Graafisen datankeruutyökalun toteutus ja testaus

Angular & .Net

LAB-ammattikorkeakoulu
Digitradenomi (AMK), Liiketalous
2020

Tiivistelmä

<table>
<tr><td rowspan="2">Tekijä(t)
Karjalainen Antti</td><td>Julkaisun laji
Opinnäytetyö, AMK</td><td>Valmistumisaika
2020</td></tr>
<tr><td>Sivumäärä
29</td><td></td></tr>
<tr><td colspan="3">Työn nimi
Graafisen datankeruutyökalun toteutus ja testaus
Angular & .Net</td></tr>
<tr><td colspan="3">Tutkinto
Tradenomi(AMK)</td></tr>
<tr><td colspan="3">Toimeksiantajan nimi, titteli ja organisaatio
Lappeenrannan-Lahden teknillinen Yliopisto / LUT Business Management</td></tr>
</table>

Tiivistelmä

Lappeenrannan-Lahden teknillisellä yliopistolla oli tarve kesällä 2019 graafiselle datankeruutyökalulle, jota lähdin itsenäisesti rakentamaan tutkijoiden kanssa. Suunnittelu tapahtui yhteistyössä työnantajien kanssa mutta teknisestä toteutuksesta vastasin täysin itsenäisesti. Tässä opinnäytetyössä käyn työkalun kehitysprosessia läpi analysoiden full-stack sovellusta, mikä tästä työkalusta kehittyi.

Tämän opinnäytteen tavoitteena on osoittaa osaaminen opetelluista sovelluskehityksen teknologioista sekä samalla tuottaa kattava ohjepaketti tästä datankeruutyökalusta mahdolliselle jatkokehittäjälle.

Asiasanat

Angular, .Net, sovelluskehitys, sovellustestaus, Front-End, Back-End, SQL-tietokanta, ohjelmistokehys, Rest-rajapinta, JSON, Komponentti

Abstract

Author(s)	Type of Publication	Published
Karjalainen Antti	Thesis, UAS	2020
	Number of Pages 29	

Title of Publication

Implementation and testing of a graphical data collection tool

Angular & .Net

Name of Degree

Bachelor of Business Administration (BBA)

Name, title and organization of the client

LUT University

In the summer of 2019, Lappeenranta-Lahti University of Technology had a need for a graphical data collection tool which I set out to build independently with the researchers. The planning took place in cooperation with the researchers, but I was completely independent of the technical implementation. In this thesis, I go through the development process of this application, while analyzing the full stack application that evolved from this tool.

The aim for this thesis is to demonstrate the knowledge of the learned application development technologies and at the same time produce a comprehensive instruction package from this data collection tool for a potential further developer.

Keywords

Angular, .Net, Software development, Software testing, Front-End, Back-End, SQL-Database, Framework, Rest-API, JSON, Component

Sisällys

1 Johdanto ... 1
1.1 Opinnäytetyön tavoite ... 1
1.2 Keskeiset käsitteet ... 1
1.3 Sovelluksen kuvaus ... 3
2 Teknologiat ... 4
2.1 Front-End ... 4
2.2 Back-End ... 4
2.3 Tietokanta ... 5
3 Toteutus ... 6
3.1 Katsaus Angular-sovelluksen arkkitehtuuriin ... 6
3.1.1 Komponentit ... 7
3.1.2 Servicet ... 8
3.1.3 Admin -puoli ... 10
3.1.4 Angular-sovelluksen ja API-rajapinnan välinen yhteys ... 12
3.2 API:n rakentaminen ... 13
4 Testaaminen ... 17
4.1 Sovellustestaus ... 17
4.2 Sovelluksen sisäiset testit (Angular-sovellus) ... 17
4.2.1 Sovelluksen sisäiset testit(API) ... 18
4.3 Henkilötestaus kevät 2020 ... 19
4.3.1 Toteutettu testaus ja tulokset ... 19
4.4 Korjaukset sovellukseen testauksen perusteella ... 19
5 Sovelluksen julkaisu Azure-pilvipalveluun ... 20
5.1 Tarvittavat palvelut Azuressa ... 20
5.2 Luodun Azure-tietokannan yhdistäminen rajapintaan ... 21
5.3 Rajapinnan ja tietokannan julkaiseminen Azureen ... 22
5.3.1 Front-endin julkaisu Azureen ... 24
6 Johtopäätökset ja jatkokehitys ... 26
6.1 Pohdintaa projektista ... 26
6.2 Jatkokehitys ... 26
Lähteet ... 28

1 Johdanto

1.1 Opinnäytetyön tavoite

Vuoden 2019 kesällä aloitin työskentelyn LUT Business Managementilla, sillä heillä oli tarve saada tutkimuksen tekemistä helpottava työkalu. Tutkijat (professori Ari Jantunen, apulaisprofessori Anssi Tarkiainen ja tutkijaopettaja Anni Tuppura) tarvitsivat sovelluksen, joka helpottaisi ja nopeuttaisi tutkimuksen tekemistä. Noin vuodessa syntyi full-stack sovellus, jonka olen kokonaan itse tehnyt hyödyntäen koulussa opittua osaamista. Sovelluksen kehitys jatkuu yhä syksyllä 2020.

Tämän opinnäytetyön tavoitteena on dokumentoida Lappeenrannan-Lahden Teknilliselle Yliopistolle itse tekemäni sovellus ja näin osoittaa osaaminen opetelluista sovellus-teknologioista ja samalla tuottaa tästä opinnäytetyöstä kattava ohjepaketti datankeruutyökalun mahdolliselle jatkokehittäjälle.

1.2 Keskeiset käsitteet

Tämän opinnäytetyön keskeiset käsitteet ovat seuraavat.

Ohjelmistokehys (engl. Framework) tarkoittaa ohjelmistotuotetta, joka muodostaa rungon sovellukselle joka rakennetaan. Tässä tapauksessa Angularia käytetään sovelluksen tekemiseen. Muun muassa eri näkymien tekeminen on helppoa, kun on valmiit komennot komponenttien luomiseen.

Angular on avoimen lähdekoodin Typescript-pohjainen ohjelmistokehys(engl. Framework). Angularilla voi kehittää sovelluksia verkkoselaimille, mobiililaitteille sekä tietokoneille. Tässä työssä Angularilla tehtiin itse sovelluksen käyttöliittymä.

.NET Framework on Microsoftin kehittämä ohjelmistokomponenttikirjasto, jota Microsoft Visual Studio-ympäristössä kehitetyt ohjelmistot käyttävät**.** Tämän opinnäytetyön sovelluksessa .NET Frameworkkia käytetään rest-rajapinnan luomiseen.

Back-End tarkoittaa sovelluksen käyttäjältä piilossa olevaa koodia. Tähän lukeutuu esimerkiksi mahdollinen tietokanta tai Rest-rajapinta jos sovelluksessa niitä on käytössä.

Rest-rajapinnalla tarkoitetaan rajapintaa, jolla tietoa saadaan välitettyä tässä tapauksessa sovelluksen ja tietokannan välillä. Tietotyyppinä on JSON(JavaScript Object Notation).

JSON (JavaScript Object Notation) on tiedostomuoto tiedonvälityksessä. Json-tyyppistä todella helppoa kirjoittaa ja se muotoutuu yksinkertaisesti.

Front-End tarkoittaa sovelluksissa sitä, mitä käyttäjä itse pystyy näkemään sivustolla. Eli siis kaikki mitä selaimessa tai applikaatiossa näkyy on front-end koodia. Nettisivu tai sovellus koostuu lähes aina jonkinlaisesta html, css – tai javascript-tiedostosta tai jostain muusta tiedostosta missä määritetään applikaation logiikka.

Sovelluskehitys tarkoitettaa koko sovelluksen teko-prosessia.

Sovellustestaus tarkoittaa tässä tapauksessa sovelluksen sisäisiä testejä. Näillä testeillä on tarkoitus tutkia ja testata sovelluksen toimivuutta sekä virheettömyyttä. Näin todetaan sovelluksen koodi toimivaksi.

Komponentti tarkoittaa tässä opinnäytetyössä yhtä sovelluksen näkymää, mikä käyttäjälle tulostuu. Yhteen komponenttiin kuuluu Html, Css sekä Typescript-tiedosto.

1.3 Sovelluksen kuvaus

Sovelluksessa on admin -puoli sekä käyttäjä -puoli. Käyttäjä -puolella pääidea on se, että käyttäjä pystyy muodostamaan syy- ja seuraussuhteista relaatioita sovelluksessa. Ensimmäisessä vaiheessa käyttäjää pyydetään syöttämään tunnistetieto, jonka perusteella vastaukset tallentuvat tietokantaan. Tämän jälkeen käyttäjä valitsee listasta syyn, seurauksen ja voimakkuuden ja pystyy näin lisäämään uuden relaation. Relaatioita voi lisätä useita, ja jo lisättyjä relaatioita pystyy poistamaan. Relaatioita voi myös tarkastella sovelluksessa. Kun relaatiot on tehty, voi käyttäjä tallentaa nämä ja poistua sovelluksesta.

Admin -puolella admin pystyy muokkaamaan sovelluksessa näkyvää ohje-tekstiä, lataamaan tietokannassa olevista relaatioista yhteenvedon excel-taulukkoon, muokkaamaan käsitelistaa sekä tarvittaessa poistamaan käyttäjien täyttämiä relaatioita.

2 Teknologiat

2.1 Front-End

Front-endillä tarkoitetaan sovelluksissa sitä, mitä käyttäjä itse pystyy näkemään sivustolla. Eli siis kaikki mitä selaimessa tai applikaatiossa näkyy on front-end koodia. Nettisivu tai sovellus koostuu aina jonkinlaisesta html, css – tai javascript-tiedostosta tai jostain muusta tiedostosta missä määritetään applikaation logiikka.

Front-endiä voi tehdä usealla eri tavalla, mutta koska olin käyttänyt angularia useissa kouluprojekteissa, päädyin siihen.

Angular on sovellusten suunnittelu- ja kehitysalusta tehokkaiden yhden sivun sovellusten rakentamiseen. Angular on typescriptiä käyttävä ohjelmistokehys, ja itse olen sitä käyttänyt lähinnä tehden työpöydälle suunnattuja sovelluksia (engl. desktop-web-application). Angularissa paras puoli on mielestäni se, että rakenteet ovat selkeitä. Yksi komponentti sisältää aina html, css, spec, sekä typescript-tiedoston. Html-tiedostossa määritetään miltä sivu näyttää, css-tiedostossa asetellaan sivu, typescript-tiedostossa on kaikki logiikka sekä spec-tiedostossa mahdolliset sovelluksen sisäiset testit komponentille. Komponenttia voidaan kutsua myös näkymäksi, sillä sovellus koostuu useista eri näkymästä. (Angular 2020).

2.2 Back-End

Sovellukseen toteutin rest-rajapinnan, sillä tarvittiin jokin yhteys tietokannan ja angular-sovelluksen välille. Rajapinnan pääideana on juuri se, että sekä tietokanta että sovellus voivat keskustella keskenään. Rajapinnassa määritellään post, put, read ja delete kutsut, joita sovellus käyttää lisätäkseen, muokatakseen, lukeakseen tai poistaakseen tietoja tietokannasta. Käytännössä siis sovelluksesta otetaan yhteys rajapintaan, joka välittää tiedot tietokantaan oikeaan tauluun. Rajapintoja olisi voinut tehdä monella eri tapaa, mutta mielestäni rest-apin tekeminen ASP.NET web-apina oli hyvä ja helppo ratkaisu. Kielenä rajapinnassa toimii C#. Rajapinnan kansiorakenteessa on kolme eri tasoa, joihin koodi rakentuu. Ensin repository-tasolla määritetään suoraan tietokannan context-tiedoston avulla post,put,read ja delete kutsut. Service-tasolla käytetään näitä valmiita repository-tason funktioita, ja Controller-tasolla kutsutaan service-tason funktioita ja ajetaan ne selaimeen tiettyyn osoitteeseen. (Asp Net Core 2020).

2.3 Tietokanta

Koko sovelluksen tekeminen lähti tietokannan luomisesta. Microsoft SQL Server Management Studiolla luotiin uusi tietokanta, johon luotiin uusi taulu. Tietorakennetta suunniteltiin moneen eri kertaan, mutta päädyttiin siihen että yksi taulu riittää tähän tarkoitukseen. (SSMS 2020).

Tähän tauluun kerätään tutkimuksen kannalta tärkeät tiedot eli käyttäjän täyttämät relaatiosuhteet sekä niiden tekijä. Tärkeimpinä kolumneina taulussa ovat syy, seuraus, relaatiovoimakkuus, päivämäärä ja täyttäjän nimi. Kun käyttäjä täyttää sovelluksessa eri relaatiota, niin ne kaikki näkyvät sitten tietokannassa.

3 Toteutus

3.1 Katsaus Angular-sovelluksen arkkitehtuuriin

Angularilla pystyy tekemään yhden sivun nettisovelluksia. Tämä tarkoittaa sitä, että kaikki toiminnallisuudet ja esimerkiksi eri komponentit ladataan kerralla selaimeen, mutta niitä ei näytetä kaikkia kerralla. Angularissa index.html-tiedosto (Kuva 1) mallintaa tätä yhden sivun periaatetta. Index.html-tiedoston <app-root> tägi sisältää sovelluksen komponentit jotka halutaan tulostaa sovelluksessa. (Angular 2020).

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>DataGainerApp</title>
  <base href="/">

  <meta name="viewport" content=
  <link rel="icon" type="image/x
  <link href="https://fonts.goog
  <link rel="stylesheet" href="h
<script src="https://code.jquery
<script src="https://cdnjs.cloud
<script src="https://stackpath.b

</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Kuva 1: <app-root> tägi index.html-tiedostossa

Kansiorakenne Angular-projektissa koostuu muun muassa komponenteista, serviceistä, environmentistä, sekä tyyleistä. Kun projekti ajetaan käyntiin *ng serve* komennolla, niin selaimeen aukeaa localhost:4200 osoite, jossa <app-root> tägi tulostaa *App*-komponentin. *App*-komponentin sisällä on määritelty <router-outlet> tägi, joka taas määrittää sen että jokainen komponentti tulostuu dynaamisesti ruudulle. Sama tägi mahdollistaa myös reitit sovelluksessa. (Router Outlet 2020).

Reitit määritetään *app-routing*-komponentissa. Reiteillä tarkoitetaan osoitteita mihin aukeaa eri näkymiä eri komponentteja. Admin -puolelle navigoidaan kirjoittamalla url-kenttään '/admin'. Kun käyttäjä syöttää '/admin' niin tulostuu käyttäjälle tuohon routeen määritetty komponentti, joka on *AdminViewComponent*. Samassa yhteydessä määritetään admin-rooli tälle komponentille, joka taas uudelleen ohjaa login-näkymään (Kuva 2).

```
{path: 'admin', component: AdminViewComponent, canActivate: [AuthGuard],
        data: {roles: [Role.Admin]}},
```

Kuva 2: App-routing-komponentin yksi reitti

3.1.1 Komponentit

Sovelluksessa erilaisia komponentteja on 19. Tämä tarkoittaa sitä, että sovelluksessa on 19 erilaista näkymää joissa jokaisella näkymällä on erilainen ulkoasu (html), tyyli (css) ja logiikka (typescript). Kun käyttäjä avaa sovelluksen, niin avautuu sovelluksen alkukomponentti *Instructions*. Tämän jälkeen käyttäjää pyydetään syöttämään viiteavain *identifier* komponentissa (Kuva 3). Viiteavaimen syöttämisen jälkeen käyttäjä pääsee suoraan tekemään relaatioita *select-term*-komponenttiin. *Select-term* komponentti koostuu kolmesta dropdown-valikosta joista valitaan syy(cause), seuraus(effect) ja voimakkuus(influence). Käyttäjä voi lisätä relaatioita haluamansa määrän ja tarkastella näitä relaatioita sovelluksessa (Kuva 4).

Write the key here *
If you have received a key in advance, please give it here. If you don't have one, just type "-". *Tunniste*
Continue

Kuva 3: identifier-komponentti

Cause	Influence	Effect
Dividends	-2	Features offered
Corporate tax rate	1	Equity ratio
Growth of the firm	-1	Features offered
In-house R&D	-1	Growth of the firm
Brand, company image	1	In-house R&D

Items per page: 5 1 - 5 of 5

Kuva 4: Käyttäjän lisäämät relaatiot *select-term* komponentissa sovelluksessa

3.1.2 Servicet

Serviceitä tarvitaan sovelluksessa jos tietoa pitää säilyttää tai liikuttaa komponenttien välillä. Serviceen voidaan määritellä esimerkiksi funktioita joita voi käyttää missä tahansa sovelluksen koodissa. Servicen avulla voidaan uudelleen käyttää esimerkiksi Http-GET metodia (Kuva 5) niin, ettei sitä tarvitse kirjoittaa uudelleen tiedostokohtaisesti. (Angular Services 2020).

Http-GET() kutsussa haetaan tietokannasta admin -puolella tässä tapauksessa valmiit *Term*-oliot jotka käyttäjät ovat syöttäneet tietokantaan. Kutsussa haetaan ja palautetaan ennalta määritellystä url-osoitteesta(suora rest-rajapinnan osoite) lista relaatioita. *Term*-luokkaan kuuluu muun muassa syy-ja seuraussuhteet (luokan sisällä nimetään termId ja targetId) sekä niiden välillä oleva arvo välilllä -3...+3. Useat eri komponentit saavat käyttöönsä saman servicen helposti, ja se tapahtuu injektoimalla (Kuva 6) service komponentin constructorissa).

```
// Haetaan valmiit Json-objektit tietokannasta
get(): Observable<Term[]> {

  return this.httpClient.get(this.url).pipe(map(response => {
    return response as Term[];
  }));
}
```

Kuva 5: Esimerkki Http:n Get-metodi

```
constructor(
  private nameService: TermNameService,
  private service: TermService,
  private httpService: TermHttpService,
  private router: Router,
  private dialog: MatDialog,
  private snackbar: MatSnackBar,
  private identifierService: IdentifierService,
  private infoTextService: InfoTextService
) {
```

Kuva 6: Constructoriin injektoituja servicejä

Sovelluksessa käytetään selaimen omaa välimuistia, johon tallennetaan tietoa ennen varsinaisen Http-Post() metodin käyttämistä. *Select-term* komponentin relaatioita tallennetaan välimuistiin niin kauan, kunnes käyttäjä on täydentänyt relaatiot mieleiseksi. Vasta kun sovelluksessa painetaan *Save and Quit* – nappulaa lähtee HTTP-Post() kutsu rajapintaan. Kun nappulaa painetaan, niin relaatiolista käydään läpi yksitellen, ja jokainen *term*-olio (Kuva 8) lähetetään tietokantaan yksitellen. Tähän käytetään *filterData()*-funktiota (Kuva 7). Tämä oltaisiin myös voitu tehdä niin, että relaatiolista(välimuistista) oltaisiin laitettu vielä uuteen listaan, ja tämän jälkeen Http-Post()-kutsulla ajettu koko lista kerralla tietokantaan, mutta näin sen toimivammaksi tällä tavalla.

```
// Loopataan localstoragen termit
filterData() {
  this.identifierService.get().subscribe((result) => {
    // console.log(result);
    this.identifier = result;
  });
  this.httpService.getTerms().subscribe((result) => {
    this.lista = result;
    this.lista.sort()
  });
  // Lähetetään jokainen term yksitellen tietokantaan
  this.lista.forEach((element) => {
    const term = new Term();
    term.secondId = element.secondId;
    term.sourceId = element.sourceId;
    term.targetId = element.targetId;
    term.termValue = element.termValue;
    term.termArrow = element.termArrow;
    term.session = element.session;
    term.termDate = element.termDate;
    term.identifier = this.identifier[0].name;
    this.httpService.create(term).subscribe((result) => {
      if(term.identifier == null) {
      }
    });
  });
```

Kuva 7: FilterData()-funktio

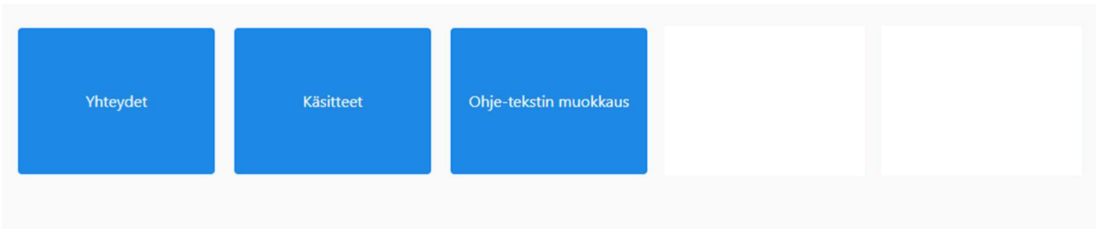
```
export class Term {
  termId: number;
  sourceId: number;
  termValue: string;
  termArrow: string;
  targetId: number;
  termDate: Date;
  session: [];
  length: number;
  secondId: number;
  identifier: string;
  constructor(termId?: number,
              termValue?: string,
              termArrow?: string, sourceId?: number,
              targetId?: number,
              termDate?: Date,
              session?: [],
              secondId?: number,
              identifier?: string

  ) {
    this.termId = termId;
    this.termValue = termValue;
    this.termArrow = termArrow;
    this.sourceId = sourceId;
    this.targetId = targetId;
    this.termDate = termDate;
    this.session = session;
    this.secondId = secondId;
    this.identifier = identifier;
```

Kuva 8: Term-Luokka

Kun käyttäjä on painanut *Save and Quit* – nappulaa, navigoi sovellus *Thank-you* – komponentille. Tässä käyttäjää kiitetään sovelluksen käytöstä ja käyttäjä voi sulkea sovelluksen.

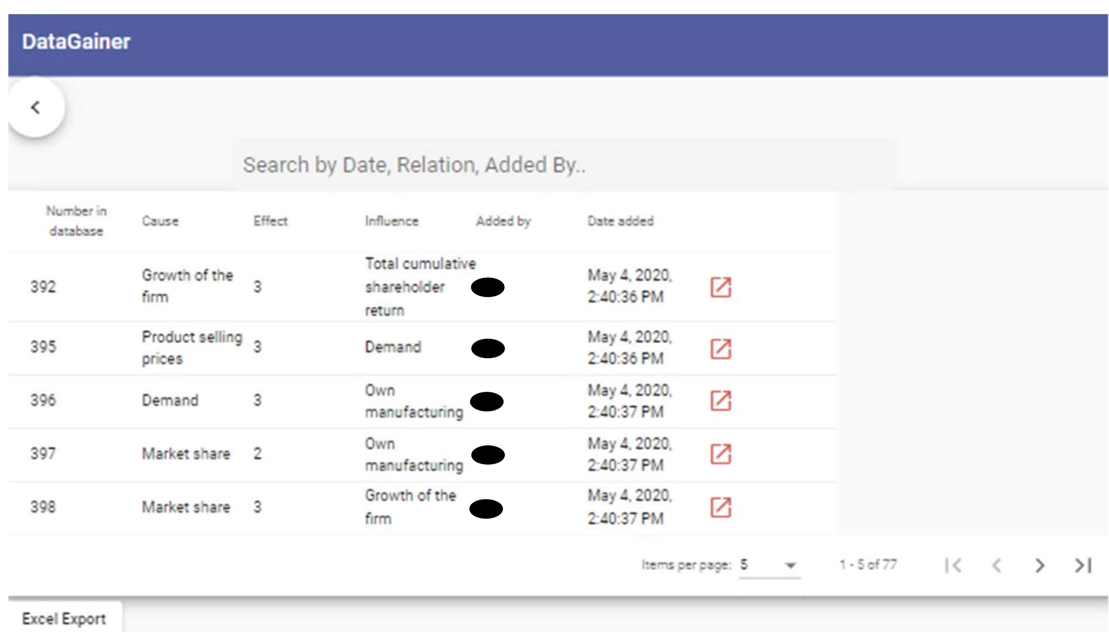
3.1.3 Admin -puoli

Admin -puolelle käyttäjä voi navigoida kirjoittamalla osoitteen perään */admin*. Login-näkymään ei tarvittu oikeaa api-pohjaista kirjautumista, joten kirjautuminen tehtiin fake-backend versiona suoraan Angular-sovelluksen koodiin. Login näkymän jälkeen käyttäjälle avautuu *admin-view* – komponentti (Kuva 9).



Kuva 9: Admin-näkymä

Yhteydet-valikosta admin pääsee näkemään koko tietokannan relaatiot kaikilta käyttäjiltä. Käyttäjä voi myös muokata ja hakea relaatioita eri nimillä tässä näkymässä. Yksi vaatimuksista sovellukseen oli Excel-integraatio, jotta saataisiin tietokanta ladattua helposti Excel-muotoisesti käyttäjälle. Tämä toiminnallisuus löytyy myös yhteydet-näkymästä (Kuva 10)



Kuva 10: Yhteydet-näkymä, jossa Excel Export vasemmassa alareunassa

Käsitteet-näkymässä admin pystyy muokkaamaan ja lisäämään käsitteitä käsitelistoihin, mistä muodostetaan relaatiot käyttäjäpuolella. Ohje-tekstin-muokkaus -näkymässä admin pystyy muokkaamaan sekä poistamaan ohje-tekstiä mikä näkyy käyttäjälle viiteavaimen syötön jälkeen. Tämäkin oli yksi vaatimuksista sovelluksessa.

3.1.4 Angular-sovelluksen ja API-rajapinnan välinen yhteys

Angular-sovellus on yhteydessä API-rajapintaan, joka pyörii Azuressa. Kun admin -puolella avataan yhteydet-näkymä niin tietokannasta tulee Term-taulusta rajapinnan kautta tiedot suoraan sovellukseen. Itse sovelluksessa HTTP Get() – kutsu määritetään termHttpService-tiedostossa. Näitä servicen metodeita kutsutaan sovelluksen komponentissa, mihin tieto halutaan hakea tai mistä tietoa halutaan lähettää (HTTP Post 2020).

Itse koodi näyttää tältä (Kuva 11):

```
// Haetaan valmiit Json-objektit tietokannasta
get(): Observable<Term[]> {

  return this.httpClient.get(this.url).pipe(map(response => {
    return response as Term[];
  }));
}
```

Kuva 11: Get-funktio termHttpService - tiedostossa

Kyseisessä funktiossa get() halutaan hakea lista relaatioita(term) tietystä url- osoitteesta. Funktioon määritetäänkin palautus suoraan httpClientin.get-metodista, johon on parametriksi liitetty this.url, joka on Azuren API:n osoite. Eli käytännössä haetaan tietokannan term-taulusta tietoa ja tuodaan se sovellukseen. Tietokannan term-taulu on samanmuotoinen kuin term-luokka sovelluksessa (Kuva 8). Kun tieto on haettu, palautetaan käyttäjälle lista relaatioista.

Muut API-kutsut noudattavat samaa logiikkaa. Samaan termHttpService-tiedostoon on määritelty myös muut tarvittavat kutsut eli HTTP Post(), HTTP Put(), ja HTTP Delete(). Post-metodi on uuden tiedon lisäämistä varten, Put-metodi on tiedon muokkaamista varten ja Delete-metodi on tiedon poistamista varten. (Angular HttpClient 2020) .

Sovelluksen käsitteet tulevat eri taulusta kuin relaatiot, ja käsitteitä varten on sovelluksessa oma service. Käsitteet tulevat taulusta nimeltä termName.

Sovelluksen käsitteitä siis käytetään relaatioiden tekemiseen. Käsitelistasta valitaan *select-term* – komponentissa kaksi eri käsitettä, ja näille valitaan relaatiovoimakkuus väliltä 3...-3. Kun käyttäjä on tehnyt valitsemansa relaatiot ja painaa save and quit-nappulaa, lähtee sovelluksesta Post-kutsu luoden kaikki nämä relaatiot Term-tauluun tietokantaan.

3.2 API:n rakentaminen

Rajapinnan rakenne muodostuu Controllereista, Modeleista, Repositoreista ja Serviceistä. Rajapinnan teko lähti liikkeelle siitä, että tietokannasta vedettiin reverse engineer-toiminnolla models-kansioon Dbcontext-tiedosto, joka määritteli tietokannan ja rajapinnan välille yhteyden. Models kansioon lisättiin malliluokat relaatioille sekä käsitteille. Repository tasolle määritetään yhteys tietokannan kanssa, ja repository-tason metodit keskustelevat suoraan tietokannan kanssa. Sekä relaatioille että käsitteille luotiin omat repositoryt, koska ne olivat eri tauluissa. Repository tasolla ensin alustettiin database-context tiedosto (Kuva 12). (Asp Net Core 2020).

```
namespace DataGainerApiV2.Repositories
{
    public class TermRepository : ITermRepository

    {
        private readonly TestdbdatagainerContext _context;
        public TermRepository(TestdbdatagainerContext context)
        {
            _context = context;
        }
```

Kuva 12: DbContext-alustus

Repository tasolle määriteltiin tarvittavat metodit: Post(Create), Put, Read(Get), Delete ja ReadById(Get by Id).

Create-funktiossa yksinkertaisesti lisätään tietokantaan uusi relaatio eli Term (Kuva 13):

```
public Term Create(Term term)
{
    _context.Add(term);
    _context.SaveChanges();
    return term;
}
```

Kuva 13: Create-funktio rajapinnassa

_context viittaa dbcontextiin, ja lisäyksen jälkeen tallennetaan ja palautetaan lisätty relaatio.

Kun haetaan yhtä tiettyä relaatiota id:n perusteella täytyi id ottaa mukaan ReadById-funktion parametriksi (Kuva 14):

```
public Term Get(int id)
{
    return _context.Term.AsNoTracking().FirstOrDefault(c => c.TermId == id);
}
```

Kuva 14: ReadById-funktio rajapinnassa

Kun repository-taso oli valmis, voitiin siirtyä serviceiden toteutukseen. Servicet ovat lähes samanlaisia idealtaan rajapinnassa kuin Angular-sovelluksessa. Niihin voidaan määritellä muutakin logiikkaa ilman että koskee repository-tasolla oleviin funktioihin. Service-tason funkiot toteutettiin samalla tavalla kuin repository-tasolla, mutta tieto tulee repository tasolta. Service toisin sanoen käyttää vain repository-tason funktioita (Kuva 15).

```
public Term Create(Term term)
{
    return _termRepository.Create(term);
}
```

Kuva 15: Service-tason Create-funktio

Controller-tasolla käsitellään pyyntöjä, joita tässä tapauksessa ovat HTTP-metodit. Controller-tasolla määritetään myös url-osoite sovellukselle. Controllerin alussa määritetään *Route*-attribuutti mikä määrittää osoitteen rajapinnalle. Tämä osoite on sovelluksessa *api/terms*. Controller-tasolla käytetään service-tason funktioita. Controller-tasolla täytyy myös määrittää samat funktiot kuin aiemminkin. Toteutus on vähän erilainen, sillä controllers-tasolla määritetään HTTP-metodit näiden funktioiden lisäksi. Aluksi injektoitiin service osaksi controlleria (Kuva 16). (API Controllers 2020).

```
public class TermController : ControllerBase
{
    private readonly ITermService _termService;
    public TermController(ITermService termService)
    {
        _termService = termService;
    }
```

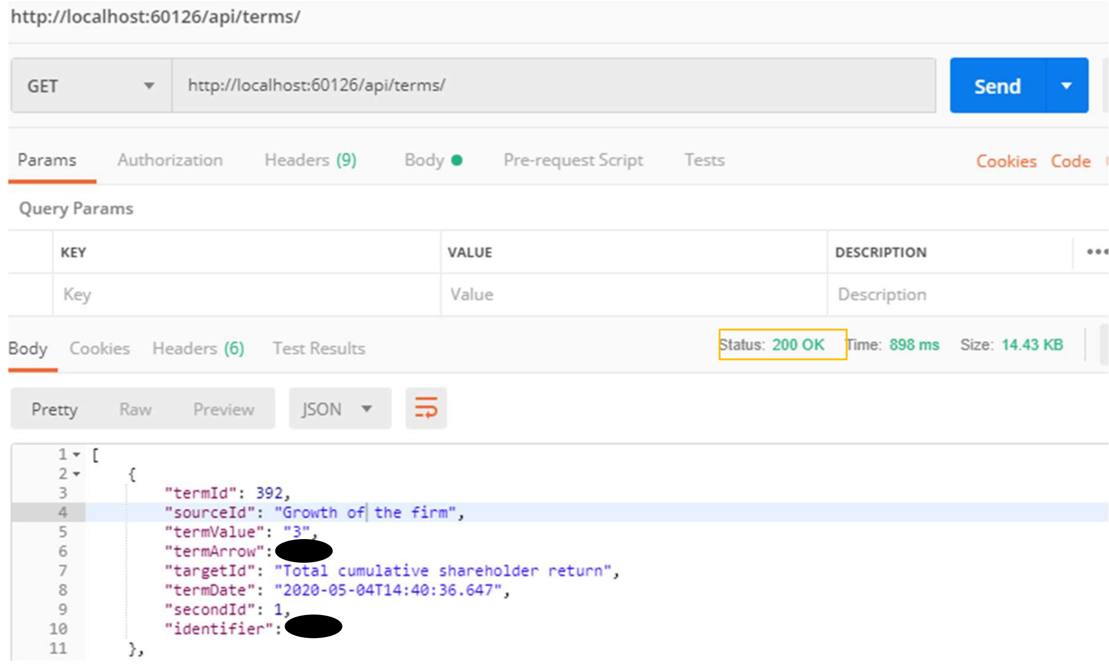
Kuva 16: TermServicen injektointi TermControlleriin

Tämän jälkeen määritetään HTTP metodit yksitellen, ja yhdistetään ne service-tason funktioihin. Esimerkkinä getById (Kuva 17):

```
// GET api/terms/{id}
[HttpGet("{id}")]
public IActionResult Get(int id)
{
    Term term = _termService.Get(id);
    return new JsonResult(term);
}
```

Kuva 17: GetById-funktio rajapinnassa

Näiden toimenpiteiden jälkeen peruspohja on kunnossa. Rajapintaa voi testata esimerkiksi Postmanilla, ja katsoa tuleeko Get-funktiolla tietoa kun rajapinta on käynnissä. Eli tehdään Get-kutsu rajapintaan /api/terms Postmanilla (Kuva 18).



Kuva 18: Yksi relaatioista palautettuna rajapinnasta

200 OK-Ilmoitus tarkoittaa sitä, että kutsu meni läpi. Rajapinta on nyt toiminnassa lokaalisti, eli tietokantaan yhdistäminen onnistui.

4 Testaaminen

4.1 Sovellustestaus

Sovellustestauksella tarkoitetaan sitä, että sovellus täytyy testata ja varmistaa toimivaksi jotta se vastaa haluttua lopputulosta. Yleisesti ottaen sovelluksen sisäisten testien tulisi kattaa 70-80% koko koodista. Tässä sovelluksessa kuitenkin päädyttiin siihen, että vain toimivuuden kannalta kriittiset osat testataan. Front-endissä tämä tarkoitti *select-term* -komponenttia, ja back-endissä jokaista repository-tason funktiota(Create, Get, GetById, ja Delete). Kun nämä on testattu ja todettu toimiviksi, voidaan katsoa että sovellus on toimiva. (Steve Cornett 2013).

4.2 Sovelluksen sisäiset testit (Angular-sovellus)

Angularissa testaus tapahtuu komponentin .spec-tiedostossa, joka generoituu automaattisesti kun uusi komponentti luodaan. Ajamalla *ng test* komennon terminaaliin, aukeaa Karma-niminen testiympäristö. Tästä voidaan tarkastella mitä testejä menee läpi (Kuva 19). (Angular Testing 2020):

Kuva 19: Testit mitkä menevät läpi komponentissa

Ng test-komentoon voidaan lisätä tägi -code coverage joka antaa lisää tietoa komponentista. Coverage tägi antaa prosenttilukuina palautteen komponentin sisällä yhteensä toteutuneista testeistä. (Kuva 20).

```
==============================
Statements   : 87.8% ( 36/41 )
Branches     : 60% ( 6/10 )
Functions    : 100% ( 8/8 )
Lines        : 87.18% ( 34/39
==============================
TOTAL: 5 SUCCESS
TOTAL: 5 SUCCESS
```

Kuva 20: Code-coverage raportti karmasta

4.2.1 Sovelluksen sisäiset testit(API)

Rajapinnassa testit tehdään hieman erilailla. Testejä varten täytyy luoda uusi xUnit – projekti vanhan projektin juureen. xUnit-projektin sisällä testataan tarvittavat funktiot, ja testit voi tarkistaa ajamalla rajapinnan käyntiin. Rajapinnassa testien täytyy mennä läpi, sillä kaikki funktiot mitä repository-tasolta löytyy, ovat kriittisiä. Testausfunktioiden kirjoittamisessa käytetään fact-attribuuttia,jotta rajapinta osaa lukea funktion testattavaksi. Rajapintaan toteutetaan neljä erilaista testiä: Relaatioiden hakemista (Kuva 21), poistamista, hakemista id:n perusteella sekä lukemista varten. Rajapintaan olisi myös hyvä toteuttaa testit funktioiden virheitä varten. Esimerkiksi get-funktiolle olisi hyvä olla testi siitä, että jos funktio ei mene läpi, niin ohjelma palauttaa vähintään virheilmoituksen käyttäjälle. (Unit Testing 2020).

```
// test if returns the list of terms in database
[Fact]
public void TestIfReturnsListOfTerms()
{

    var okResult = _controller.Get();

    Assert.IsType<JsonResult>(okResult.Result);

}
```

Kuva 21: Get-funktion testaus rajapinnassa

Jokainen testi kirjoitetaan erikseen fact-attribuutilla, ja kun testit ovat valmiit voidaan ne ajaa läpi valitsemalla rajapinnasta *Run Tests*.

4.3 Henkilötestaus kevät 2020

Sovellusta testattiin keväällä 2020. Testaus toteutettiin niin, että testausjoukolle lähetettiin ohjeistus word-tiedostona, sekä linkki itse sovellukseen. Testituloksia tuli yhteensä 10 eri henkilöltä. Testaus piti alunperin suorittaa isommalle joukolle, mutta koronarajoitusten myötä fyysinen testaus paikan päällä ei ollut mahdollista, joten päädyttiin etätestaukseen. Testihenkilöistä suurin osa oli it-alan opiskelijoita, ja tulokset olisivat varmasti olleet erilaiset jos testaukseen olisi osallistunut muitakin henkilöitä.

4.3.1 Toteutettu testaus ja tulokset

Word-dokumentissa ohjeistettiin käyttäjää täyttämään viiteavaimen kohdalle oma nimimerkki sekä lukemaan sovelluksen sisäiset ohjeet tarkasti läpi. Testauksessa kysyttiin käytettävyydestä, käyttöliittymästä ja relaatioiden lisäämisestä. Suurin osa vastaajista oli sitä mieltä, että relaatioiden-valinta näkymässä (select-term-komponentti) dropdown-listan käsitteet tulisi olla aakkosjärjestyksessä. Muuten palaute oli pääosin ulkonäköön liittyvää, ja näitä mielipiteitä on niin monta kuin on testaushenkilöitäkin. Sovellusta on kehitetty vielä marraskuuhun 2020 asti, ja isompi testaus on varmasti paikallaan heti, kun koronarajoitukset hellittävät.

4.4 Korjaukset sovellukseen testauksen perusteella

Aakkosjärjestys korjattiin sovellukseen nyt syksyllä 2020. Vaihtoehtoina oli syöttää koko tietokanta kokonaan uudestaan niin, että lista on aakkosjärjestyksessä, tai loogisempana vaihtoehtona oli vain käyttää typescriptin omaa sort()-funktiota. Sort-funktioon voidaan syöttää parametreinä ehtoja siitä, miten lista tulee järjestää. Tässä tapauksessa haluttiin lista aakkosjärjestykseen (Kuva 22).

```
// haetaan tietokannasta käsitteet termName taulusta, ja järjestetään aakkosjärjestykseen
    this.nameService.get().subscribe((result) => {
      this.termNames = result;
      this.termNames.sort((a, b) => a.termName > b.termName ? 1 : -1);

    });
```

Kuva 22: Listan filtteröinti sort-funktiolla

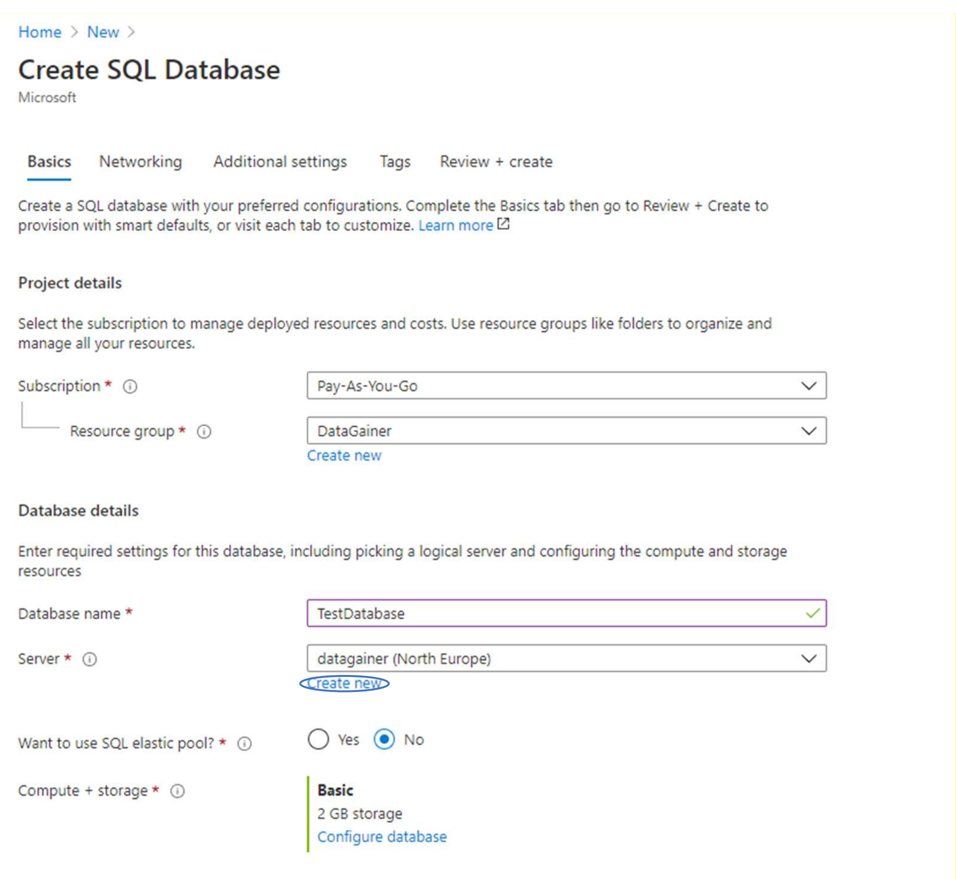
5 Sovelluksen julkaisu Azure-pilvipalveluun

Tässä osiossa käyn läpi mitä eri vaiheita kuuluu sovelluksen julkaisuprosessiin, jos sovelluksen kaikki osat haluaa julkaista pyörimään Azureen.

5.1 Tarvittavat palvelut Azuressa

Jotta kaikki palikat saadaan Azureen, täytyy ensin Azuren kotisivuilla käydä luomassa muutamia eri resursseja. Tarvitaan serveri, tietokanta ja app-service sekä Angular-sovellukselle että rajapinnalle. Azuren kotinäkymässä *create a resource* nappulaa painamalla voidaan luoda uusia resursseja Azure-tilille. Tarvitaan siis server, tietokanta sekä kaksi eri app-serviceä.

Aloitetaan tietokannasta. Kun tietokannan tietoja täyttää, niin pitää valita jokin storage-sopimus. Kun klikkaa *configure database*, niin pääsee valitsemaan halvemman 2GB:n sopimuksen. Tämä on noin 4 euroa kuukaudessa. Kun tietokantaa luodaan, niin server-kohdasta valitaan *create new* (Kuva 23).



Kuva 23: Tietokannan luonti Azuressa

Oikeaan yläreunaan ilmestyy *new server* valikko, mihin voidaan täyttää uuden serverin tiedot. Nämä tiedot kannattaa laittaa johonkin ylös, sillä niitä tarvitaan kun kirjaudutaan tietokantaan. Kun serveri ja tietokanta on tehty voi niille kirjautua sisään SQL Server Management Studiolla.

5.2 Valmiin Azure-tietokannan yhdistäminen rajapintaan

Kun tietokanta on luotu, se pitää yhdistää rajapintaan. Rajapinta-projektin *Startup.cs* tiedostossa täytyy määritellä että käytetään Azuren connection stringiä, mikä asetetaan *appsettings.cs* tiedostossa. Connection string on merkkijono, jossa on tietolähde sekä tiedot tietolähteeseen yhdistämiseen. Eli tässä tapauksessa connection string sisältää tietoa Azuren tietokannasta ja serveristä, sekä käyttäjänimestä ja salasanasta millä sinne kirjaudutaan. Connection stringin käyttö määritellään *startup.cs* tiedostoon (Kuva 24). (Connection String 2020).

```
services.AddDbContext<TestdbdatagainerContext>(options =>
{
    options.UseSqlServer(Configuration.GetConnectionString("AzureDbConnection"));
});
```

Kuva 24: Connection stringin nimi startup.cs-tiedostossa

Kun Connection string on asetettu pitää se vielä määritellä oikeaksi *appsettings.json* tiedostoon. Connection stringin saa haltuun navigoimalla Visual Studion sisällä view→ Server explorer → Data connections → Properties ja connection string. Nyt asetetaan connection string AzureDbConnection nimiseen muuttujaan jotta ohjelma osaa sen hakea oikeasta paikasta (Kuva 25).

```
AzureDbConnection": "Data Source=<*Serverin osoite*>;Initial Catalog=*Tietokannan nimi*;Integrated Security=True; User ID=*käyttäjänimi*;Password=*Salasana*",
```

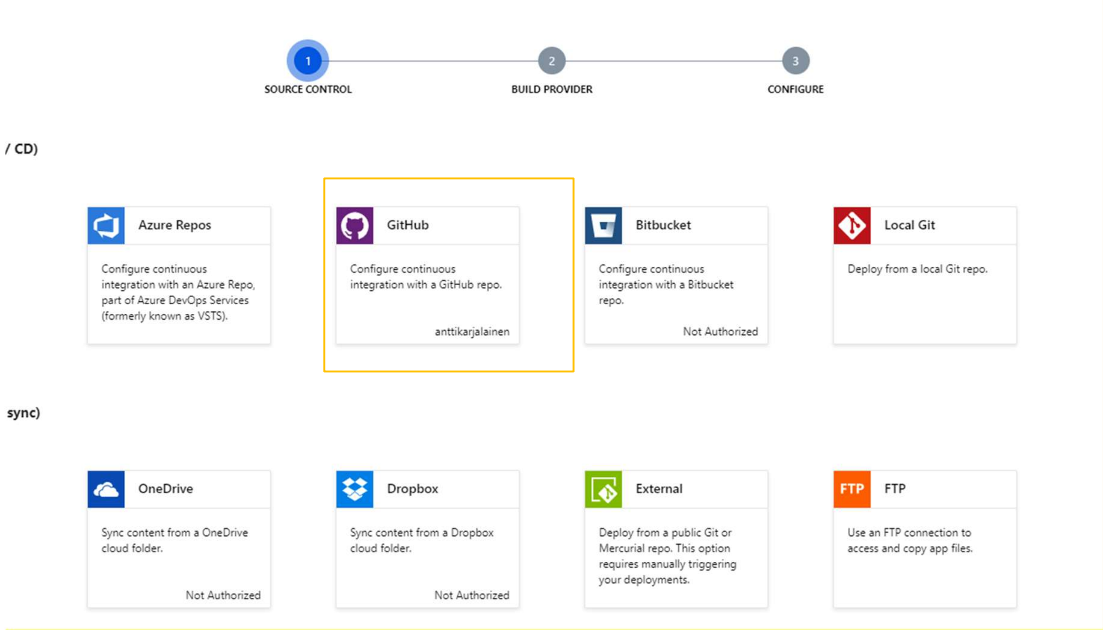
Kuva 25: Connection string esimerkki

Kun connection string on asetettu paikoilleen, on rajapinta valmis julkaistavaksi.

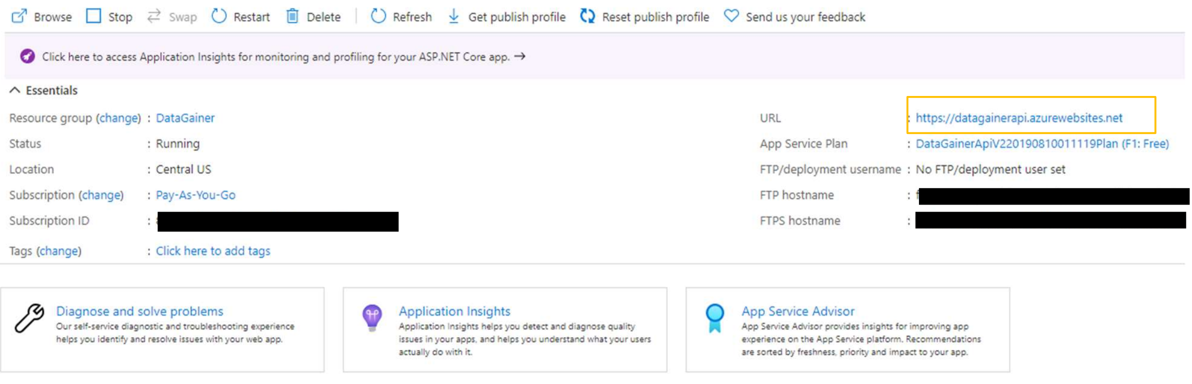
5.3 Rajapinnan ja tietokannan julkaiseminen Azureen

Rajapinnan julkaiseminen lähtee liikkeelle app-servicen tekemisestä Azureen (Kuva 26). Tämän jälkeen voidaan valita, miten rajapinta kytketään app-serviceen. Sen voi liittää github-linkin kautta, tai sen voi julkaista suoraan Microsoftin Visual Studiolla. Kun rajapinta ja tietokanta on keskenään yhdistetty, niin riittää että rajapinta julkaistaan app-serviceen. Kun app-service on luotu, navigoidaan juuri tehtyyn app-service resurssiin. Täältä valitaan deployment center, ja annetuista vaihtoehdoista valitaan github (Kuva 27). Seuraavaksi käyttäjää pyydetään syöttämään github-tunnukset, sekä valitsemaan repository mistä rajapinnan koodi tulee. Kun nämä on valittu, niin Azure alkaa yhdistämään github-repositorista löytyvää rajapintaa app-serviceen. Kun prosessi on valmis, ilmoittaa Azure että app-service on julkaistu. Navigoimalla resurssiin ja painamalla sovelluksen linkkiä voit testata rajapinnan toimivuutta (Kuva 28).

Kuva 26: App-servicen luonti Azureen



Kuva 27: Github valittuna Azuressa



Kuva 28: Rajapinnan osoite azuressa

5.3.1 Front-endin julkaisu Azureen

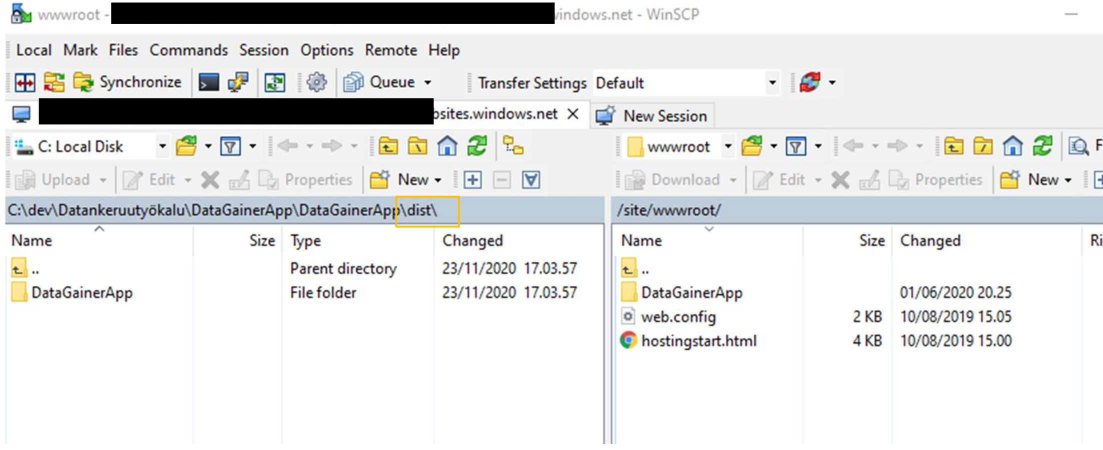
Angular sovelluksen julkaisu poikkeaa rajapinnan julkaisusta. Vaihtoehtona oli github-julkaisu myös itse sovellukselle mutta mielestäni FTP:tä käyttäen onnistui kätevämmin. FTP eli File transfer protocol tarkoittaa tapaa millä siirtää tiedostoja. Protokollia on useampia ja tunnetuin niistä on HTTP. FTP-yhteys toimii siten, että asiakas ottaa omalta tietokoneeltaan yhteyden haluttuun palvelimeen: Otetaan siis Azuren FTP-palvelimelle yhteys omalta tietokoneelta, ja lähetetään palvelimelle angular-sovellus. (Jon Martidale 2020).

Aluksi täytyy tehdä app-service samalla tavalla, kuin rajapinnan julkaisussa. Deployment center – kohdasta valitaan tällä kertaa FTP. Kun tämä valitaan, aukeaa näkymään FTP:ssä käytettäviä tietoja joita tarvitaan, eli nämä täytyy kirjata ylös. Kirjattavia tietoja on käyttäjänimi (username), salasana (password) ja palvelimen osoite (host name). Itse käytin FTP-yhteyden luomiseen WinSCP-nimistä työkalua. WinSCP on avoimeen lähdekoodiin perustuva FTP-yhteyksien luontiin tehty työkalu. (WinSCP 2020).

Kun avaa WinSCP-työkalun, täytyy aluksi syöttää Azuresta saadut käyttäjänimi, salasana ja palvelimen osoite. Näiden jälkeen pääset kirjautumaan palvelimelle. Käyttöliittymässä voi drag and drop-tyyppisesti pudottaa vasemmalta oikealle tiedostoja. Vasen ikkuna on lähde ja oikea on kohde. Eli nyt halutaan pudottaa oikeaan ikkunaan buildattu sovelluskansio.

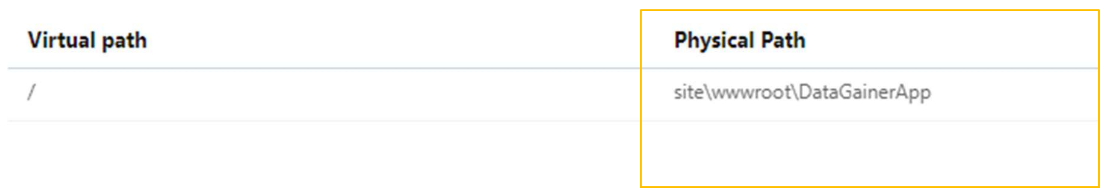
Angular-sovelluksen buildaus tapahtuu komennolla *ng build* johon voi lisätä erilaisia parametrejä jotta esimerkiksi sovelluksen kokoa voidaan modifioida. Projektin juuressa ajetaan komento *ng build* parametreillä: *ng build --prod --base-href / --aot --build-optimizer=true --vendor-chunk=true --optimization=true.* (Angular Build 2020).

Edellä mainittu komento generoi buildatun version dist kansioon projektin sisälle. WinScp työkalussa halutaan lähettää palvelimelle dist-kansion sisältö, ja se onnistuu raahaamalla vasemmalta oikealle (Kuva 29).



Kuva 29: Kuvakaappaus WinSCP-ohjelmasta

Kun siirto on valmis, täytyy Azuren app-servicen configuration-settings kohdassa käydä vaihtamassa polku vastaamaan palvelimen polkua. Eli kirjoitetaan dist-kansion sisällä olevan kansion nimi polun jatkoksi (Kuva 30).



Kuva 30: /DataGainerApp lisättynä polkuun

Sovelluksen toimivuutta voi testata navigoimalla app-servicestä löytyvään linkkiin, joka ohjaa itse sovellukseen. Kun linkki toimii, on julkaisuprosessi valmis.

6 Johtopäätökset ja jatkokehitys

6.1 Pohdintaa projektista

Sovelluksen kehityksen aikana yhteistyö tutkijoiden kanssa on ollut mutkatonta ja työ on edennyt hyvin ja kaikki osapuolet ovat olleet tyytyväisiä. Projektista tärkeimpänä oppina itselle on jäänyt käteen sovelluksen suunnitteluvaihe. Jos kehitystyön alussa oltaisiin osattu määritellä suoraan tämänkaltainen sovellus, olisi sen kehitys varmasti ollut nopeampaa. Kuitenkin ajatukset vaihtuvat sovelluskehityksessä nopeasti, ja uusia ideoita sovellukseen saattaa tulla lyhyellä aikavälillä useitakin. Vaatimusmäärittelyn tärkeys on todella iso osa sovelluskehitystä ja sen tekeminen perusteellisesti auttaa sovelluskehitystä huomattavasti. Toisena oppina itselle jäi käteen se, että koodin sisäiset nimeämiset tulisi hoitaa jatkossa mahdollisimman yksinkertaisesti ja järkevästi. Nimesin joitakin komponentteja samoilla nimillä, ja jossain vaiheessa koodirakenteesta tuli niin suuri, että oli hankalaa löytää tiettyjä komponentteja koodista, kun komponentit oltiin nimetty epäselvästi. Kun kansiorakenne on kunnossa, on sitä paljon mukavampi myös muiden lukea.

Sovellus otetaan käyttöön luultavasti vuoden 2021 alussa, ja sitä päästään testaamaan kunnolla. Tutkijat ovat olleet tyytyväisiä työkaluun, ja se toimii niin kuin pitää, joten lopputulos on onnistunut.

6.2 Jatkokehitys

Syksyllä 2020 sovellukseen on suunniteltu muutamia uusia jatkokehityskohteita, joista osa on jo toteutettu. Sovellukseen suunniteltiin esivalinta -näkymä, josta käyttäjä voi filtteröidä käsitelistasta pois haluamiaan käsitteitä. Admin pystyy admin -puolella säätämään minimi ja maksimi arvoja jotka käyttäjän täytyy esivalinta -näkymässä valita. Esimerkiksi jos admin asettaa arvot 10 ja 20 admin -puolella(Kuva 31), niin käyttäjän täytyy ennen varsinaista relaatioiden muodostamista esivalinta -näkymässä valita käsitelistasta vähintään 10 ja enintään 20 käsitettä, joita relaatioiden muodostamisessa käytetään.

Valitse vähintään 10 käsitettä ja enintään 20 käsitettä

- ☑ Brand, company image
- ☐ Buying technology and design licences
- ☐ Capacity allocation
- ☐ Competition in the market
- ☐ Consumer price elasticity
- ☐ Contract manufacturing
- ☐ Corporate tax rate
- ☐ Demand
- ☐ Dividends
- ☐ Employee training and education
- ☐ Equity ratio
- ☐ Features offered
- ☐ Growth of the firm
- ☐ In-house R&D
- ☐ Interest rates
- ☐ Internal loans
- ☐ Inventory management
- ☐ Investment in production plants
- ☐ Logistics priorities
- ☐ Long-term debt
- ☐ Long-term profitability
- ☐ Market selection decisions
- ☐ Mission and vision
- ☐ Network coverage
- ☐ Number of R&D personnel
- ☐ Number of shares outstanding
- ☐ Own manufacturing
- ☐ Product selling prices
- ☐ Product-market decisions (Technology)
- ☐ Promotion
- ☐ R&D employee turnover
- ☐ Sales
- ☐ Short-term Profitability
- ☐ Total cumulative shareholder return
- ☐ Transfer prices
- ☐ Transportation costs
- ☐ Wages of R&D employees

Vähintään 10 pitää olla valittuna
Enintään 20 pitää olla valittuna

Jatka

Kuva 31: Esivalinta-näkymä

Admin -puolelle toteutettiin myös relaatiovoimakkuuksien-muokkaus mahdollisuus. Alunperin relaatiovoimakkuudet olivat vain kovakoodattuja numeroita välillä -3..3. Siirsin tämän listan tietokannan tauluun, ja vaihdoin tietotyyppiä niin, että numeroiden lisäksi relaatiovoimakkuuksiin pystytään laittamaan myös tekstiä. On tarpeellista, että jos tutkimuksessa käytetään esimerkiksi vain arvoja positiivinen ja negatiivinen, niin pystyy nämä vaihtoehdot syöttämään admin -puolella relaatiovoimakkuus listaan.

Lähteet

1. Angular 2020. Virallinen Angularin dokumentaatio. Viitattu 20.11.2020

   Saatavissa: https://angular.io/docs

2. Asp Net Core. Virallinen Microsoftin Asp.Net dokumentaatio. Viitattu 25.11.2020

   Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-5.0

3. SSMS. Virallinen Microsoftin dokumentaatio SQL Server Studiosta. Viitattu 25.11.2020

   Saatavissa: https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15

4. Router Outlet. Virallinen Angularin dokumentaatio RouterOutletista. Viitattu 25.11.2020

   Saatavissa: https://angular.io/api/router/RouterOutlet

5. Angular Services 2020. Virallinen Angularin dokumentaatio. Viitattu 20.11.2020

   Saatavissa: https://angular.io/guide/architecture-services

6. Angular HttpClient. Virallinen Angularin dokumentaatio. Viitattu 23.11.2020

   Saatavissa: https://angular.io/guide/http

7. API Controllers. Microsoft .NET virallinen dokumentaatio. Viitattu 25.11.2020

   Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-5.0

8. Steve Cornett. Steve Cornettin artikkeli sovellustestauksesta. Viitattu 23.11.2020

   Saatavissa: https://www.bullseye.com/minimum.html

9. Angular Testing. Virallinen Angularin dokumentaatio. Viitattu 23.11.2020

   Saatavissa: https://angular.io/guide/testing

10. Unit Testing. Microsoftin virallinen dokumentaatio. Viitattu 23.11.2020

    Saatavissa: https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-

11. Connection Strings. Microsoftin virallinen dokumentaatio. Viitattu 23.11.2020

Saatavissa: https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/creating-a-connection-string

12. Jon Martidale. Digitaltrendsin artikkeli. Viitattu 25.11.2020

Saatavissa: Saatavissa: https://www.digitaltrends.com/computing/what-is-ftp-and-how-do-i-use-it/

13. WinSCP. WinSCP:n virallinen dokumentaatio. Viitattu 25.11.2020

Saatavissa: https://winscp.net/eng/docs/introduction

14. Angular Build. Angularin virallinen dokumentaatio. Viitattu 25.11.2020

Saatavissa: https://angular.io/cli/build