan referenssi modalInstance-muuttujalla. Tässä tapauksessa halutaan tietää, mitä arvoja modaali palautti. Tietojen tuominen modaalista takaisin kontrolleriin tapahtuu kuuntelemalla modaalin tarjoamaa result-ominaisuutta, joka on Promise eli asynkroninen objekti ja sen palauttamat arvot saadaan sen then-metodilla tai vaihtoehtoisesti käyttäen await-avainsanaa. Tässä tapauksessa päädyttiin ensimmäiseksi mainittuun vaihtoehtoon. Se palauttaa funktion, jossa on parametrina löydetyt käännökset. Kun nämä on löydetty, ne lisätään tehtävään vastauksiin ja tallennetaan tietokantaan. (Kuva 12)

```
1069        // Show translation suggestions for the question word or sentence.
1070        this.showSuggestions = item => {
1071            const modalInstance = $uibModal.open({
1072                templateUrl: "../partials/modal_show_suggestions.html",
1073                controller: "modalInstanceShowSuggestionsController",
1074                size: "md",
1075                resolve: {
1076                    item: () => {
1077                        return item;
1078                    },
1079                    study_set: () => {
1080                        return this.study_set;
1081                    }
1082                }
1083            });
1084
1085            modalInstance.result.then(
1086                translations => {
1087                    // Add the translations to the item answers.
1088                    translations.forEach(translation => {
1089                        item.answer_json.list.push({ text: translation});
1090                    });
1091
1092                    // Save the study item to the database.
1093                    this.saveStudyItem(item);
1094                }
1095            );
1096        };
```

Kuva 12. Kysymykselle löydetyt käännökset esitetään Bootstrap-modaali-ikkunassa. Käyttäjän valittua kysymykset, ne tallennetaan sovelluksen tietokantaan.

1.13 **Toimintalogiikan laajentaminen**

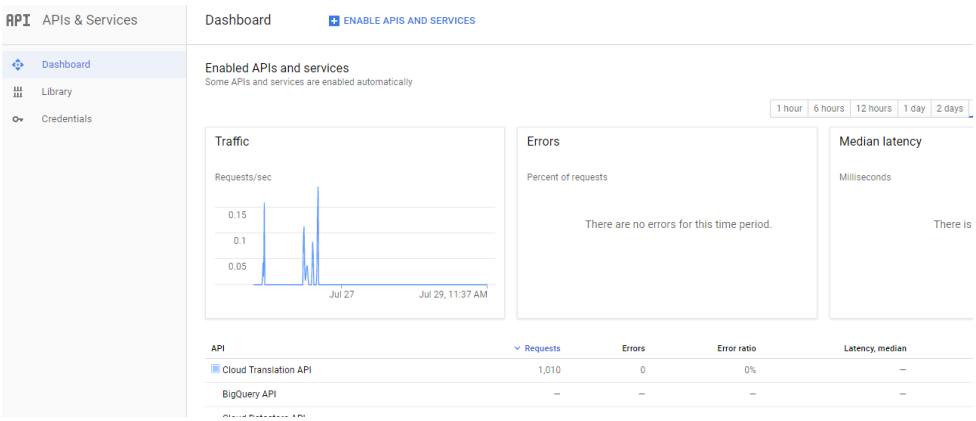
Tässä vaiheessa käännösten ehdottamiseen hyödynnetään ainoastaan sovelluksen tietokannassa jo olemassa olevia käännöksiä. Ne haetaan oppitunnin opiskelukielen ja opiskeltavan kielen mukaan etsimällä vastaavia kysymyksiä oppitunnin kieliparilla ja hakemalla näiden vastaukset. Tätä voidaan vielä laajentaa tekemällä päinvastainen haku. Eli jos vastaavia kysymyksiä ei löytynyt, haetaan vastauksia, jotka vastaavat käännettävää kysymystä opiskellulla kielellä.

Tästä syntyy kuitenkin ongelma, jos sovelluksen tietokannasta ei löydy tiettyä käännöstä, jolloin se ei pysty näyttämään sanalle tai lauseelle oikeaa käännöstä. Tämän ratkaisemiseksi päädytään hyödyntämään olemassa olevaa sanakirja- tai käännös-APIa. Ehtoina APIn valintaan ovat: mahdollisuus kääntää suomen kielelle ja suomesta muille kielille sekä sanavaraston riittävä laajuus. Tänä päivänä on olemassa monia varmasti hyviä vaihtoehtoja kielten kääntämiseen API:n avulla. Yksi näistä vaihtoehdoista on Wiktionary, jota Sanakirja.org hyödyntää (Sanakirja.org, n.d.). Toinen vaihtoehdoista on Microsoft Translator Text API, joka on Microsoftin pilvipohjainen ratkaisu (Microsoft, n.d). Kolmas ratkaisu on Google Cloud -pilvipalvelun tarjoama Google Cloud Translation API, joka on myös pilvipohjainen rajapinta ja hyödyntää koneoppimista (machine learning) käännösten ehdottamisessa (Google Cloud, n.d). Googlella on itseasiassa kaksi APIa käännösten ehdottamiseen: edellä mainittu Cloud Translation API ja Translate API, jota käytetään Google Kääntäjä -palvelussa. Ratkaisuksi valittiin Google Translate API sen yksinkertaisen käytettävyyden, ilmaisuuden ja nopean käyttöönoton ansioista. Kuitenkin pian testaamisen aloittamisen jälkeen huomattiin, että pyynnöt eivät enää menneet läpi, joten Translate API päädyttiin vaihtamaan maksulliseen Cloud Translation APIin.

Googlen Cloud Translation APIn käyttöön on saatavilla kirjastoja monille yleisimmille ohjelmointikielille kuten C#, Java, PHP, Python, Ruby ja Node.js (JavaScript), joista viimeksi mainittu on käytössä tässä projektissa. API-kirjasto ladattiin käyttäen npm-paketin hallintaa, joka on oletuksena Node.js:n paketinhallintajärjestelmä. Kirjaston lataaminen tapahtuu komennolla: npm install --save @google-cloud/translate. Komennon ”--save”-parametri lataa tarvittavat riippuvuudet (dependencies) kyseisen paketin käyttöä varten. Lataamisen jälkeen kirjasto siirtyy projektikansion juuressa sijaitsevaan node_modules-nimiseen kansioon, johon projektin kaikki npm:n kautta ladatut npm-paketit siirtyvät. Tämän jälkeen sen integrointi sovelluksen koodiin on suhteellisen yksinkertaista ja se onnistuu muutamalla koodirivillä. Ensinnä lisätään Cloud Translation API -paketti käyttöön koodissa Node.js:n require-komennolla. Tämän jälkeen lisätään koodi, jolla haetaan käännökset käännettäville merkkijonoille (Kuva 13). API:n diagnostiikka ja hallinnointi tapahtuu Google Cloud Platform:n avulla. (Kuva 14) API:n käyttö vaatii Google-tunnuksen, johon on liitetty luottokortti, käytön mukaan laskutusta varten.

```
245         // Translate the question to the studied language.
246         const res = await translator.translate(question, { from: study_lang, to: studied_lang });
247         console.log('Translator response:', res, res.data.translations); // debug
248         if (res[0]) found_translations.push(res[0]);
```

Kuva 13. Google Cloud Translation API:n integrointi koodiin onnistuu parilla koodirivillä.



Kuva 14. Google Cloud Translation API:n hallinnointi tapahtuu Google Cloud Platformin kautta.

Käännösehdotuksia on myös suodatettava jollain tavalla väärien käännösten välttämiseksi. Kuten sallimalla ainoastaan opettajien tekemien käännösten ehdottaminen tai edellyttämällä tietyn minimilukumäärän käännökselle, jotta se näytetään. Eräs vaihtoehto on myös huonojen käännösten raportointisysteemi joko manuaalisella tai automaattisella tarkistuksella. Tällöin käännösten perässä näytettäisiin raportointinappi, jota painamalla käännös raportoidaan. Jos käytössä olisi manuaalinen tarkistus, niin raportoitaessa käännöksen tekijälle tulisi ilmoitus, että heidän on tarkastettava käännös, ennen kuin se voitaisiin näyttää uudelleen. Automaattisella tarkistustavalla algoritmi laskisi raportointien määrää ja päättäisi milloin käännös hylättäisiin. Tällöin käännös poistuisi näkyvistä, kun käännöstä on tarpeeksi monta kertaa äänestetty huonoksi. Käännöksen tekijälle näkyisi, että hänen pitää korjata käännös tehtävien materiaaliluontinäkymässä ennen kuin se tulisi näkyviin uudestaan.

1.14 **Optimointi**

Koodin optimointi on olennainen osa sovelluksen kehitystä ja sitä tehdään sovelluksen käytettävyyden ja sulavamman käytön mahdollistamiseksi. Toiminnallisuuden optimointi aloitetaan poistamalla koodista hitaiksi havaittuja koodirakenteita, metodeja sekä funktioita ja vaihtamalla ne nopeammiksi todetuiksi vastineiksi. Myös kaikki ylimääräinen tiedon prosessointi, jonka voi tehdä käyttäjän selaimen puolella, siirretään palvelimelta selaimelle, tällöin kevennetään palvelimen kuormitusta ja hyödynnetään käyttäjien laitteiden suorituskykyä. Tämän lisäksi tietokannan hakujen suorituskykyä parannetaan lisäämällä indeksit käytettyihin hakuparametreihin, jolloin tieten löytämiseksi sen ei tarvitse skannata koko taulua, vaan

se voi etsiä hakutuloksia hakuparametrejä vastaavista indekseistä (Kuva 15). JSON-sarakkeen korvaaminen JSONB-tietotyypillä (data type) tai tehtävien vastausten siirtäminen oman tauluunsa luultavasti parantaisi haun suorituskykyä, sillä ne voitaisiin indeksoida.

```
5 CREATE INDEX study_item_question_lower_index
6     ON public.study_item USING btree
7     (lower(question::text) COLLATE pg_catalog."default")
8     TABLESPACE pg_default;
```

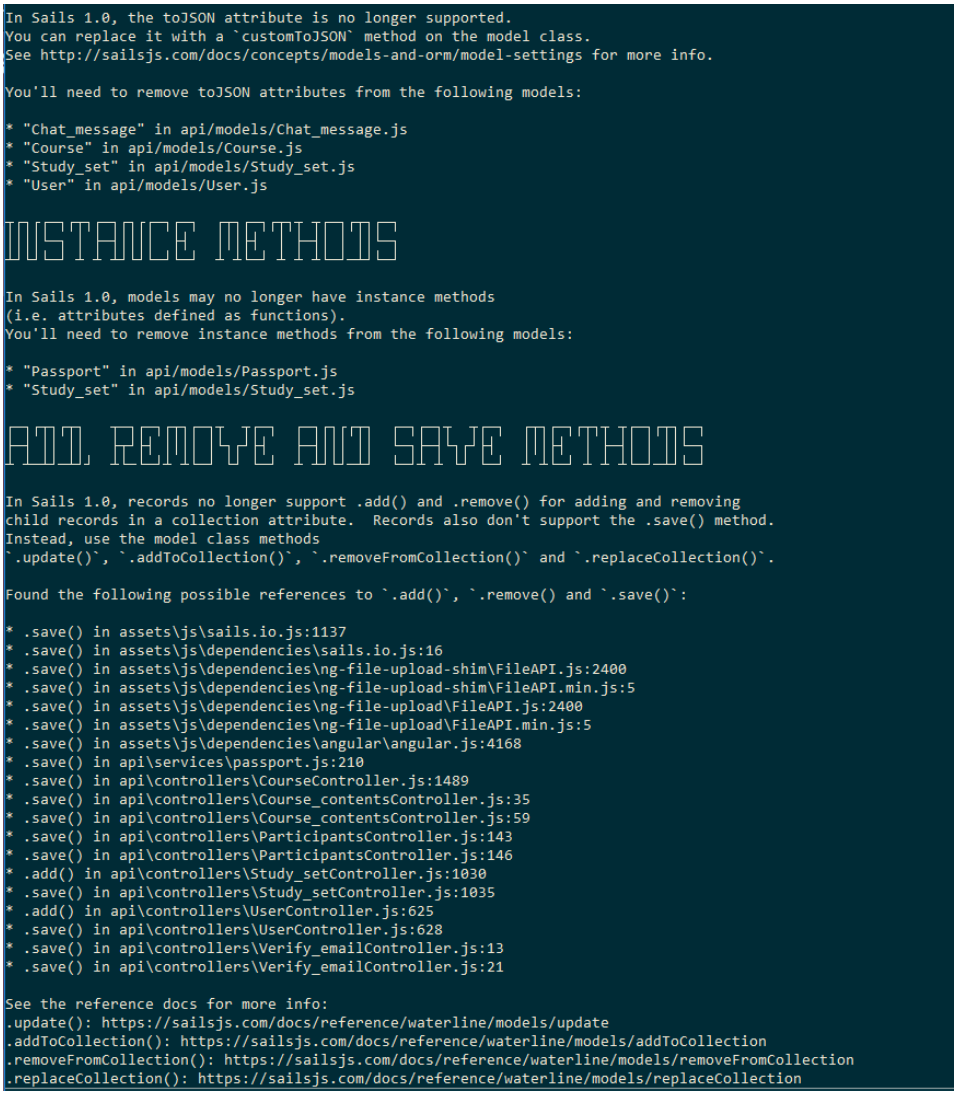
Kuva 15. Tehtävän vastaukseen luotu kirjainkokoriippumaton indeksi nopeuttaa niiden hakemista tietokannasta. Kuvan SQL-koodin mukainen indeksi on luotu PostgreSQL-hallintaan luodulla pgAdmin-ohjelmalla.

Koska käännösten hakemisessa tietokannasta on monia välihakuja, jonka avulla lopullinen tieto haetaan, voidaan niistä valita vain tarvitut sarakkeet (column) kaikkien sarakkeiden sijasta, jolloin hakua saadaan nopeutettua, kun tietokannasta ei haeta palautettavan lopputuloksen kannalta turhaa tietoa. Myös haut käännettävän sanan perusteella voitaisiin siirtää välimuistiin ohjelman koodissa tai selaimessa käyttäen JavaScriptin tarjoamaa sessionStorage- tai localStorage -ominaisuutta, jolloin ne haetaan käyttäjän selaimen välimuistista palvelimen sijasta, nopeuttaen toistuvia hakuja ja samalla vähentäen palvelimen kuormitusta.

1.15 **Ongelmat**

Serverin Sails.js-viitekehystä päätettiin päivittää versiosta 0.12 versioon 1.0 opinnäytetyön aikana siihen varsinaisesti liittymättömistä syistä. Pääasiallisina syinä olivat suorituskyvyn ja tietoturvan parantaminen. Muita syitä olivat kehitystyön ja projektin ylläpitämisen helpottaminen tulevaisuutta ajatellen. Tämä kuitenkin aiheutti sen, että opinnäytetyötä ei voinut jatkaa ennen kuin tämä päivitys oli valmis, sillä viitekehys ei ollut projektin versionhallinnassa mukana. Päivitys vaati monia muutoksia, koska ennen versiota 1.0 Sails.js:n kehitys oli beta-asteella. Monia käytettyjä metodeja ja funktioita jouduttiin muuttamaan ja korvaamaan uusilla metodeilla ja funktioilla sekä tekemään tarvittavat muutokset muualla koodissa. Tämä hidasti itse projektin edistymistä jonkin verran. Onneksi Sails.js:n kehitystiimi oli tehnyt päivittämistä varten luodun migraatiotyökalun, jota päätettiin käyttää tässä päivittämistä helpottamaan (Kuva 16). Työkalu kertoi mitä asioita projektissa on muutettava, jotta se toimii uudella versiolla, ja joitain asioita se osasi päivittää automaattisestikin (Kuva 17).

Kuva 16. Sails.js:n kehitystiimin päivitystä varten luoma työkalu helpotti projektin kehitystä luomalla päivitetyt versiot projektin tiedostoista.



Kuva 17. Sails.js:n migraatiokalu helpotti sen päivittämistä kertomalla versioiden välillä tapahtuneista muutoksista.

Päivitys myös rikkoi aiemmin käytössä olleen Sails-Hook-Autoreloadin-koodikoukun (code hook), joka helpotti kehitystyötä nopeuttamalla muutosten testaamista viemällä serveripuolen koodiin tehdyt muutokset automaattisesti ilman sen kokonaan uudelleenkäynnistämistä. Sails.js-serverin käynnistys tässä projektissa vie noin 3-10 sekuntia tämän hetkisellä kehitysympäristöllä. Ja sen tekeminen joka kerta, kun muutoksia halutaan testata, hidastaa kehitystä ja iterointia huomattavasti. Valitettavasti Sails-Hook-Autoreloadin ylläpitäminen on lopetettu noin pari vuotta sitten, joten sen päivittäminen ei enää ollut mahdollisuus. Ratkaisuna oli olemassa kaksi vaihtoehtoa: Nodemon ja Forever, jotka ovat molemmat tehty serveripuolen kehitystä helpottamaan. Ratkaisuna päädyttiin käyttämään ensimmäistä eli Nodemonia, koska se oli entuudestaan tuttu tämän projektin tekijälle aiemmasta Node.js-projektista. Nodemon asennettiin käyttäen npm-paketinhallintaa komennolla: ”npm install nodemon”. Sen jälkeen projektin juureen luotiin nodemon.json-asetustiedosto, jossa määriteltiin Nodemonin vahtimat tiedostopolut. Muutosten vieminen on huomattavasti hitaampaa Nodemonilla kuin Sails-Hook-Autoreloadilla, sillä Sails-

Hook-Autoreload vei muutokset alle sekunnissa Nodemonin noin 3-10 sekunnin sijasta, mutta se on kuitenkin kätevämpää kuin serverin käynnistäminen käsin joka kerta, kun tehdään muutoksia. Eli se on kompromissina varsin hyvin toimiva ratkaisu.

Projektissa käytössä ollut ilmainen Google Translate API lakkasi toimimasta kesken kehityksen johtuen sen pyyntömäärärajoituksesta, joten sille oli löydettävä korvaava vaihtoehto. Käytännössä ratkaisuna oli kaksi varteenotettavaa vaihtoehtoa: Microsoft Translator ja Google Cloud Translation API. Ratkaisuna päädyttiin Googlen maksulliseen Cloud Translation API:in, joka on osana Googlen Cloud Platform -pilvipalvelualustaa.

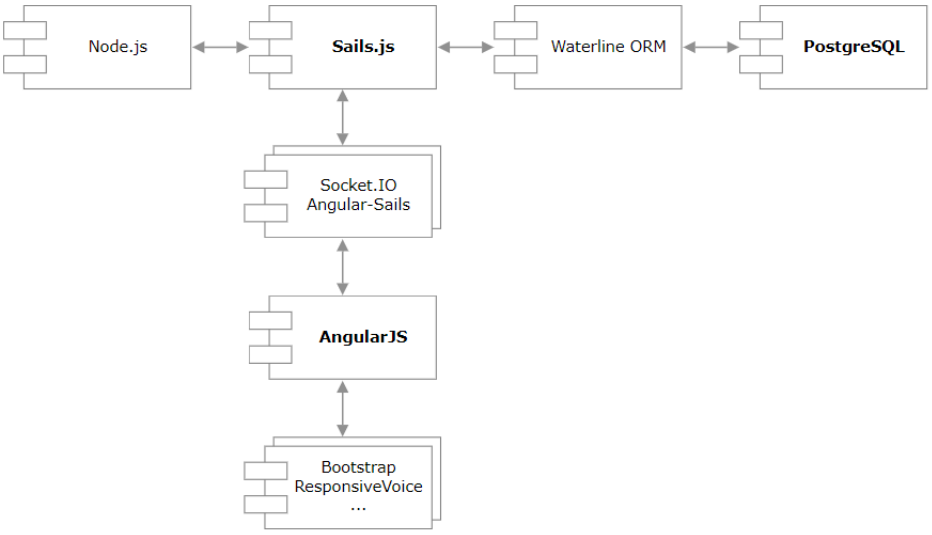
Sails.js:n mukana tulevassa Waterline ORM:ssa on varsin rajoitettu tuki SQL-aggregaattifunktioille kuten SUM ja MAX. Edellä mainituista jälkimmäinen puuttuu tällä hetkellä kokonaan Waterlinesta. Tämän takia joitain tiedon hakuja tietokannasta joudutaan toteuttaa useammalla eri kyselyllä ja kokoamalla nämä yhteen tai kirjoittamalla raa'alla SQL-syntaksilla hyödyntämättä ORM:n tuomia etuja ja ominaisuuksia.

Ajan puutteen vuoksi projektin ominaisuuksia jouduttiin karsimaan alkuperäisestä suunnitelmasta. Tarkoituksena oli alun perin luoda tälle käännösjärjestelmälle tietokantaan erillinen taulu, jossa seurataan ja ylläpidetään käännösten laatua tarkkailemalla käännösten käyttöä ja poistoa. Lopulta kuitenkin päädyttiin hakemaan käännökset olemassa olevien tehtävien tauluista. Lisäksi tällä hetkellä siinä ei ole mitään laaduntarkkailujärjestelmää. Käännöksiä ei myöskään seulota millään tavalla, mikä on ongelma käyttäjämäärän kasvaessa, koska se tuo myös vääjäämättä mukanaan häiriköitä ja niin sanottuja trolleja, joiden käännökset näkyvät mukana niin sanottujen hyvien käännösten seassa, heikentäen sovelluksen käyttäjäkokemusta ja laatuvaikutelmaa loppukäyttäjän kannalta.

Myöskään johdannossa mainittu kokeiden tehtävien tarkastamisen parantaminen ei mahtunut mukaan tähän projektiin ajan puutteen vuoksi. Lisäksi käännösten ehdotussysteemin laajuus kasvoi projektin edetessä. Siihen haluttiin lisää ominaisuuksia kuin oli alun perin kaavailtu. Ja uhkana oli niin sanottu ”feature creep” eli ominaisuuksien määrän kasvaminen liialliseksi ja toiminnallisuuden monimutkaistuminen.

1.16 **Lopputulos**

Ongelmista huolimatta lopputuloksena saatiin kuitenkin luotua toimiva käännösten ehdotussysteemi. Järjestelmä hyödyntää taustapuolella Node.js ajoympäristöä, jonka päällä ajetaan Sails.js viitekehystä. Sovelluksen tiedot tallennetaan PostgreSQL-tietokantaan. Etupuolella puolestaan toimii AngularJS-viitekehys ja sitä avustavat kirjastot, kuten Bootstrap. (Kuva 18)



Kuva 18. Reactored-sovelluksen kokonaisrakenne.

Tehtäviä luodessa sovellus osaa ehdottaa käännöksiä tehtävän kysymykselle nappia painamalla. Käyttäjän painaessa nappia, käynnistyy sovelluksen tietokannasta haku, joka etsii käännöksiä kyseiselle kysymykselle. Ensin haetaan samoja kysymyksiä samoilla kieliasetuksilla ja jos näitä löytyy, niin niiden vastaukset haetaan ja näytetään käännöslistalla. Jos niillä ei löydy yhtään käännöksiä, haetaan vastauksia, jotka vastaavat kysymystä, käänteisillä kieliasetuksilla ja näiden kysymykset haetaan ja näytetään käännöslistalla. Jos vielä tämänkään jälkeen ei löydy käännöksiä, niin käytetään Googlen Cloud Translate API:a kysymyksen kääntämiseen. Sen jälkeen löydetyt käännösehdotukset näytään käyttäjälle. Ehdotukset näytetään sovelluksen päälle aukeavassa modaali-ikkunassa. Siinä käyttäjä voi valita haluamansa käännökset tehtävän kysymykseen ja käyttäjän valitessa haluamansa käännökset ja painamalla tallenna-nappia sovellus tallentaa ne automaattisesti luotuun tehtävään.

YHTEENVETO

Projektin aihe oli mielestäni kiinnostava ja siinä oppi paljon hyödyllisiä asioita sovelluskehityksestä oikeassa työelämässä. Sain siis projektin toimeksiannon työnantajaltani ja minulle annettiin vapaat kädet toteuttamisen suhteen. Olin jo jonkin verran ehtinyt tehdä kehitystä kyseisessä sovelluksessa, ennen kuin aloitin tekemään tätä opinnäytetyötä. Vaikka tämä oli ensimmäinen projekti, jossa olin käyttänyt Sails.js-viitekehtystä, pääsin kuitenkin suhteellisen nopeasti kehitykseen mukaan, koska olin käyttänyt Node.js:ia, jonka päälle Sails.js rakentuu, aikaisemmassa projektissa. Kyseessä ei siis ollut täysin ventovieras ympäristö minulle, mutta opin silti uutta sovelluksesta ja sen kehittämisestä, johtuen sovelluksen laajuudesta. Koska kyseessä oli jo aiemmin kehityksessä ollut sovellus, pyrin hyödyntämään kehityksessä jo käytettyjä tekniikoita. Mutta projektin kehityksen edetessä, tarvitsi sen tavoitteiden saavuttamiseksi hyödyntää myös siinä aiemmin olemattomia tekniikoita, kuten Googlen Cloud Platform -alustan tarjoamaa Cloud Translate API:a. Onneksi Google oli dokumentoinut alustan ja API:n hyvin, joten ongelmia sen käyttöönoton ja käytön kannalta ei tullut.

Kehitys eteni pääosin hyvin, mutta oli matkalla muutama ongelmakin. Esimerkiksi sovelluksen palvelinpuolella käytettyä Sails.js-viitekehystä päätettiin päivittää kesken projektin ja se hajotti joitain sovelluksessa käytettyjä asioita. Tämä siis myös vaikutti projektin etenemiseen, koska viitekehys ei ollut mukana projektin versionhallinnassa, joten projektin erillinen kehityshaara ei sivuuttanut ongelmia. Projektin eteneminen siis pysähtyi, ennen kuin ongelmat saatiin korjattua. Sovellus kuitenkin saatiin päivitettyä Sails.js-tiimin tekemän erinomaisen päivitystyökalun avulla ja ongelmat korjattua ja projektin eteneminen jatkui taas. Myös joitain projektin kuvauksessa esitettyjä tavoitteita ja ominaisuuksia jouduttiin karsimaan tai jättämään kokonaan pois. Esimerkiksi johdannossa mainittu kokeiden tarkistusmetodin parantaminen jäi kokonaan pois ajan puutteen vuoksi.

Vaikka projektin tavoitteet muuttuivatkin matkan varrella, saatiin se silti vietyä kunniakkaasti loppuun asti. Projektia tehdessä tuli myös uusia ideoita järjestelmän jatkojalostamiseen tulevaisuudessa. Käytössä olleet kirjastot ja viitekehykset mahdollistivat tavoitteiden saavuttamisen ja projektin melko mutkattoman toteuttamisen. Sovelluksessa tehtävien luonnissa pystytään nyt lisäämään käännöksiä kysymyksille, vaikka tekijä ei itse tietäisikään kysymyksen käännöstä tai hän haluaa lisätä nopeasti monta vastausta siihen, mikä helpottaa tehtävien tekemistä ja avaa uusia mahdollisuuksia palvelua käyttäville kielten opiskelijoille ja opettajille.

LÄHTEET

AngularJS. n.d, AngularJS. Noudettu 21.11.2017 Osoitteesta https://angularjs.org/.

AngularJS. n.d, AngularJS. Noudettu 27.10.2018 Osoitteesta https://docs.angularjs.org/api/ng/function/angular.element.

Bootstrap. n.d, Boostrap. Noudettu 20.11.2018 Osoitteesta https://get-bootstrap.com/.

Chacon, S. (2014). Pro Git. New York, New York, Yhdysvallat: Apress.

Devblast. 2016, The Differences Between Angular 1.X and Angular 2. Noudettu 21.11.2017 Osoitteesta https://devblast.com/b/differences-angular-1-x-angular-2.

ECMA-international, 2015, ECMA-international, Noudettu 7.10.2018 Osoitteesta http://www.ecma-international.org/ecma-262/6.0/.

Google Cloud. n.d, Cloud Translation, Noudettu 20.10.2018 Osoitteesta https://cloud.google.com/translate/.

JQuery. n.d, jQuery API. Noudettu 30.11.2017 Osoitteesta https://api.jquery.com.

Krol, J. (2014). Web Development with MongoDB and Node.js. Birmingham, Yhdistynyt kuningaskunta: Packt.

Microsoft. n.d, Microsoft Translator Text API, Noudettu 20.10.2018 Osoitteesta https://www.microsoft.com/en-us/translator/business/translator-api/.

Node.js. n.d-a, Node.js. Noudettu 2.11.2017 Osoitteesta https://nodejs.org/en/.

Node.js. n.d-b, About Node.js. Noudettu 3.11.2017 Osoitteesta https://nodejs.org/en/about/.

PostgreSQL. n.d, About. Noudettu 10.11.2017 Osoitteesta https://www.postgresql.org/about/.

Sails.js. n.d, Sails.js, Noudettu 10.11.2017 Osoitteesta https://sailsjs.com/.

Sanakirja.org. n.d, Sanakirja.org, Noudettu 20.10.2018 Osoitteesta https://www.sanakirja.org/.

Texeira, P. (2012). Professional Node.js: Building Javascript Based Scalable Software. New York, New York, Yhdysvallat: John Wiley & Sons.