Pasi Semykine

Sovelluskehitys MEAN-ohjelmistopinoa käyttäen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

30.11.2017

Metropolia

Tekijä Otsikko Sivumäärä Aika	Pasi Semykine Sovelluskehitys MEAN-ohjelmistopinoa käyttäen 30 sivua 30.11.2017
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tietotekniikka
Ammatillinen pääaine	Ohjelmistotekniikka
Ohjaajat	Lehtori Simo Silander

Työn tavoitteena oli kehittää kielenoppimista tukeva sovellus, jolla käyttäjä voi lisätä omia sanoja tietokantaan ja harjoitella sanojen muistamista, kirjoittamista ja mahdollisesti erilaisen kirjoitusjärjestelmän näppäimistöasettelua.

Työn kehittämiseen käytettiin MEAN-ohjelmistopinoa, jonka nimi tulee sen jäsenten nimien alkukirjaimista: MongoDB, Express.js, Angular ja Node.js. Kehityksen aikana opittiin ohjelmistopinon jäsenten käyttäminen, niiden heikkoudet ja vahvuudet.

Kehityksessä luotiin MongoDB-tietokanta pilveen ja Express.js-kirjastoa käytettiin kehitettävän ohjelmiston palvelinrajapintana, joka haki ja lähetti tietoa MongoDB-tietokantaan.

Sovelluksen käyttöliittymänä ja selainpuolena toimi Angular-ohjelmistokehys, joka oli yhteydessä Express.js-rajapintaan, ja molemmat näistä käyttivät Node.js-palvelinta.

Kehitettyä ohjelmistoa ei tämän työn aikana tai lopussa julkaistu, vaan sen kehitys ja uusien ominaisuuksien lisääminen tulee jatkumaan tulevaisuudessa, mutta tavoitteena on jonain päivänä julkaista valmis tuote muidenkin käyttöön.

Avainsanat	MEAN, verkkosovellus, MongoDB, Express, Angular, Node.js

Author Title Number of Pages Date	Pasi Semykine Software Development with MEAN Software Stack 30 pages 30 November 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Simo Silander, Senior Lecturer

The goal of the project was to develop a tool for supporting language learning, memorization of the language and training the keyboard layout of a different writing system.

For the development of the project, the MEAN software stack was used. The stack's name comes from the first letter of the names of its member software: MongoDB, Express.js, Angular and Node.js. The usage of the software included in the stack, as well as their strengths and weaknesses were learned during the development process.

During development, a Mongo database was created in the cloud and the Express.js library was used as an interface for the developed application that fetched and submitted data to the Mongo database.

The user interface and front-end were created with the Angular framework that communicated with the Express.js interface and both of them were hosted on a Node.js server.

The software developed during the project was not yet released, but its development will continue, and new features will be added in the future. It is, however, the goal to someday publish the present project for others to use, too.

Keywords	MEAN, web app, MongoDB, Express, Angular, Node.js

Sisällys

Lyhenteet

1 Johdanto 1

2 Ohjelmistopinot 1

3 MEAN-ohjelmistopinon jäsenet 2

3.1 MongoDB 3
3.1.1 Kuvaus 3
3.1.2 Edut ja haitat verrattuna SQL tietokantaan 3
3.2 Express.js 5
3.3 Angular 5
3.3.1 Alkuperä ja AngularJS 5
3.3.2 Angular CLI 6
3.3.3 Angularin arkkitehtuuri 7
3.3.4 Angularin ominaisuuksia 7
3.3.5 TypeScript 9
3.3.6 Vaihtoehtoinen työkalu Angularille - React 9
3.4 Node.js 9
3.5 Ulkoiset lisäosat 10

4 Sovelluksen suunnittelu 13

4.1 Inspiraatio ja samantyyliset ohjelmistot 13
4.2 Suunnitellut ominaisuudet 15

5 Sovelluksen kehitys 16

5.1 Projektin aloitus ja palvelinpuoli 16
5.2 Selainpuolen kehitys 17

6 Jatkosuunnitelmat, johtopäätökset ja yhteenveto 26

6.1 Kehityksen jatkosuunnitelmat 26
6.2 Johtopäätökset 26
6.3 Yhteenveto 27

Lähteet 29

Lyhenteet

MEAN	MongoDB, Express, Angular, Node.js. Ohjelmistopino, jonka nimi tulee sen jäsenten alkukirjaimista.
JSON	JavaScript Object Notation. Avoimen standardin tietomuoto, joka on suunniteltu olemaan helposti luettava ihmisille.
SQL	Structured Query Language. Reaalitietokannan kyselykieli, jolla voidaan tehdä muutoksia yhteensopivaan tietokantaan.
NoSQL	Not only SQL. Tietokantatyyppi joka poikkeaa perinteisestä relaatiotietokannasta.
API	Application programming interface. Ohjelmointirajapinta. Mahdollistaa eri ohjelmien siirtää tietoa keskenään.
HTML	Hypertext Markup Language. Kuvauskieli, jolla näytetään verkkosivuja.
CSS	Cascading Style Sheets. Tyyliohjedokumentti verkkosivuille. Määrittelee verkkosivun ulkonäköä.
NPM	Node Package Manager. Node.js:n mukana tuleva pakettien hallintatyökalu.
LAMP	Linux, Apache, MYSQL, Perl or PHP or Python. Linux-pohjainen avoimen lähdekodin ohjelmistopino.
XAMPP	Crossplatform, Apache, MariaDB, PHP, Perl. Monille käyttöjärjestelmille sopiva avoimen lähdekoodin ohjelmistopino.
WINS	Windows Server, Internet Information Services, .NET, SQL Server. Windows-pohjainen ohjelmistopino.
CLI	Command Line Interface. Komentoriviliittymä.

1 Johdanto

MEAN-ohjelmistopino koostuu MongoDB-tietokannasta, Express.js-verkkosovelluskehyksestä, Angular-verkkosovellusalustasta ja Node.js-suoritusympäristöstä. Kaikki pinon teknologiat ovat melko uusia, ja ne tuovat suuria uudistuksia omiin aihealueisiin. Tässä raportissa tullaan perehtymään MEAN-ohjelmistopinon teknologioiden historiaan, niiden hyötyihin ja haittoihin sekä verrataan niitä muihin vastaaviin teknologioihin.

Raportissa tullaan näkemään, miltä uudempi tietokanta näyttää verrattuna perinteisempään tietokantaan. Sekä raportissa tutustutaan ohjelmointikieleen, joka luotiin parannuksena jo aiemmin olemassa olevaan kieleen, mutta toimiakseen se kuitenkin käännetään siihen kieleen, minkä korvaaja sen pitäisi olla.

Aihe tuli valittua, koska minulla oli halu insinöörityön tekemisen ohella oppia jotakin kokonaan uutta. Olen myös kiinnostunut Angular-kehityksestä, ja MEAN-pinon muut jäsenet sopivat hyvin tukemaan Angular-sovelluksia.

2 Ohjelmistopinot

Ohjelmistopino, englanniksi software stack tai solution stack, on ryhmä ohjelmistoja, jotka sopivat hyvin yhteen luodakseen suuremman kokonaisuuden. Yhdessä niillä voi luoda tuotteen, joka olisi vaikeaa tai mahdotonta luoda vain yhdellä ryhmän jäsenellä. On myös tarkoitus saada aikaan tuote, joka ei tarvitse pinon ulkopuolisia ohjelmistoja toimiakseen.

Tietenkin ohjelmistopinoja voi olla lukemattomia määriä ja kuka tahansa voi päättää, että hän tykkää jostain tietystä ryhmästä ohjelmia ja käyttää niitä yhdessä. Mutta kun ohjelmistokehityksessä puhutaan ohjelmistopinosta, yleensä tarkoitetaan jotain yleistä pinoa, jonka jäsenten on todistettu toimivan hyvin yhdessä. On yleistynyt tarpeeksi, että pinon nimi tunnistetaan ja jäsenten jatkuvaa yhteensopivuutta tuetaan.

Tällaisten, yleisten ohjelmistopinojen suuri etu on sen yleisyys kehitysyhteisöissä. Kun kaikki käyttävät samaa pinoa projekteihinsa, niin on helppo saada tukea omaan

projektiinsa, kun yhteisön muut jäsenet törmäävät samoihin ongelmiin ja voivat jakaa niihin toimivia ratkaisuja. Tällainen on paljon vaikeampaa, jos eri ihmisillä on yhteistä vain yksi sovellus eikä koko sovelluspino, joka on todennäköisesti lähes koko kehitysympäristö. [1.]

LAMP (Linux, Apache, MYSQL, Perl or PHP or Python)

LAMP on yksi vanhemmista ohjelmistopinoista, joka on mainio vaihtoehto dynaamisten sivujen ja ohjelmien kehittämiseen. Kaikki ohjelmistot ovat avointa lähdekoodia. Tämä on yksi syy, miksi pino on saanut suosionsa ja miksi sille löytyy paljon tukea.

XAMPP (Crossplatform, Apache, MariaDB, PHP, Perl)

XAMPP on mahtava pino, joka saa nopeasti käytettävän palvelimen pystyyn, koska se sisältää myös ohjauspaneelin, jolla on helppo hallita pinon jäseniä. Siksi sitä on hyvä käyttää vaikka vain Apachen käyttöön, sillä sen käyttöönotto XAMPP-pinon ohjauspaneelilla on vaivattomampaa kuin itse Apachen asennus erikseen.

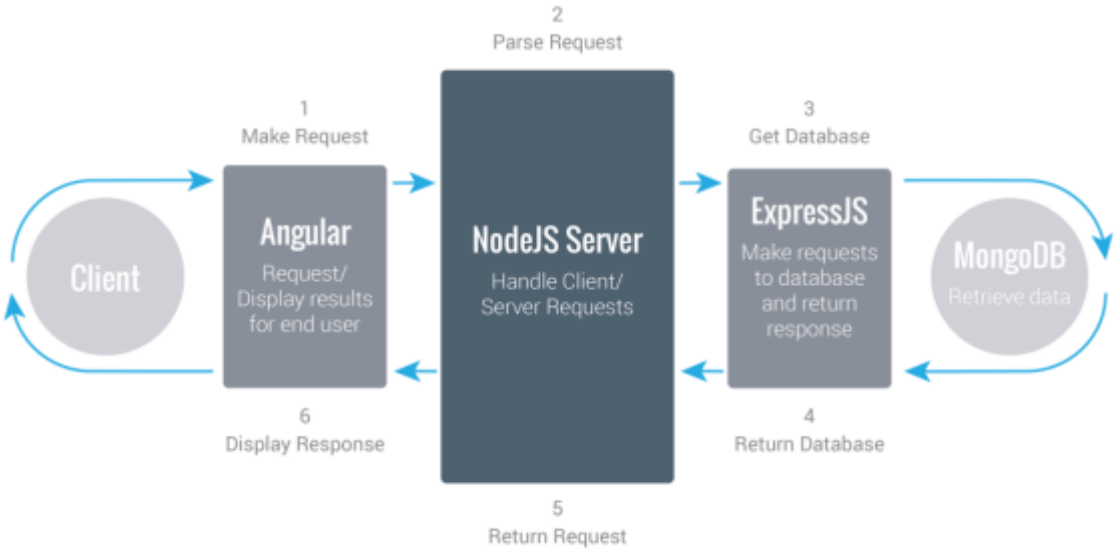
Kuten on yleistä muissa suosituissa ohjelmistopinoissa, XAMPP on myös avointa lähdekoodia ja sisältää palvelinohjelmiston lisäksi tietokannan. XAMPP-pinon helppo asennus ja saatavuus kaikille yleisimmille tietokonekäyttöjärjestelmille on hyvä syy pitää se mielessä aina, kun alkaa kehittää verkkoprojektia.

WINS (Windows Server, Internet Information Services, .NET, SQL Server)

WINS-sovelluspino käyttää Microsoftin kehittämiä ohjelmistoja ja poikkeaa yleisimmistä sovelluspinoista niin, ettei se ole avointa lähdekoodia. [2.]

3 MEAN-ohjelmistopinon jäsenet

MEAN-ohjelmistopinoon kuuluvat MongoDB-tietokanta, Express.js-verkkosovelluskehys, Angular-verkkosovellusalusta ja Node.js-suoritusympäristö. Kuva 1 visualisoi miten nämä sovellukset kommunikoivat keskenään.



Kuva 1. MEAN-ohjelmistopinon jäsenten suhteet toisiinsa visualisoituna. Käyttäjälle näytetään Angular-näkymä, josta hän voi tehdä pyyntöjä. Nämä pyynnöt menevät Node.js-palvelimen kautta Express.js-palvelinkehyksen käsiteltäväksi, Express.js tekee tarvittavan kyselyn MongoDB-tietokantaan. Kun Express saa vastauksen tietokannalta, tämä välittää tiedot takaisin Node.js-palvelimen kautta Angular-näkymän näytettäväksi. [3.]

3.1 MongoDB

3.1.1 Kuvaus

MongoDB on Document Store -tyyppinen NoSQL-tietokantajärjestelmä. NoSQL-tietokantatyypin nimi tulee englanninkielisistä sanoista Not only SQL, jossa SQL viittaa perinteisempiin tietokantoihin. MongoDB-tietokantarakenne jäljittelee JSON-tiedostomuotorakennetta ja on hyvin skaalautuva isoihin tietokantoihin.

3.1.2 Edut ja haitat verrattuna SQL tietokantaan

MongoDB mahdollistaa monen asian tekemisen yksinkertaisemmin. Kuvassa 2 nähdään heti alussa, että MongoDB:lle ei tarvitse luoda taulua, sillä itse data määrittelee tietokantarakenteen. Lisäksi nähdään, että molemmat datan hakeminen ja tallentaminen ovat hieman yksinkertaisempaa, sillä ei tarvitse valmistaa kyselykielilauseita.

SQL

```
DDL: create table contact ( … )

init()
{
  contactInsertStmt = connection.prepareStatement
  ("insert into contact ( id, name ) values ( ?,? )");
  fetchStmt = connection.prepareStatement
  ("select id, name from contact where id = ?");
}

save(Map m)
{
    contactInsertStmt.setString(1, m.get("id"));
    contactInsertStmt.setString(2, m.get("name"));
    contactInsertStmt.execute();
}

Map fetch(String id)
{
    Map m = null;
    fetchStmt.setString(1, id);
    rs = fetchStmt.execute();
    if(rs.next()) {
      m = new HashMap();
      m.put("id", rs.getString(1));
      m.put("name", rs.getString(2));
    }
    return m;
}
```

MongoDB

```
DDL:  none

save(Map m)
{
    collection.insert(m);
}

Map fetch(String id)
{
    Map m = null;
    DBObject dbo = new BasicDBObject();
    dbo.put("id", id);
    c = collection.find(dbo);
    if(c.hasNext()) {
      m = (Map) c.next();
    }
    return m;
}
```

Kuva 2. Kaksi koodiesimerkkiä, jotka tekevät saman asian SQL-tietokannalla ja MongoDB-tietokannalla. [4.]

Koska MongoDB-tietokannan rakenne jäljittelee JSON-tiedostomuotoa, siitä on helppo saada JavaScript- tai TypeScript-objekteja rakennettua suoraan.

Yksi iso haitta on se, ettei ole järjestävää itsensä lisäävää ID-kenttää. Sen sijaan MongoDB tarjoaa Object ID:n, jonka alku on kentän luontiaika sekunnilleen. Toinen osa pitää objektit uniikkeina, mutta tämä toinen osa on täysin satunnaisesti generoitu eikä sitä voi käyttää saadakseen objektit luontijärjestykseen. Etuna tästä tulee kuitenkin se, että objektin luontiaika on tiedossa melko tarkasti.

```
[
    {
        "_id": "597880e2f36d286610575f25",
        "day_num": 1,
        "text_main": "나",
        "text_alt": "na",
        "meaning": "me",
        "related_words": [
            {
                "_id": "5978810bf36d286610575f30",
                "text_main": "저",
                "text_alt": "jeo",
                "meaning": "I (polite)"
            },
            {
                "_id": "59788119f36d286610575f34",
                "text_main": "제",
                "text_alt": "je",
                "meaning": "I, my (polite)"
            }
        ],
        "updated": "1502879162585"
    }
]
```

Esimerkkikoodi 1. Esimerkki MongoDB-tietokannasta ja JSON-tietorakenteesta

3.2 Express.js

Express.js on TJ Holowaychuk -nimisen kehittäjän julkaisema minimaalinen palvelinkehys. Ensimmäinen versio julkaistu vuonna 2010 ja nykyään se on de facto -standardi palvelinkehys Node.js-ajoympäristölle. Sillä voi tehdä verkkosovelluksia ja API-ohjelmointirajapintoja. MEAN-pinossa Express on ohjelmiston palvelinpuoli ja välittää tiedot MongoDB-tietokannan ja Angular-näkymän välillä. Express on JavaScript-kielellä kirjoitettu, mikä ei ole palvelinpuolelle kovin yleistä, sillä JavaScript on tyypillisesti käytössä selainpuolella. [5.]

3.3 Angular

3.3.1 Alkuperä ja AngularJS

Angular on Googlen kehittämä JavaScript-kehys, joka auttaa kehittäjiä toteuttamaan nykypäivän monimutkaisia vaatimuksia verkkosovelluksille helpommin. Angular oli aluksi Googlen kehittäjän sivuprojekti, mutta pian sen suuret hyödyt tulivat muille selväksi, ja tiimi julkaisi projektista 1.0-version vuonna 2011 avoimena lähdekoodina. [6.]

Angularin suosio kasvoi nopeasti julkaisun jälkeen, mutta kun versio 2 julkaistiin, Angularia käyttävät kehittäjät eivät olleet iloisia, sillä versio 2 oli kirjoitettu kokonaan uudelleen, eikä mikään ollut yhteensopivaa version 1 ja 2 välillä. Tämän takia nykyään on kaksi aktiivisesti tuettua Angular-versiota: Angular 1.x, joka nykyään tunnetaan paremmin nimellä AngularJS, ja Angular 2+, jota usein sanotaan yksinkertaisesti Angulariksi.

Koska Angularin ensimmäisen version suosio ehti kasvaa niin nopeasti, AngularJS on vieläkin suositumpi versio Angularista ja useimmiten, kun puhutaan MEAN-ohjelmistopinosta. Siinä se A tarkoittaa AngularJS:ää.

Angular 2 toi yhtenä merkittävänä muutoksena kielen vaihdon. Siirryttiin JavaScriptistä käyttämään TypeScriptiä. Angular 2 -päivityksessä tehtiin myös koodikannasta modernimpaa. Nämä muutokset tekivät koodista helpommin luettavaa ja paransivat käytettävyyttä. Angular 2 -päivitys oli luotu tarkkaan kuunnellen ensimmäisen version kehittäjäyhteisön palautetta. Yksi päätavoite oli parantaa suorituskykyä ja pitää silmällä mobiililaitteiden vaatimuksia. [7.]

Kirjoitushetkellä Angularin uusin vakaa versio on numero 4 ja sitä käytetään myös tämän raportin projektissa.

3.3.2 Angular CLI

Angular CLI on komentoriviliittymä, joka asennetaan erikseen Angularista ja tukee Angular-projektin kehitystä.

Angular CLI:n yleisin käyttötarkoitus on projektin generoiminen ja aloittaminen. Se on Angularin virallisissa ohjeissa suositeltu tapa aloittaa projekti, eikä edellisiä tapoja virallisesti tueta enää.

Mutta Angular CLI tarjoaa muutakin kuin vaan projektin generoinnin ja alkuun pääsemisen. Se voi myös generoida sovelluksen osia kuten palveluita ja komponentteja. Angular CLI:n avulla voi myös kääntää projektisi JavaScriptiin, ja tässä muodossa sitten sovellukset julkaistaan tuotantoon. [8.]

3.3.3 Angularin arkkitehtuuri

Angural koostuu monesta kirjastosta. Jotkut niistä ovat pakollisia ja muut vapaaehtoisia. Angular-sovellukset ovat modulaarisia sen oman NgModule-modulaarisuusjärjestelmän avulla. Modulaarisuuden avulla sovelluksia voidaan erotella selvästi eri ominaisuuksiin, ja tämä helpottaa ominaisuuksien lisäämistä lisäämällä ulkopuolisia moduuleja. Modulaarisuuden avulla voidaan myös rajoittaa sovelluksen kokoa. Sillä voidaan jättää pois joitakin Angularin mukana tulevia kirjastoja.

Jokaisessa Angular-sovelluksessa on vähintään yksi NgModule-moduuli. Yleensä tämä on nimerry AppModule-nimiseksi ja sijaitsee tiedostossa app.module.ts. Tähän tiedostoon tullaan viittaamaan myöhemmin, sillä se on se, joka käynnistää sovelluksen. Siihen lisätään viittaukset muihin ominaisuuksiin ja komponentteihin, joita halutaan osaksi sovellusta.

Osana moduuleja ovat komponentit. Angular-komponentit tyypillisesti koostuvat HTML-, CSS- ja TypeScript-tiedostosta. Tosin HTML- ja CSS-osuuden voi myös joko kirjoittaa TypeScript-tiedostoon tai jättää pois. Komponentit hallitsevat sovelluksen näkymää ja Angular luo, päivittää ja tuhoaa komponentteja sen mukaan, kun käyttäjä navigoi ympäri sovellusta. [9.]

3.3.4 Angularin ominaisuuksia

Angular tarjoaa paljon erilaisia ominaisuuksia. Tässä mainitaan vain muutama niistä. Erityisesti keskitytään niihin, jotka olivat tärkeitä tämän raportin projektin kehitykseen.

Yksi Angularin hienoimmista ja ensimmäisenä huomattavista ominaisuuksista on, miten sulavasti sillä on mahdollista tehdä sivustoja, jotka eivät päivity kokonaan, kun klikataan linkkiä, vaan ne päivittävät vain tarpeellisen komponentin, joka sitten näyttää uutta tietoa.

Tyypitys on ominaisuus, joka tulee TypeScriptistä. Se on näin myös Angularin vahvuus. Se helpottaa kehitystä niin, että vääristä tyypeistä varoitetaan ja näin vältytään virheiltä. Toinen tällainen ominaisuus on rajapinnat (englanniksi interface) ja rajapintojen avulla voidaan määrittää objektien ominaisuuksien tyypit.

```
interface RelatedWord{
  _id: string,
  text_main: string,
  text_alt: string,
  meaning: string,
  audio_path: string,
  audio_time: string
}
interface Word{
  _id: string,
  day_num: number,
  text_main: string,
  text_alt: string,
  meaning: string,
  audio_path: string,
  audio_time: string,
  updated: string,
  related_words: RelatedWord[]
}
```

Esimerkkikoodi 2. Kaksi rajapintaa TypeScript-objekteille, kuten loppupäässä näkyy, Word-rajapinta sisältää RelatedWord-tyyppisen taulukon, jonka ominaisuuksien tyyppi on määritelty juuri ennen Word-rajapintaa omalla rajapinnallaan.

Direktiivit (englanniksi directive) Angularissa mahdollistavat muun muassa if- ja for-lauseiden lisäämisen suoraan HTML-koodiin. Tämä mahdollistaa aivan erilaisen verkkosovellusten kehityksen. HTML-tagissa oleva if-lause voi muuttujan perusteella määrittää, onko elementti näkyvissä vai piilossa, ja for-lause määrittää, montako kopiota elementistä on.

Direktiivien tukena datan sitominen (englanniksi data binding) ja interpolointi (englanniksi interpolation) antavat Angularissa kyvyn lisätä muuttujia HTML-koodiin. Esimerkkikoodissa 3 nähdään, kuinka näitä ominaisuuksia käytetään yhdessä käydäkseen läpi people-taulukon muuttujia.

```
<tr *ngFor="let person of people">
  <td>{{person.name}}</td>
</tr>
```

Esimerkkikoodi 3. HTML-koodi, joka sisältää ngFor-direktiivin, jossa luetaan person-objektin name-muuttujaa. Tämä luo saman määrän tr-tägejä, kuin person-taulukossa on alkioita.

Datan sidonnan tueksi on olemassa ngModel-direktiivi, joka auttaa kaksisuuntaista datansidontaa. Tällä tarkoitetaan sitä, että sovelluksen mallin lisäksi myös näkymä voi muokata suoraan sidottua dataa.

Esimerkiksi tietty TypeScript-muuttuja on sidottu kahteen eri HTML-elementtiin sivussa, ja molemmilla näistä elementeistä on suora yhteys muuttujaan. Jos yksi näistä

elementeistä on tekstikenttä, jota käyttäjä muokkaa. Tämä muokkaus vaikuttaa heti muuttujan sisältöön, ja näin myös toisen elementin sisältö muuttuu saman tien.

3.3.5 TypeScript

TypeScript on Microsoftin kehittämä parannus JavaScriptiin. Sen suunnitteli C#-kielen suunnittelija Anders Hejlsberg. TypeScript lisää tyypitystä muuttujiin ja tukee paremmin olio-ohjelmointia lisäämällä tukea luokille, rajapinoille, periytymiselle ja muuta. TypeScript on kuitenkin erittäin vahvasti kytkeytynyt JavaScriptiin, sillä se käännettään JavaScriptiksi. Näin kaikki selaimet, jotka tukevat JavaScriptiä, tukevat myös TypeScriptiä.

Kehityksen aikana voidaan käyttää JavaScript-kirjastoja tai jopa kirjoittaa suoraan JavaScript-koodia keskelle TypeScript-koodia. TypeScript antaa kehittäjille mahdollisuuden kehittää verkko-ohjelmistoja perinteisemmän ohjelmointikielen tapaan. Näin vältetään virheitä, jotka helposti tapahtuvat JavaScripin tyylisessä kielessä. Mutta se kuitenkin riippuu kehittäjästä, kuinka paljon hän käyttää hyväksi TypeScriptin etuja. [10.]

3.3.6 Vaihtoehtoinen työkalu Angularille - React

React on Facebookin kehittämä ja ylläpitämä JavaScript-kirjasto, joka on helposti verrattavissa Angulariin. Molemmat ovat tuoreita työkaluja responsiivisten sivujen kehitykseen ja jakavat paljon yhtenäisyyksiä. Eroina Reactin ja Angularin välillä on muun muassa se, että Reactissä käytetään JavaScriptiä, kun taas Angularissa on käytössä TypeScript. Lisäksi React on huomattavasti pienempi kooltaan, mutta sen takia se sisältää myös vähemmän ominaisuuksia vakiona. [11.]

3.4 Node.js

Node.js on verkkopalvelin, jonka päämääränä on mahdollistaa helposti kehitettävä, nopea ja skaalautuva verkko-ohjelmisto. Se on rakennettu Googlen Chromen JavaScript-moottorin päälle ja on avointa lähdekoodia. Node.js:n kehittäjä, Ryan Dahl julkaisi ensimmäisen version vuonna 2009 ja nykyään sitä käyttävät monet alan suurimmat yritykset, kuten muun muassa eBay, Microsoft, PayPal ja monet muut.

Node Package Manager

NPM tulee Node.js-asennuksen mukana, ja se on Node.js:n vakio paketinhallintasovellus. NPM mahdollistaa monien eri kirjastojen ja sovellusten asentamisen yhdellä komentorivillä. Koska Node.js:n moduulit vievät paljon tilaa, niitä ei suositella lisäävän versionhallintaan, mutta sinne lisätään ”package.json”-tiedosto, joka sisältää kaikkien moduulien nimet ja versiot. Näin NPM voi ”npm install” -komennolla hakea kaikki tarvittavat moduulit sovelluksen ajamiseen ja kehityksen jatkamiseen.

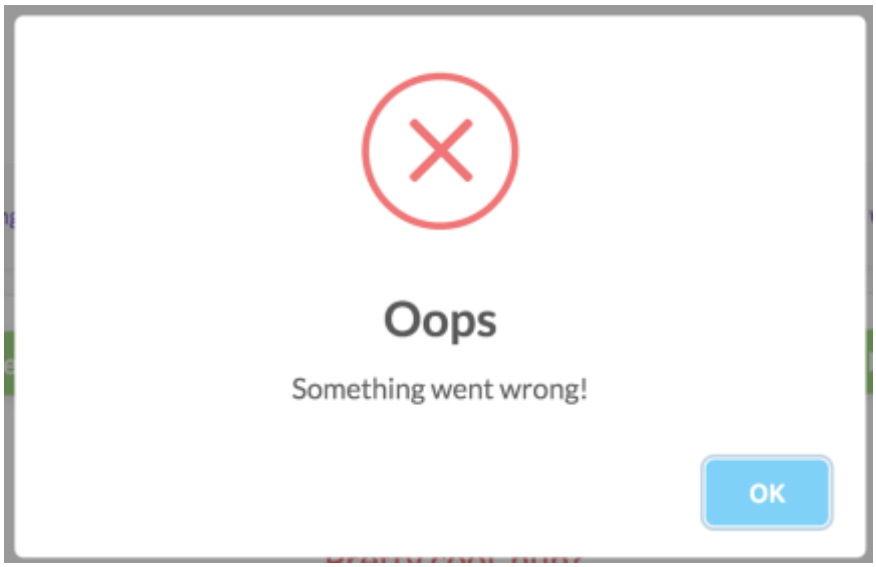
3.5 Ulkoiset lisäosat

Helpottaakseen kehitystä, kaksi JavaScript-kirjastoa, kaksi CSS-tiedostoa ja fontti, jotka eivät kuulu MEAN-ohjelmistopinoon, lisättiin projektiin. Nämä asentuivat helposti NPM:n avulla.

Sweet Alert

Sweet Alert on Tristan Edwards -nimisen kehittäjän luoma JavaScript-kirjasto, jonka avulla voi helposti tehdä kauniita ja muokattavia ponnahdusikkunaviestejä. Selaimet tarjoavat oman ponnahdusviestijärjestelmän, mutta ne eivät ole kovin käyttäjäystävällisiä tai viehättävän näköisiä. Alla on kaksi kuvaa, jotka vertailevat Chrome-selaimen alert-järjestelmää ja yksinkertaista Sweet Alert -viestiä.

Kuva 3. Alert-viesti Chrome-selaimessa.



Kuva 4. Yksinkertainen viesti Sweet Alert JavaScript -kirjastoa käyttäen.

Kuten kuvasta 3 ja kuvasta 4 näkyy, Sweet Alert -viesti on yhdessä yksinkertaisimmassaan muodossaan jo paljon viehättävämpi. Lisäksi jälkimmäisellä on käyttäjäystävällisempi käyttäytyminen selaimessa. Sen voi sulkea painamalla OK-nappia tai painamalla tummennettua taustaa, kun taas Chomen alert-viesti jumittaa sivua niin, ettei voi edes vierittää sivua tai tehdä mitään muuta kuin sulkea viestin joko painamalla OK-nappia tai vaihtamalla välilehteä.

Osoittaakseen, kuinka helppoa molempien viestien luonti on koodissa, alla on molempien kutsusta koodiesimerkki.

```
alert ( "Oops, something went wrong!" )
```

Esimerkkikoodi 4. Alert-viestin kutsu, joka sisältää halutun viestin lainausmerkeissä.

```
swal ( "Oops" ,  "Something went wrong!" ,  "error" )
```

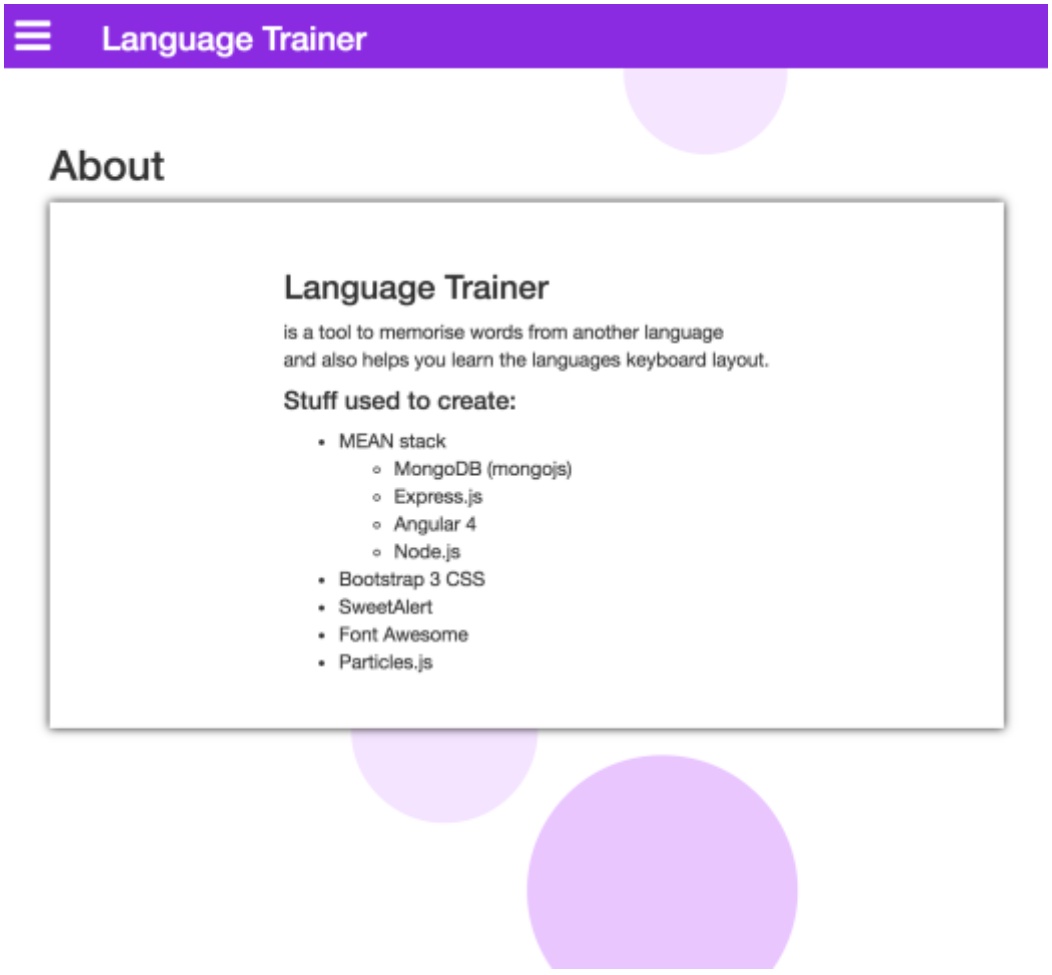
Esimerkkikoodi 5. Sweet Alert-viestin kutsu, jossa on kolme eri kohtaa. Ensimmäisissä lainausmerkeissä eroitetaan ”Oops”-sana muusta viestistä otsikoksi, toisessa kohdassa on normaali viesti ja lisäämällä kolmannen kohdan saamme viestiimme kuvassa 3 ilmenevän punaisen ”x” kuvion.

Kuten esimerkkikoodeissa näkyy, kuvassa 4 näkyvän viestin saa aikaan vain hieman monimutkaisemmalla koodirivillä. Poistamalla ”Oops”-sanan otsikoinnista, sekä punaisen ”x”-ikonin, koodirivin saisi yhtä yksinkertaiseksi kuin Alert-viestin kutsun, mutta viesti silti näyttäisi ja tuntuisi paremmalta. [12.]

Particles.js

Particles.js JavaScript -kirjaston avulla lisättiin liikkuvia partikkeleja sovelluksen taustalle ilman, että se hidasti sovellusta tai vaati käyttäjän tietokoneelta paljon lisätehoa sivun käyttämiseen. Partikkelijärjestelmällä saa aikaan monia erinäköisiä taustoja, ja se kaikki onnistuu määrittämällä asetukset yhdessä JSON-tiedostossa.

Helpottaakseen asetusten määrittämistä kirjaston kotisivulla on saatavilla työkalu, jolla voi määritellä näitä asetuksia hiirellä klikkaillen ja heti nähdä muutokset taustalla olevista partikkeleista. Tämän avulla tehtiin liikkuva partikkelitausta, jonka näkee kuvassa 5. [13.]



Kuva 5. Kuvakaappaus sovelluksen About-sivusta, jossa nähdään, kuinka taustalla olevat vaalean purppurat partikkelit eivät tunkeudu alueille, joissa on sisältöä, vaan liikkuvat pelkästään taustalla.

Font Awesome

Font Awesome tarjoaa satoja hyvännäköisiä, skaalautuvia vektori-ikoneja, jotka ovat helposti sivuun lisättäviä ja helposti CSS:llä muokattavia. Font Awesome oli alun perin suunniteltu Bootstrapia varten, ja nykyään se toimii hyvin kaikkien ohjelmistokehysten kanssa, sillä se on yksinkertaisesti vain fontti ja sitä tukevaa CSS-koodia. Font Awesome on saatavilla ilmaiseksi kaupalliseen käyttöön, mutta siitä on myös maksullinen paketti, joka tarjoaa enemmän ikoneja ja muita etuja. [14.]

Bootstrap 3 CSS

Bootstrapia kehittivät Mark Otto ja Jacob Thornton Twitterissä, ja he julkaisivat sen ilmaisen avoimen lähdekoodin projektina elokuussa 2011. Bootstrapistä on siitä lähtien tullut erittäin suosittu ohjelmistokehys verkkosovellusten selainpuolen kehittämiseen. Se tarjoaa monenlaisia HMTL- sekä CSS-malleja ja myös JavaScript-liitännäisiä. [15.]

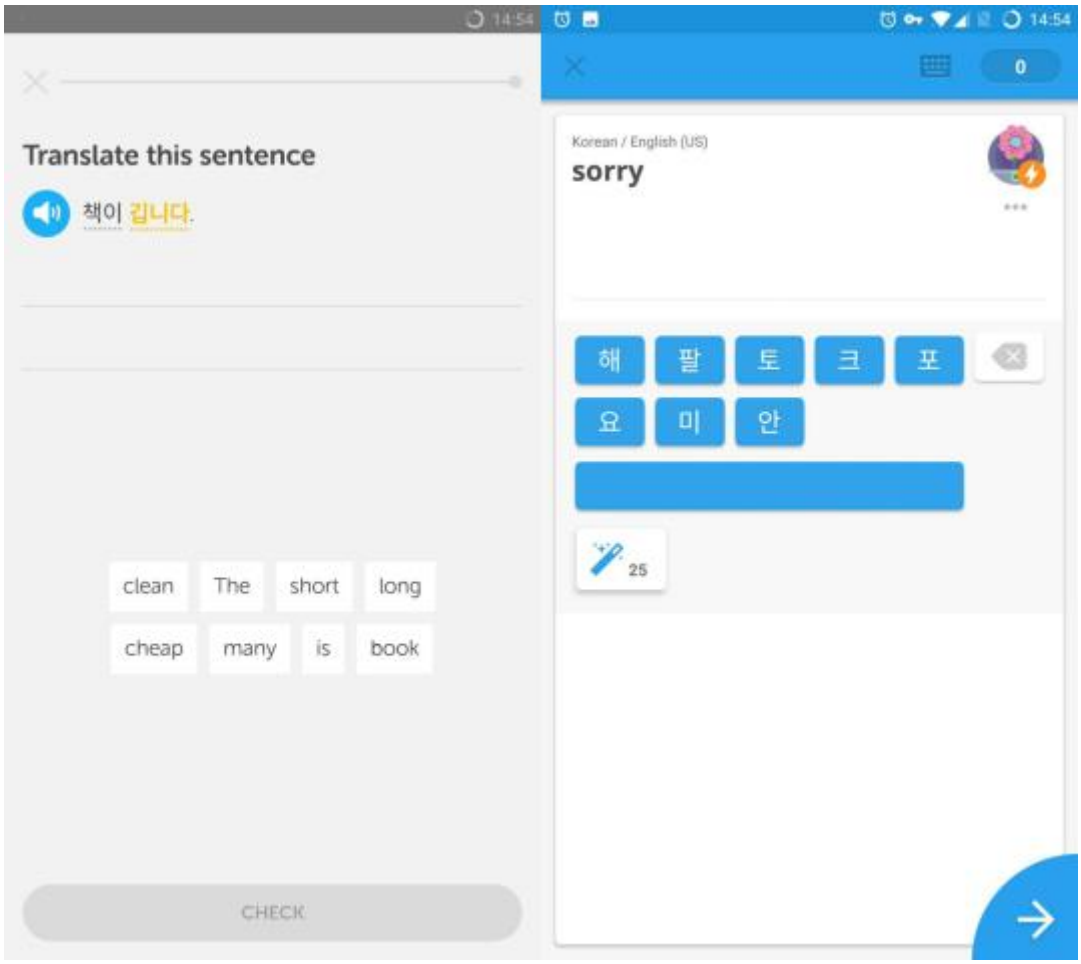
Tässä projektissa kuitenkin käytetään vain Bootstrapin CSS-osuutta saadakseen helposti yhtenevää tyyliä sovelluksen lomakkeisiin ja nappeihin.

4 Sovelluksen suunnittelu

4.1 Inspiraatio ja samantyyliset ohjelmistot

Kehitettävän ohjelmiston tarkoitus olisi tukea kielen oppimista, erityisesti erilaisen kirjoitusjärjestelmän, sekä sen näppäinasettelun oppimista näppäimistöllä. Sovelluksessa käyttäjä voisi lisätä tietokantaan omia sanoja, joita haluaa harjoitella ja sovellus pisteyttäisi käyttäjää suorituksen mukaan.

Tällaisia ohjelmistoja on tietenkin jo olemassa, ja ne ovat jopa erittäin suosittuja. Kaksi tällaista palvelua ovat Memrise ja Duolingo. Molemmat näistä ovat saaneet ”Editor’s choise” -merkinnän Googlen Play-kaupassa, sekä Memrise sai jopa ”Best App” -palkinnon vuonna 2017. [16; 17.]



Kuva 6. Kuvakaappaukset Duolingon (vasemmalla) ja Memrisen (oikealla) Android-versioista.

Molemmissa Memrise- sekä Duolingo-palveluissa on kuitenkin parantamisen varaa. Duolingo oli kehitetty eurooppalaisia kieliä silmällä pitäen, ja tämän projektin kehityksen alkuvaiheessa Duolingon verkkosovellusversio tuki vain kieliä, jotka käyttivät latinalaista tai kyrillistä kirjoitusjärjestelmää. Joten vaikka Duolingo tarjosi japanin kielen kurssin, se oli käytettävissä vain mobiiliversiossa.

Toinen Duolingon heikkous on, että se ei salli käyttäjän lisätä omaa materiaalia vaan heillä on vain yksi kurssi jokaiselle kielelle, mitä he tukevat. Toisin sanottuna käyttäjillä ei ole valinnanvaraa, jos he eivät pidä Duolingon tarjoamasta kurssista tietylle kielelle.

Memrise taas sallii käyttäjien luoda omat kurssit. Niinpä heillä on tarjolla monta kurssia, mistä valita jokaiselle kielelle sekä heidän verkkosovellusversio tukee eri kirjoitusjärjestelmien kieliä kuten korean ja japanin kieltä.

Mutta Memrisellä on omat heikkoutensa. Heidän verkkoversiossa on tehtävien vastaamiseen asetettu aikaraja, jota ei saa asetuksista pois päältä. Tämä on ongelmallista, jos valittu kielikurssi käyttää erilaista kirjoitusjärjestelmää, johon käyttäjä ei ole tottunut, koska vaikka käyttäjä tietäisi vastauksen, sen kirjoittamisessa voi mennä liian kauan.

Memrisen mobiiliversiossa ei ole aikarajaa, mutta tuo versio ei käytä puhelimen omaa näppäimistöä esimerkiksi korean kielen kurssissa, jonka käyttäjä voisi oman puhelimensa asetuksissa valita oikeaksi. Sen sijaan, kuten kuvassa 6 nähdään, Memrise tekee omat napit, jotka koostuvat kokonaisista tavuista, ja helpottaa vastaamista huomattavasti.

Työn tavoitteena oli kehittää ohjelmisto, joka ottaa mallia näiden palvelujen hyvistä puolista, eikä sisällä yllä mainittuja heikkouksia.

4.2 Suunnitellut ominaisuudet

Koska ohjelmistossa halutaan käyttää itse lisättyä dataa, heti ensimmäisenä piti saada käyttöön tietokannan hallintaominaisuus, sillä manuaalisesti suoraan tietokantaan datan lisääminen olisi hankalaa ja veisi aikaa. Tämä näkymä olisi erillään pääohjelmistosta.

Ohjelmiston pääominaisuutena oli tavoitteena saada sivu, jossa käyttäjälle annetaan sana, jonka käyttäjän pitää kääntää toiselle kielelle. Käyttäjän kirjoittamaa vastausta vertaillaan sitten oikeaan vastaukseen ja tuloksesta annetaan käyttäjälle suorituksen mukaisesti pisteytys, jonka jälkeen jatketaan seuraavaan sanaan.

Kääntösivun näkymässä tuli olla annetun sanan ja vastauskentän lisäksi muutama nappi. Ensimmäinen näistä olisi vihje nappi, joka tuo esille sanalle määritellyn vihjeen. Toinen olisi ääninappi, jota voi painaa vain, jos sanalle löytyy äänitiedosto, jossa se lausutaan. Kolmas olisi ohita-nappi, jolla jätetään nykyiseen sanaan vastaamatta ja siirrytään seuraavaan. Lopuksi neljäs nappi olisi vahvistusnappi, joka lähettää vastauksen.

Samassa näkymässä yllä mainittujen elementtien alla tuli olla lista aiemmista vastauksista ja niistä saaduista pisteistä. Klikkaamalla aiempaa käyttäjän kirjoittamaa vastausta tulisi tulla esiin käyttäjälle annetun sanan tiedot.

Pisteytyssäännöt ja vaikeusasteet

Suunnitelmana oli saada kolme vaikeusastetta: helppo, keskitaso ja vaikea. Valittu taso muuttaa näkymää hieman, saattaa rajoittaa, mitä tietoa on saatavilla ja vaikuttaa pisteytykseen.

Helppo vaikeustaso näyttää kaiken, mitä on saatavilla. Vihje on heti näkyvissä ja käyttäjä voi vapaasti katsoa listaa aiemmista vastauksista. Osittain oikeasta vastauksesta saa puolikkaan pisteen.

Keskitasossa vihje on vakiona piilotettu. Sen saa auki napilla, mutta tässä tapauksessa oikeasta vastauksesta vähennetään neljäsosa pisteestä. Aiempien vastausten lista on näkyvissä, mutta käyttäjä ei voi avata sitä nähdäkseen annetun sanan. Hän näkee vain kirjoittamansa vastauksen ja siitä saadun pisteen. Vastauksen pitää olla kirjoitettu oikein saadakseen pisteen. Saadut pisteet kerrotaan kahdella.

Vaikealla tasolla käyttäjä ei saa käyttää vihjettä. Lista vanhoista vastauksista on piilossa ja on aikaraja vastata kysymykseen. Vastauksen pitää olla kirjoitettu oikein saadakseen pisteen ja pisteet kerrotaan kolmella.

5 Sovelluksen kehitys

5.1 Projektin aloitus ja palvelinpuoli

Heti projektin alussa tiedettiin, että olisi kannattavaa jakaa projekti kolmeen osaan, joita olisivat tietokanta, palvelin ja selaimessa avattava sivusto. Koska projektin haluttiin olevan helposti saatavilla eri tietokoneille, tietokanta laitettiin pilveen mLab-nimisen palvelun avulla ja molemmille sekä palvelinpuolelle että selainpuolelle projektista tehtiin omaa haaraa git-versionhallintaan.

MongoDB ja Express.js

Palvelinpuolella Express.js teki palvelimen peruskonfiguroinnista vaivatonta, ja se saatiin toiminnalliseksi nopeasti vain noin 50 rivin server.js-tiedoston kirjoittamisella, joka meni suurimmaksi osaksi vain ohjeita seuraamalla. Lisäsin siihen vain minulle sopivan portti-numeron, sallitun osoitteen, joka voi tehdä kutsun palvelimelle sekä sallitut operaatiot.

Mutta palvelinpuolen hommat eivät loppuneet tähän, sillä suunnitelmissa oli kehittää oma datanhallintasivu, jossa lisätään, muokataan ja poistetaan dataa. Tämä osuus jaettiin eri tiedostoon, joka hoiti kaikki yhteydet meidän MongoDB-tietokantaan.

Aluksi kirjoitettiin funktiot, jotka vain hakivat kaiken datan tai yhden alkion ID:n mukaan sitten myös alkioiden lisääminen ja sille Object ID:n generointi ja myös ID:n mukaan tietyn alkion muokkaaminen ja poisto. Aluksi nämä kaikki jäivät vielä melko yksinkertaiseksi, mutta näihin palattiin myöhemmin, kun selainpuolen kehitys eteni ja tiedettiin tarkemmin, millaista tietoa sieltä tarkalleen lähetetään ja myös varmistetaan, että sieltä ei pääse läpi mitään vaarallista tietokantaan. Mihinkään selainpuolelta tulevaan dataan ei luotettu, vaikka siellä puolella tehtiinkin omat tarkistukset, koska käyttäjät voivat kehittäjätyökaluilla päästä muokkaamaan selainpuolen koodia.

5.2 Selainpuolen kehitys

Heti aluksi asennamme Angular CLI –komentoriviliittymän. Se onnistuu nopeasti NPM:n avulla (esimerkkikoodi 6).

```
npm install –g @angular/cli
```

Esimerkkikoodi 6. NPM-komentorivi Angular CLI:n asennukseen

Angular-kehityksen aloittamisesta on tullut erittäin helppoa Angular CLI -komentoriviliittymän avulla, sillä se generoi toimivan projektin vain yhdellä komentorivillä ja kahden seuraavan komentorivin jälkeen generoitu projekti on jo päällä ja auki selaimessa (esimerkkikoodi 7).

```
ng new language-trainer
cd language-trainer
ng serve –open
```

Esimerkkikoodi 7. Uuden Angular-projektin generointi ensimmäisellä rivillä, generoidun kansioon navigoiminen toisella rivillä, käynnistäminen ”ng serve”-komennolla ja selaimeen avaaminen ”--open”-loppuosalla.

Esimerkkikoodissa 7 ”ng serve –open”-komento avaa Angular-sovelluksen selaimeen, mutta tämä ei tallenna käännettyä versiota sovelluksesta. Tämä komento on kehittämistä varten, ja sovellus päivittyy selaimessa aina, kun kehitystiedostoihin tallennetaan muutos. [11.]

Datapalvelu

Angular CLI:n avulla generoidaan Datapalvelu (esimerkkikoodi 8), joka tulee hoitamaan kaikki yhteydenotot Express.js-palvelimeen.

```
ng generate service services/data
```

Esimerkkikoodi 8. Komentorivi joka pyytää Angular CLI:n generoimaan data-nimisen palvelun services-kansioon.

Koska kaikki datan varmistaminen ja oikea yhteys MongoDB-tietokantaan hoidetaan palvelinpuolella Express.js:n avulla, datapalvelun koodi jää lyhyeksi. Siinä määritellään URL-osoite Express.js-palvelimen yhteydenottoa varten ja tehdään kaikki samat funktiot kuin palvelinpuolella eli kaikkien alkioiden haku (esimerkkikoodi 9), yhden alkion päivittäminen, poistaminen, hakeminen ID:n mukaan sekä uuden alkion lisääminen.

```
getWords(){
    return this.http.get( this.url + '/words').map(res => res.json());
 }
```

Esimerkkikoodi 9. Data-palvelu pyytää Express.js-palvelimelta kaikki sanat ja lähettää tiedon eteenpäin JSON-muodossa.

Esimerkkikoodissa 9 nähdään map-funktion kutsu, jonka sisällä on nuolifunktioilmaisu. Tämä nuolifunktio on vain erilainen tapa ilmaista funktio, jossa annetaan parametrina funktiolle ”res” ja funktio palauttaa ”res.json()”.

App-komponentti

Osana projektin generointia luotiin myös kehys projektin ensimmäiseen komponenttiin. Tämä komponentti on juurikomponentti, joka ajetaan aina ensimmäisenä ja jonka sisälle myöhemmin ohjataan muut komponentit näkyviin. App-komponentti tukee muita komponentteja niin, että siinä on otsikkopalkki, jossa on menu-nappi. Menu-nappi tuo esille valikon, jossa voidaan navigoida alikomponenttien välillä. Lisäksi App-komponenttiin lisättiin div-tagi, joka kattaa koko sovelluksen taustan ja siinä näytetään particles.js-kirjaston partikkelijärjestelmän tuottamat, liikkuvat partikkelit.

Data Management -komponentti

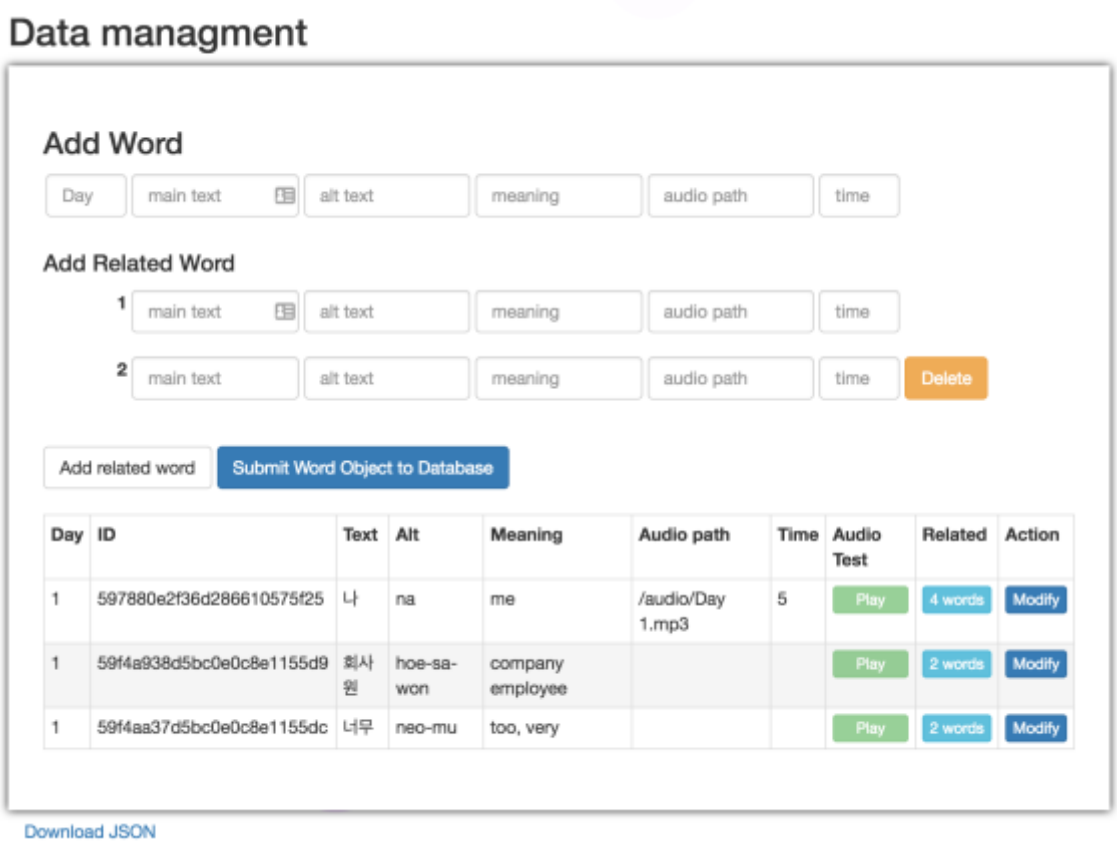
Data Management-komponentin oli suunniteltu olevan aputyökalu datan hallitsemiseen, ja siitä tuli iso osa projektia. Koska tällaisessa sovelluksessa on erittäin tärkeää pystyä helposti lisäämään uutta dataa ja helposti korjaamaan vanhaa dataa, päädyttiin panostamaan datan hallintakomponenttiin vähintäänkin sen verran, että siinä on helppo tutkia olemassa olevaa dataa ja helppo lisätä uusia objekteja.

Komponentin kehitys alkoi generoimalla komponentin runko Angular CLI:n avulla komentorivillä (esimerkkikoodi 10), joka myös automaattisesti lisää komponentin app.module.ts-tiedostoon, jolloin komponentti tulee käyttöön osana sovellusta.

```
ng generate component components/data-management
```

Esimerkkikoodi 10. Komponentin generointi Angular CLI:n avulla.

Kuten nimistä voi jo päätellä, Data Management -komponentti ja datapalvelu ovat jatkuvassa yhteydessä toisiinsa. Heti Data Management -komponentin avaamisessa komponentti hakee uusimmat tiedot datapalvelulta. Se tarvitsee näitä näyttääkseen täsmällistä tietoa taulukossaan, joka näyttää nykyisen tilanteen tietokannasta. Tämän takia joka muutoksen jälkeen komponentti hakee uudestaan tiedot, jotta käyttäjä näkee, että muutokset ovat tallentuneet ja missä kunnossa tietokanta on.



Kuva 7. Data management -komponentin perusnäkymä. Näkymän ylemmässä puoliskossa pystyy lisäämään sanaobjekteja tietokantaan ja alemmassa puoliskossa näkyy taulukko tietokannassa olevista objekteista.

Data Management -komponentin näkymän (kuva 7) luomisessa käytettiin paljon direktiivejä ja datan sidontaa. Kuten kaikki näkyvät lomakkeet on sidottu TypeScript-muuttujiin, jos jokin muuttuja on sidottu kahteen eri paikkaan, se muuttuu automaattisesti ja molemmissa paikoissa, jos sitä muokataan yhdessä. Näkymän related word rivit lisääntyvät heti, kun Add related word -nappia painetaan, joka lisää TypeScriptin puolella related word -taulukkoon uuden alkion. Vastakkainen tapahtuu, kun painaa related word -listan viimeisen rivin oikeallapuolella olevaa Delete-nappia. Delete-nappi pysyy aina viimeisellä rivillä käyttäen ngIf-direktiiviä (esimerkkokoodi 11).

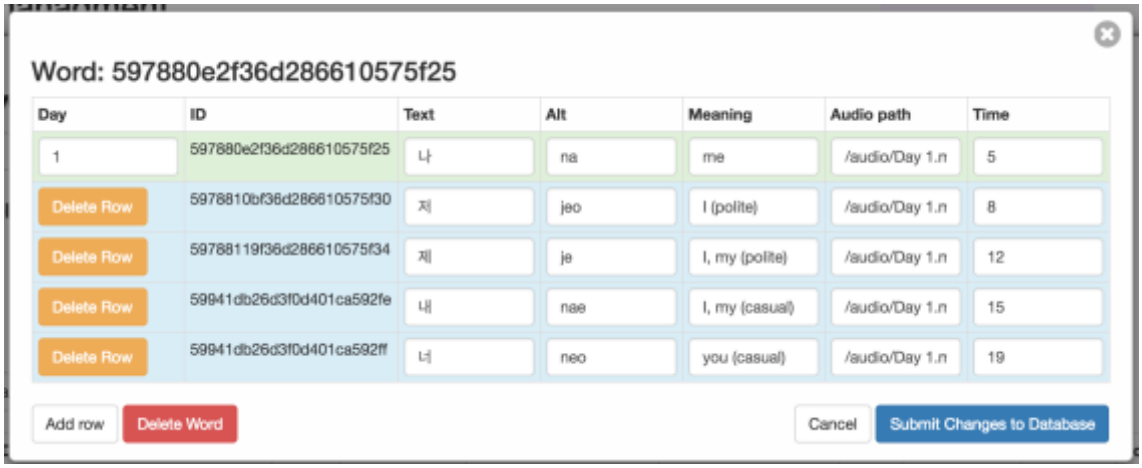
```
<input *ngIf="last" type="button" (click)="btnRemoveRelatedRow()"
value="Delete" class="btn btn-warning">
```

Esimerkkikoodi 11. Delete-napin HTML-koodi. ngIf-direktiivi määrittää, että nappi on näkyvissä vain kun last-muuttuja on true-arvoinen.

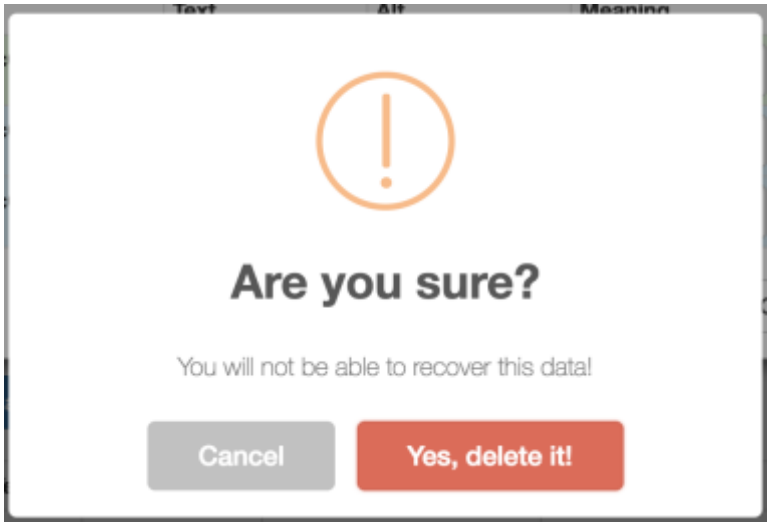
Samaa tekniikkaa käytettiin kuvan 7 alemmassa osassa näkyvän taulukon luontiin. For-lause-direktiivissä käydään läpi, montako sanaa taulukossa on, ja siitä tehdään sama

määrä rivejä. Komponentin loppuun tehtiin myös nappi, joka tallentaa koko tietokannan JSON-tiedostoon ja lataa sen käyttäjälle selaimen kautta.

Klikkaamalla taulukossa olevia Modify-nappeja avautuu ikkuna (kuva 8), jossa nähdään valitun sanan related word -aliobjektit, eli liittyvät sanat. Tässä näkymässä voi muokata tai poistaa objektin ja sen aliobjekteja tai lisätä aliobjekteja.



Kuva 8. Modify-napista ilmestyvä ikkuna.

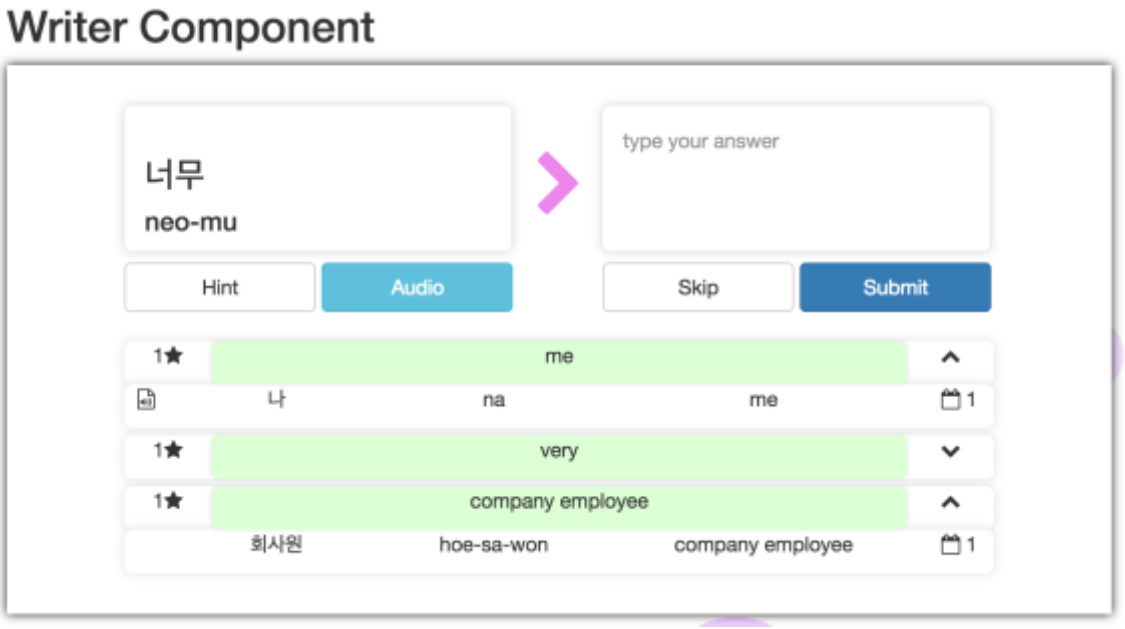


Kuva 9. Kuva 8:ssa näkyvää Delete word -nappia painamalla tulee esiin Sweet Alert -varmistuskysely.

Writer-komponentti

Writer-komponentti, joka on sovelluksen pääkäyttönäkymä, rakennettiin käyttämällä monia samoja tekniikoita kuin Data management -komponentissa. Vanhojen vastausten lista tehdään ngFor-direktiivillä, ja näkymä muuttuu vaikeusasteen valinnan mukaan, jota tarkkaillaan muutamilla ngIf-direktiiveillä.

Kuten myös Data management-komponentissa, Writer-komponentin vastauskenttä on sidottu TypeScript-muuttujaan, ja kun Submit-vahvistusnappia tai näppäimistön enter-painiketta painetaan, jos tämä kenttä ei ole tyhjä, käyttäjän vastausta vertaillaan haluttuun oikeaan vastaukseen ja sitten pisteytetään, ja tulos lisätään alla olevaan listaan (kuva 10).



Kuva 10. Kuvakaappaus Writer-komponentista leveässä muodossa. Vasemmassa yläkulmassa otsikon alla näkyy käyttäjälle annettu sana käännettäväksi ja sanan alla on paljastettu vihje teksti. Oikeassa yläkulmassa on vastauskenttä. Seuraavalla rivillä ovat vihje-, ääni-, ohitus- ja vahvistus-napit. Siitä alaspäin on lista edellisistä vastauksista, ensimmäinen ja kolmas alkio ovat klikattu auki nähdäkseen käyttäjälle annettu sana ja sen tiedot. Eroina nähdään, että kolmannelle ei ole määritelty ääni tiedostoa. Niinpä rivin alussa oleva, klikattava äänitiedostoikoni puuttuu.

Writer-komponentille tehtiin myös kapeampi mobiilinäkymä. Tämä onnistui kokonaan CSS-koodilla, ja näkymä muuttuu kuvassa 11 näkyvään muotoon, kun huomataan selaimen ikkunan olevan liian kapea normaaliin leveään näkymään.

Kuva 11. Kuvakaappaus Writer-komponentin kapeasta näkymästä, joka on tarkoitettu mobiililaitteille

Muut komponentit

Data Management- ja Writer-komponenttien lisäksi lisättiin pari yksinkertaisempaa komponenttia. Front- ja About-komponentit eivät sisällä mitään erikoisempaa toiminnallisuutta, vaan ovat vain HTML-asettelua, jonka sekaan on kirjoitettu tekstiä ja CSS-tyylisivukoodia parantaakseen ulkonäköä.

Front-komponentti on se komponentti, joka näkyy ensimmäisenä. Siihen kirjoitettiin lyhyt tervehdysteksti käyttäjille sekä, ohje miten päästä muihin, sovellukselle olennaisempiin komponentteihin.

About-komponentin kohdalla kirjoitettiin hieman, mikä on sovelluksen tarkoitus ja mitä työkaluja sen kehittämiseen käytettiin. About-komponentista voi nähdä kuvakaappauksen raportissa aiemmin Particles.js-kirjaston yhteydessä näytetystä kuvasta 5.

Routing

Angularin yksi suurimpia ominaisuuksia on vain osittainen sivun päivitys. Saadakseen tämän aikaan pitää konfiguroida reitit (englanniksi routes). Tämän voi hoitaa app.module.ts-tiedostossa, siihen lisätään aluksi import-lause tarvittaville paketeille (esimerkkikoodi 12).

```
import { RouterModule, Routes } from '@angular/router';
```

Esimerkkikoodi 12. Import-lause, jolla tuodaan RouterModule- ja Routes-paketit sovelluksen käytettäväksi.

Tämän jälkeen luodaan pysyvä muuttuja, jossa määritellään polku jokaiselle meidän komponentille.

```
const appRoutes: Routes = [
  {path:'', component: FrontComponent},
  {path:'writer', component: WriterComponent},
  {path:'about', component:AboutComponent},
  {path:'data', component:DataManagementComponent}
];
```

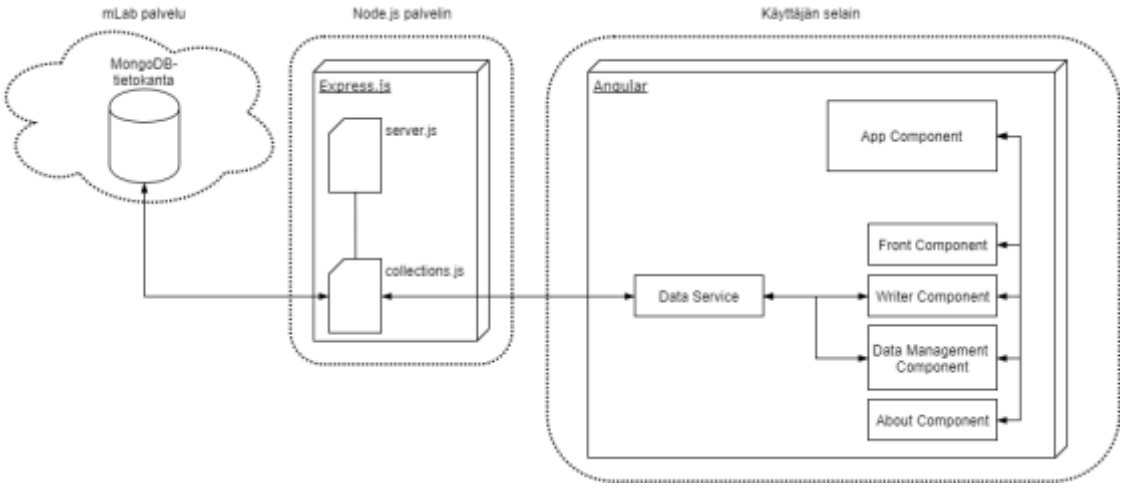
Esimerkkikoodi 13. Routes-tyyppinen appRoutes-muuttuja sisältää viittauksen sovelluksen komponentteihin ja niille määrätyn polun.

Esimerkkikoodissa 13 nähdään, että FrontComponent-komponentille on annettu tyhjä polku. Tämä tekee siitä sovelluksen vakiokomponentin, joka ohjataan esille ensimmäisenä.

Kun reitit on konfiguroitu, pitää lisätä HTML-koodiin ”router-outlet”-tagi, joka määrittelee, mihin komponenttien näkymä ohjataan. Tässä tapauksessa se lisätään ”app.component”-komponenttiin, joka on jokaisen Angular-sovelluksen vakiokomponentti.

Sovelluksen rakenne visualisoituna

Kuvassa 12 nähdään vasemmalla puolella, että MongoDB-tietokanta on pilvessä mLab-palvelun palvelimilla.



Kuva 12. Kehitetyn sovelluksen rakenne yleisesti.

Keskellä nähdään Express.js-osuus sovelluksesta, joka on saatavilla Node.js-palvelimen avulla, ja sovelluksen oleellisimmat osat on jaettu server.js- ja collections.js-tiedostojen välillä. Collections.js-tiedostoon on eroteltu kaikki tietokantaan ja selainpuoleen yhteyksissä olevat funktiot.

Kuvassa oikealla puolella nähdään Angular-osuus, joka suoritetaan käyttäjän selaimessa.

Kun katsomme Angular-osuuden sisään, näemme App-komponentin muiden komponenttien yläpuolella. Tämä komponentti on se, joka aina Angular-sovelluksen käynnistyessä ajetaan ensimmäisenä, ja jonka sisälle on tässä tapauksessa konfiguroitu muut komponentit ilmestymään.

Angular-osuudessa nähdään myös, että siihen ollaan ulkopuolelta yhteydessä vain datapalvelun kautta, ja tähän palveluun pääsevät käsiksi vain Writer- ja Data Management -komponentit. Nämä komponentit tarvitsevat datapalvelun hakemia tietoja toimiakseen, kun taas Front- ja About-komponentit ovat vain valmiiksi määriteltyä tekstiä.

6 Jatkosuunnitelmat, johtopäätökset ja yhteenveto

6.1 Kehityksen jatkosuunnitelmat

Tulevaisuuden suunnitelmina tämän sovelluksen suhteen on aluksi hioa nykyiset ominaisuudet erinomaiseen kuntoon ja myöhemmin lisätä ominaisuuksia, lisää tapoja oppia ja harjoitella kieliä.

Yhtenä ideana tulevista ominaisuuksista olisi komponentti, jossa esitetään kaksi riviä sanoja, yksi rivi opittavalla kielellä ja toinen tunnetulla kielellä. Käyttäjän tehtävänä olisi hiirellä yhdistää sanat erikieliseen pariinsa. Toinen idea olisi sellainen komponentti, jossa käytetään sanoja, joille on määritelty äänitiedosto. Käyttäjälle toistetaan äänitiedosto ja käyttäjän pitää joko valita lyhyestä listasta lausuttu sana, tai kirjoittaa se itse vastauskenttään.

Lisäksi, jotta voisin julkaista tämän joskus muillekin ihmisille, haluan lisätä tunnusjärjestelmän, jossa eri käyttäjillä voi olla eri tietokanta käytössä ja heidän pisteitään voisi verrata muiden pisteisiin.

Kaukaisempana ideana kehityksen jatkamiseksi olisi tehdä mobiilisovellusversio käyttämällä työkalua nimeltä Electron, jonka avulla voi kehittää mobiilisovelluksia samankaltaisessa ympäristössä kuin Angular.

6.2 Johtopäätökset

Projektin alussa Angular-kehitys tuntui aivan niin uudelta ja erilaiselta, että projektin kehitys tuntui vaikealta ja hitaalta ja siinä oli paljon uutta opittavaa. Angularin oppiminen ei ole helppoa, jos on tottunut muihin verkkosovelluskehitysympäristöihin, sillä se on monella tavalla erittäin erilainen. Mutta se on sen arvoista, koska se, mitä Angular tarjoaa, on erikoista ja mahdollistaa verkkosovellusten kehityksen aivan erilaisella ajattelutavalla.

Kehityksen aikana tuli kyllä erittäin selväksi, että AngularJS on se suositumpi versio verrattuna Angulariin. Jos nyt heti haluaa jommallakummalla saada tuotantoon menevää ohjelmistoa aikaan, ja oppiminen alkaisi alusta, luulen, että suosittelisin AngularJS:ää

enemmän. Angular on ehkä tulevaisuutta, mutta AngularJS on nykypäivää ja sille löytyy enemmän tukea, sillä siinä esiintyviin ongelmiin löytyy enemmän vastauksia ja esimerkkejä muilta kehittäjiltä.

MongoDB taas tarjosi alussa aivan erilaisen tuntuman. Se tuntui helpolta, koska se sopi niin hyvin yhteen JavaScriptin ja TypeScriptin kanssa ja koska tietokannan rakenne ei ollut läheskään niin hankala kuin perinteisemmät SQL-tietokannat. Huomasin myös, ettei järjestettävä itsensä iteroiva ID-kenttä, jonka SQL-tietokannat tarjoavat, ei ollutkaan niin olennainen ja pakollinen osa tietokantaa.

Express.js oli juuri sitä, mitä paperilla luvattiin: minimaalinen, nopea JavaScript-palvelinkehys. Alkukonfiguraation jälkeen siitä ei tarvinnut huolehtia. Kun jossain vaiheessa piti tehdä muutoksia, se onnistui helposti.

Node.js hoiti hommansa niin kuin pitikin. Sen kanssa ei joutunut mitään säätämään: sen vain laittoi päälle ja se toimi.

Nämä ohjelmistot toimivat mahtavasti yhdessä, ja jos jokin kokenut verkkosovelluskehittäjä haluaa kokeilla jotainkin hieman uutta, tätä pinoa minä suosittelisin ja tulen itsekin käyttämään jatkossa.

6.3 Yhteenveto

Insinöörityönäni kehitin kielioppimista tukevaa ohjelmistoa, jossa käyttäjät voivat lisätä omaa dataa, harjoitella erilaisen kielen kirjoittamista ja erilaisen kirjoitusjärjestelmän näppäinasettelua. Sain aikaan vahvan perustan sovellukseen, jota haluan parantaa omaan käyttööni ja myöhemmin julkaista muidenkin käyttöön.

Siinä samalla ohjelmistoa kehittäessä opin myös minulle täysin uusia teknologioita kuten kaikki MEAN-ohjelmistopinon jäsenet. Opin miten ne toimivat yksittäin ja yhdessä, opin niiden heikkouksia ja vahvuuksia sekä opin uudenlaista verkkosovelluskehitystä Angularin kanssa.

Projektia tehdessäni hyödynsin myös MEAN-ohjelmistopinon ulkopuolisia teknologioita. Nämä olivat pääasiassa sovelluksen ulkonäön parannusta varten, ja ne samalla

paransivat sovelluksen käyttäjäystävällisyyttä. Ulkoisten kirjastojen käyttäminen osoittautui tärkeäksi Angularin oppimisen yhteydessä, sillä siinä tuli selväksi, miten helposti JavaScript-kirjastot sopivat yhteen TypeScript-kielen kanssa ja miten niitä pitää lisätä Angular-ympäristön saamiseksi käyttöön.

Projektia tehdessä opin myös lisää tietokannan suunnittelusta, sillä siitä ei ollut vielä paljon kokemusta ohjattujen ja valvottujen koulutusympäristöjen ulkopuolella. Tämän takia kehityksen aikana tietokannan rakenne muuttui parikin kertaa, kun tuli keksittyä parempia tapoja toteuttaa se.

Työn suurimpana haasteena oli Angularin kanssa aloittaminen. Se tuntui niin erilaiselta ja tuntemattomalta aluksi, että alkuun pääseminen ja siitä sujuvaan nopeaan kehittämiseen siirtyminen oli haaste. Mutta tunnen, että opin jotain täysin uutta, ja samalla myös opin, miten lähestyä jotain minulle tuntematonta paremmin.

Lähteet

1 Techopedia, Verkkodokumentti, Software Stack https://www.techopedia.com/definition/27268/software-stack Luettu: lokakuu, 2017.

2 Carey Wodehouse, Verkkodokumentti, Choosing the Right Software Stack https://www.upwork.com/hiring/development/choosing-the-right-software-stack-for-your-website/ Luettu: lokakuu, 2017.

3 Codewave Technologies, Verkkodokumentti, Why I recommend MEAN stack for scalability https://codewave.com/insights/praveen-on-building-scalable-apps-with-mean-stack/ Luettu: marraskuu, 2017.

4 Buzz Moschetti, 2014, Verkkodokumentti, MongoDB vs SQL: Day 1-2 https://www.mongodb.com/blog/post/mongodb-vs-sql-day-1-2 Luettu: lokakuu, 2017.

5 Express.js, Verkkodokumentti, Express https://expressjs.com/ Luettu: lokakuu 2017.

6 TJ VanToll, 2017, Verkkodokumentti, What is Angular https://developer.telerik.com/topics/web-development/what-is-angular/ Luettu: lokakuu 2017.

7 Rangle.io, Verkkodokumentti, Why Angular? https://angular-2-training-book.rangle.io/handout/why_angular_2.html Luettu: lokakuu 2017.

8 Google, 2016 Verkkodokumentti, Angular CLI https://cli.angular.io/ Luettu: lokakuu 2017.

9 Angular, Verkkodokumentti, Architecture Overview https://angular.io/guide/architecture Luettu: lokakuu 2017.

10 Tutorials Point, Verkkodokumentti, TypeScript - Overview https://www.tutorialspoint.com/typescript/typescript_overview.htm Luettu: lokakuu 2017.

11 Facebook, Verkkodokumentti, React https://reactjs.org/ Luettu: lokakuu, 2017.

12 Tristan Edwards, Verkkodokumentti, Sweet Alert https://sweetalert.js.org/ Luettu: lokakuu 2017.

13 Vincent Garreau, Verkkodokumentti, particles.js - A lightweight JavaScript library for creating particles. https://github.com/VincentGarreau/particles.js/ Luettu: lokakuu 2017.

14 Font Awesome, Verkkodokumentti, Font Awesome - The iconic font and CSS toolkit, http://fontawesome.io/ Luettu: lokakuu 2017.

15 W3 schools, Verkkodokumentti, Bootstrap Get Started https://www.w3schools.com/bootstrap/bootstrap_get_started.asp Luettu: lokakuu 2017.

16 Google Play, Verkkodokumentti, Memrise, https://play.google.com/store/apps/details?id=com.memrise.android.memrisecompanion Luettu: lokakuu 2017.

17 Google Play, Verkkodokumentti, Duolingo https://play.google.com/store/apps/details?id=com.duolingo Luettu: lokakuu 2017.