jamk.fi

Siirrettävyyden toteutus ASP.NET-sovelluksessa

Case: Eitbit Oy

Satu Rajala, Emilia Niemi

Opinnäytetyö
Maaliskuu 2017
Liiketalouden ala
tradenomi (AMK), tietojenkäsittelyn koulutusohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Rajala, Satu Niemi, Emilia	Opinnäytetyö, AMK	Maaliskuu 2017
	Sivumäärä 45	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Siirrettävyyden totetutus ASP.NET-sovelluksessa
Case: Eitbit Oy

Tutkinto-ohjelma
Tietojenkäsittelyn koulutusohjelma

Työn ohjaaja(t)
Tuikka, Tommi

Toimeksiantaja(t)
Eitbit Oy

Tiivistelmä

Microsoft-teknologioilla on pystytty tekemään siirrettäviä asiakaspuolen ja palvelinpuolen sovelluksia esimerkiksi Mono-alustaa käyttäen. Opinnäytetyön toimeksiantaja, Eitbit Oy:n toimitusjohtaja Poikolainen, on kuitenkin huomannut aukkoja ASP.NET-sovellusten siirrettävyydessä. Kevään 2016 Build-seminaarien esitysten perusteella näytti siltä, että siirrettävyys olisi kehittymässä ja muodostumassa Microsoft-arkkitehtuurin perustaksi.

Opinnäytetyön tavoitteena oli tutustua muutamaan siirrettävyyttä helpottavaan tekniikkaan ja luoda yksinkertainen sovellus, jossa tekniikoita kokeiltiin käytännössä. Sovelluksen tarkoituksena oli myös testata millaista työtä arkkitehtuurien kerrosten siirtäminen toiselle laitealustalle vaatii.

Opinnäytetyö toteutettiin kehittämistutkimuksena. Sovelluksessa sekä teoriaosuudessa käytettiin .NET Core -, Angular 2 -, Ionic 2 -ja TypeScript -tekniikoita. Näitä käyttäen luotiin uutissovellus, jossa siirrettävyyttä testattiin sovelluksen asetusnäkymän kautta: asetusnäkymässä sai valita kumpaa kahdesta tietokannasta, MongoDB vai DocumentDB, käytetään ja mitä kolmesta palvelimesta, Windows, Linux vai Docker, käytetään. Luotu sovellus toimi pääpiirteittäin, mutta erityisesti Linux-palvelimella esiintyi ongelmia, jotka estivät siirrettävyyden testaamisen Linuxin puolella. Tätä virhettä ei työssä ratkaistu.

Teoriaosuudesta voidaan päätellä, että Microsoft-ympäristöön on tullut edellisten tekniikoiden rinnalle lisää muita tekniikoita, jotka pyrkivät helpottamaan koodin siirrettävyyttä sekä tekemään sovelluksista järjestelmäriippumattomia. Kattavien dokumentointien ansiosta näitä tekniikoita on suhteellisen helppo käyttää.

Avainsanat (asiasanat)
Siirrettävyys, ASP.NET Core, Angular 2, Ionic 2, Eitbit Oy, alustariippumattomuus

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Rajala, Satu Niemi, Emilia	Bachelor's thesis	March 2017
		Language of publication: Finnish
	Number of pages 45	Permission for web publication: x

Title of publication

Implementation of portability in ASP.NET application

Case: Eitbit Oy

Degree programme

Business Information Systems

Supervisor(s)

Tuikka, Tommi

Assigned by

Eitbit Oy

Abstract

Microsoft technologies such as Mono have enabled portable client-side and server-side applications; however, Mr. Poikolainen, the CEO of Eitbit Ltd and the commissioner of the thesis, has noticed some limitations in porting ASP.NET applications. In the spring of 2016, it was demonstrated on Build -seminars that the porting of these applications was being developed and becoming the base of Microsoft's architecture.

The goal of the thesis was to study a few technologies that make porting easier and to create a simple application that uses those techniques in practice. The application was also meant to be a test that shows how much work goes into porting different architectural platforms into other environments.

The thesis was realized as a design-based research. Technologies such as .NET Core, Angular 2, Ionic 2 and TypeScript were used in the application and introduced in the theory part. The created application was a news application. Its porting abilities could be tested via its settings page: the user could choose which of the three servers, Windows, Linux or Docker, and which of the two databases, MongoDB or DocumentDB, they wanted to use. The application functioned fairly well, however, there were problems, especially on the Linux server that prevented the testing of portability on Linux. This error was not solved in the thesis.

Based on the theory part, it can be seen that Microsoft environment has now more techniques that help with porting the code and make applications work fluently as cross-platform. The technologies are fairly easy to use due to their comprehensive documentations.

Keywords/tags (subjects)

Portability, ASP.NET Core, Angular 2, Ionic 2, Eitbit Oy, cross-platform

Miscellaneous

Sisältö

Kuviot

Termit

Android: Googlen omistama mobiililaitteille suunniteltu ohjelmistopino, joka sisältää käyttöjärjestelmän lisäksi perusohjelmia sekä väliohjelmistoja.

Arkkitehtuuri: Projektin rakenne, sen ulospäin näkyvät ominaisuudet ja niiden väliset yhteydet ja riippuvaisuudet. Muodostaa rungon suunnittelulle ja toteutukselle.

Asiakaspuoli (engl. client side): Toiselta nimeltään front-end, jolla tarkoitetaan ohjelmoinnissa sitä sovelluksen osaa, joka näkyy käyttäjälle selaimessa.

ASP.NET: Microsoftin kehittämä avoimen lähdekoodin ohjelmistokehys (engl. framework) jota käytetään nykyaikaisten websovellusten sekä -palveluiden luomisessa.

C#: Microsoftin kehittämä yksinkertainen ja moderni olioperustainen ohjelmointikieli. Sitä käytetään yleisesti esimerkiksi .NET-ohjelmistokehyksessä.

Docker: Alusta, jonka avulla voidaan paketoida tietokoneohjelmat eräänlaisiksi konteiksi. Niiden avulla ohjelma pyörii samalla lailla ympäristöstä riippumatta.

Emulaattori: Laitteisto tai tietokoneohjelma, joka esittää olevansa jokin muu laitteisto tai ohjelma, jonka kanssa tiettyjen komponenttien tulisi pystyä toimimaan.

HTML: Standardi merkkikieli nettisivujen luomiseen. Se on lyhenne sanoista Hypertext Markup Language.

Hybridi mobiilisovellus: Mobiililaitteisiin erikseen ladattava web-sovellus, joka on toteutettu HTML5-kielellä. Hybridisovelluksen käyttökokemus ei aina vastaa natiivia. Esimerkiksi NativeScript vastaa natiivia.

Intranet: Verkkosivusto tai -palvelu, joka on tarkoitettu tietyn yhteisön sisäiseksi kommunikointi- sekä työvälineeksi.

iOS: Applen mobiililaitteille kehitetty käyttöjärjestelmä.

JavaScript: Dynaaminen ohjelmointikieli, joka tekee nettisivuista interaktiivisia.

Linux: Unixin kaltainen käyttöjärjestelmä, joka käyttää Linux-ydintä.

Microsoft Visual Studio: Microsoftin kehittämä sovelluskehitystyökalu.

Microsoft Azure: Microsoftin julkinen pilvialusta. Tarjoaa erilaisia pilvipalveluita kuten varastointia, virtuaalikoneita ja kehitysympäristöjä.

Natiivisovellus: Sovellus, joka on ohjelmoitu vain tietylle laitealustalle. Esimerkiksi Android- ja iOS-sovellukset ohjelmoidaan eri kielillä, joten Androidin natiivisovellus ei toimi iOS-alustalla eikä toisinpäin.

NuGet: Paketinhallintatyökalu Microsoftin kehitysalustoille.

Ohjelmistokehys: Ohjelmistokehys (englanniksi framework) toimii nimensä mukaisesti kehyksenä jollekkin ohjelmistolle.

Ohjelmistorajapinta: Ohjelmistorajapinta (englanniksi Application Programming Interface, API) on työkalu, joka mahdollistaa eri ohjelmistojen sekä sovellusten välisen keskustelun. Keskustelussa välitetään toimintoja tai tietoja toiselle osapuolelle.

Palvelinpuoli: Palvelinpuoli eli back-end vastaa sovelluksen siitä osasta, joka on sitä ylläpitävällä palvelimella. Sovelluksen käynnistyessä ensin ladataan palvelinpuoli ja sitten asiakaspuoli.

REST: Sovelluksen arkkitehtuurimalli (englanniksi Representational State Transfer eli REST), joka käyttää HTTP-protokollaa kommunikoimiseen.

SDK: SDK on lyhenne sanoista Software Development Kit. Se tarkoittaa erilaisten sovelluskehityksessä tarvittavien työkalujen muodostamaa pakkausta.

UAP: UAP on lyhenne sanoista Universal App Platform. Se on API-pinta Windows laitteille.

UML: Graafinen mallinnuskieli, joka on lyhenne sanoista Unified Modelling Language. Se on Object Management Group (OMG) -yhdistyksen standardoima.

XAML: XAML eli Extensible Application Markup Language on XML-pohjainen merkintäkieli, jonka Microsoft on kehittänyt. XAML tekee .NET-sovelluksien käyttöjärjestelmien luomisesta yksinkertaisempaa.

1 Johdanto

Sovelluskehityksessä siirrettävyydellä tarkoitetaan sitä, että sovellus toimii usealla eri alustalla, esimerkiksi eri käyttöjärjestelmillä (Mooney n.d., 1). Toimeksiantajamme Timo Poikolaisen mukaan Microsoft-teknologioilla on jo aikaisemmin pystytty toteuttamaan siirrettäviä client- ja server-sovelluksia, esimerkiksi Monoa hyödyntäen, mutta ASP.NET-sovellusten siirrettävyydessä on ollut aukkoja. Poikolainen oli seurannut kevään 2016 Build-seminaarin esityksiä ja todennut, että niiden sekä yleisten trendien perusteella siirrettävyys olisi kehittymässä ja muodostumassa Microsoft-arkkitehtuurin perustaksi. Seminaareissa oli tuotu esiin myös TypeScript-kieli, jonka rooli olisi kasvamassa siirrettävien sovellusten kehittämisessä.

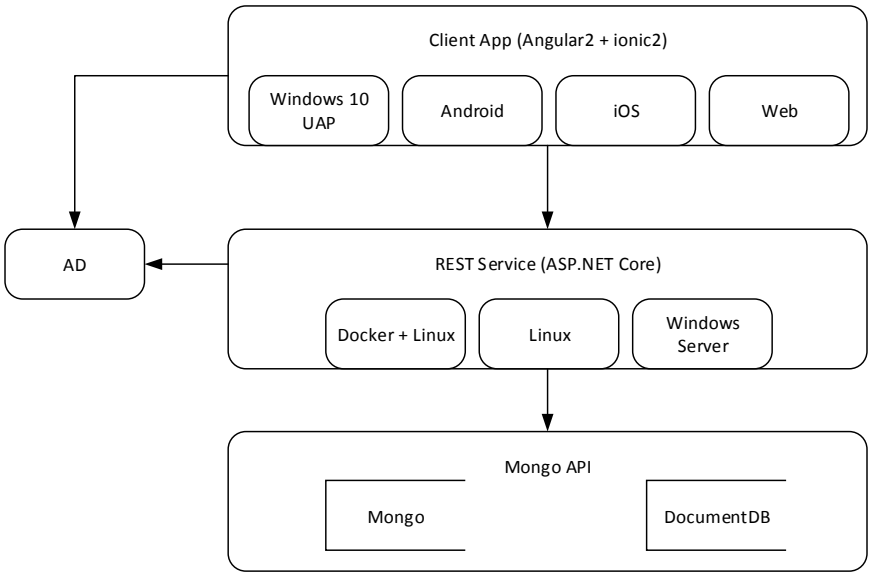
Toimeksiantajan toimesta opinnäytetyössä tutkitaan ASP.NET-sovellusten siirrettävyyttä tutustumalla uusiin tekniikoihin sekä rakentamalla yksinkertaisen intranet-uutissovelluksen, jonka arkkitehtuurin kaikki kolme kerrosta ovat rakennettu siirrettäväksi. Työssä käytetään pääasiallisesti Visual Studio ohjelmankehitysympäristöä sekä teknologioita kuten ASP.NET Core, Angular 2, Ionic 2 ja TypeScript.

2 Tutkimusasetelma

2.1 Toimeksiantaja, tavoitteet ja rajaukset

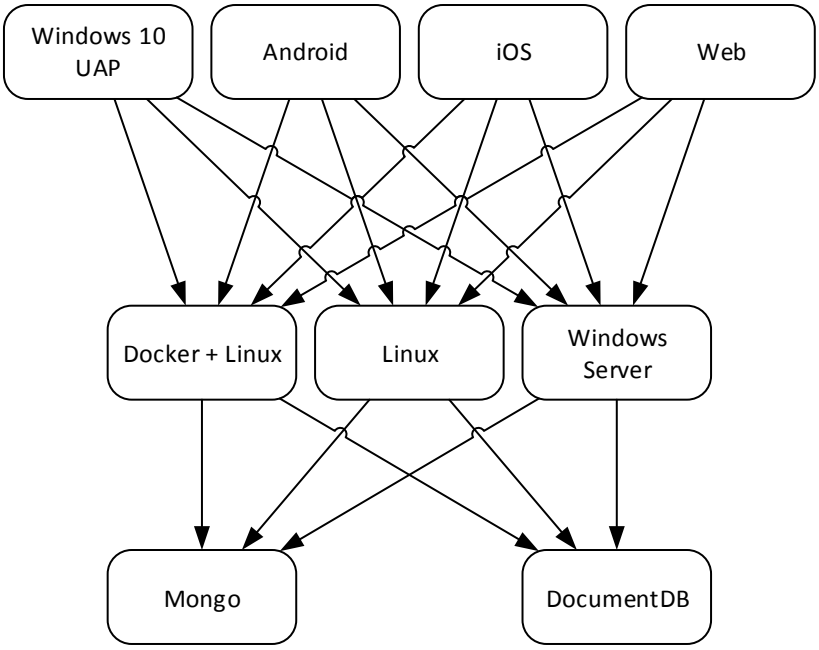
Opinnäytetyön toimeksiantajana toimii suomalainen yritys nimeltä Eitbit Oy, joka on erikoistunut tuottamaan .NET-pohjaisia sovelluksia, taustajärjestelmiä sekä integraatioratkaisuja. Tavoitteena on tutustua siihen, millaista työtä arkkitehtuurin kerroksien, eli front-endin, back-endin tai tietokantojen, siirtäminen toiselle laitealustalle vaatii. Samalla tutkitaan erilaisia tapoja, niin vanhoja kuin uusiakin tekniikoita, joita voidaan käyttää opinnäytetyötä varten kehitetyssä intranet-

sovelluksessa (Ks. Kuvio 1.). Kyseisessä sovelluksessa pyritään rakentamaan kaikki kolme arkkitehtuurin kerrosta siirrettäväksi käyttäen uusia tulokkaita, joilla Microsoft-ympäristön siirrettävyyttä on pyritty kehittämään. Tällaisia tekniikoita on muun muassa TypeScript, Angular 2, Ionic 2 ja ASP.NET Core.



Kuvio 1. Opinnäytetyön sovelluksen arkkitehtuurimalli opinnäytetyön alkuvaiheessa (Poikolainen 2016).

Opinnäytetyö rajataan muutaman uuden tekniikan tutkimiseen sekä yksinkertaisen sovelluksen toteuttamiseen, jossa näitä tekniikoita tullaan soveltamaan. Sovelluksessa voidaan AD- eli Active Directory -kirjautumisen jälkeen lukea intranetin uutisia, jotka on integraatioiden avulla tuotu Mongo- ja DocumentDB-tietokantoihin. Sovelluksessa on seuraavat näytöt: kirjautumis-, uutislista-, luku- ja asetusnäytöt. Integraatiot eivät kuulu toteutettavan sovelluksen piiriin vaan toimeksiantaja hoitaa sen puolen. Sovelluksen todellisen käyttötilanteen helpompaa testausta varten tulee asetuksista pystyä valitsemaan kumpaa kahdesta tietokannasta ja mitä kolmesta palvelimesta käytetään (Ks. Kuvio 2.). Sovelluksen päätarkoituksena on toimia helpomman siirrettävyyden toteuttamisessa ja testaamisessa.



Kuvio 2. Sovelluksen siirrettävyyden kombinaatiot (Poikolainen 2016).

Työympäristönä toimii Microsoftin teknologiat. Sovelluksen toteuttamisessa käytetään .NET Corea ja Visual Studiota. Front-end puoli tehdään käyttäen Angular 2, Ionic 2 ja Typescript–tekniikoita ja tietokantapuolella käytetään Mongo- ja DocumentDB-tietokantoja. Kirjautuminen hoidetaan Active Directorya käyttäen. Front-end -puoli ohjelmoidaan toimimaan seuraavilla alustoilla: Windows, Android, iOS ja internet. Palvelinpuolen tulee taas toimia Linux-, Windows Server- ja Docker-alustoilla.

2.2 Tutkimusmenetelmät ja –kysymykset

Tutkimusmenetelmät

Opinnäytetyön tutkimusmenetelmänä on kehittämistutkimus. Kehittämistutkimuksessa yhdistyvät sekä kvalitatiivinen että kvantitatiivinen tutkimus. Tässä opinnäytetyössä keskitytään kuitenkin kehittämistutkimuksen

kvalitatiiviseen puoleen, koska kyseisessä menetelmässä pyritään saamaan aikaan muutoksia. (Kananen 2015, 39-40)

Kvalitatiivisen tutkimuksen prosessi on joustava ja tietoa tarvittavasta aineistomäärästä tulee sitä myöten, kun ilmiötä saadaan purettua ja ymmärrettyä paremmin (Kananen 2015, 35). Itse kehittämistutkimuksessakin tähtäimenä on muutos. Kyseinen tutkimusmenetelmä sopii opinnäytetyön toteuttamiseen, koska siinä luodaan sovellus, jossa pyritään löytämään parhaimmat siirrettävyysmenetelmät eli toisin sanoen kehittämään kartoitettua alkutilannetta. Kehittämistutkimuksessa pyritään myös poistamaan jokin ongelma tai ainakin edetä kohti parempaa olotilaa. (Kananen 2015, 39-40.)

Tutkimuskysymykset

Opinnäytetyössä pyritään vastaamaan seuraaviin tutkimuskysymyksiin ja alakysymyksiin:

1. Mitä tekniikoita käytetään Microsoft-ympäristössä, kun halutaan tehdä ASP.NET-sovelluksista siirrettäviä?
2. Miten siirrettävyys toteutetaan opinnäytetyön sovelluksessa?
3. Kuinka hyvin toteutettu siirrettävyys toimii opinnäytetyön sovelluksessa?
   3.1. Kuinka helposti toteutettu siirrettävyys on sovellettavissa?

3 Siirrettävyys ja tekniikat

3.1 Siirrettävyys ja järjestelmäriippumattomuus

Tämän opinnäytetyön tavoitteena on tutkia ASP.NET-sovelluksen siirrettävyyttä. Siirrettävyys-käsitteellä on monta määritelmää. Pohjimmiltaan se kuitenkin tarkoittaa olemassa olevan sovelluksen muokkaamista siten, että siitä saadaan uudessa ympäristössä toimiva versio. Yksi käsitemääritelmä lisää vaatimukseksi myös

sen, että siirrettävyyden toteutuksen tulisi olla halvempaa kuin kokonaan uuden sovelluksen luominen. (Mooney n.d., 1.) Siirrettävyydestä on tullut nykyaikana olennaisempi ominaisuus, sillä ihmiset käyttävät elämässään yhä enemmän teknologiaa ja eri laitealustoja. On tärkeää, että sovellukset toimivat eri alustoilla. (Dawson 2011, luku 4.)

Siirrettävyyteen liittyy myös termi järjestelmäriippumaton. Yhden määritelmän mukaan sovellus on järjestelmäriippumaton, kun se toimii useammalla kuin yhdellä alustalla lähes identtisellä tavalla. Alustalla voidaan tarkoittaa esimerkiksi käyttöjärjestelmää, prosessorityyppiä tai laitteistoryhmän tyyppiä. (Cross-platform Definition 2005.)

3.2 Tekniikat

Koska opinnäytetyötä täytyi rajata pienemmäksi, keskitytään tässä työssä pääasiassa muutamaan uuteen tekniikkaan, joita käytetään siirrettävän .NET-sovelluksen luomisessa ja joita opinnäytetyön toimeksiantaja halusi tutkittavan. Näitä tekniikoita on Angular 2, Ionic 2, .NET Core ja TypeScript. Työssä halutaan myös tuoda hieman esille .NET-sovellusten siirrettävyyden alkutilannetta, eli kuinka siirrettävyys on toteutettu ennen viimeisimpien tekniikoiden tuloa markkinoille. Tästä esimerkkinä esitellään lyhyesti kehitysalusta nimeltä Mono.

3.2.1 Mono

Mono on avoimen lähdekoodin MIT-lisenssiä käyttävä kehitysalusta, joka perustuu .NET-sovelluskehykseen. Sen avulla ohjelmistokehittäjät voivat tehdä järjestelmäriippumattomia sovelluksia. Mono sisältää sekä kehittäjän työkalut että tarvittavan infrastruktuurin .NET-asiakasohjelmien ja palvelinohjelmien pyörittämiseen. Se on rakennettu ECMA-standardin Common Language Infrastructure -toteutuksen päälle sekä C#-ohjelmointikieltä käyttäen. Alunperin

Mono-alustaa tuki Novell ja Xamarin, mutta nykyään sitä tukee myös Microsoft ja .NET-säätiö. (About Mono 2017.) Viimeisin Mono-versio on 16. marraskuuta 2016 julkaistu Mono 4.6.2 (Mono Releases 2017).

Mono-kehitysalusta koostuu neljästä eri komponentista: Monon C#-kääntäjästä, Mono Runtime-komponentista, .NET-viitekehyksen luokkakirjastosta, jonka luokat ovat nimensä mukaan yhteensopivia .NET-luokkien kanssa sekä Mono-luokkakirjastosta, joka tarjoaa enemmän mahdollisuuksia kuin Microsoftin luokkakirjastot (About Mono 2017).

Mono on kehitysalustana käytännöllinen useiden eri ominaisuuksien ja toimintojen ansiosta. Se on saavuttanut suosiota .NET-alustan avulla. Mono on helppo oppia. C#-ohjelmoijien suuren määrän takia netissä on paljon ohjeita ja tutoriaaleja, jotka auttavat sovelluskehittäjiä ratkaisemaan monimutkaisiakin ongelmia. Ohjelmointia helpottaa myös Mono:n laaja luokkakirjasto, joka helpottaa tiettyjen koodiosioiden luomista. Mono toimii järjestelmäriippumattomuutensa ansiosta useilla laitealustoilla, kuten Linux-, Windows-, Mac OS X-, iPhone- ja Android-alustoilla. Mono-alustan CLR eli Common Language Runtime antaa sovelluskehittäjän valita mieluisimman ohjelmointikielen työstettäväksi. Saatavilla olevia ohjelmointikieliä on muun muassa C#, VB, Java, Python ja Ruby. (About Mono 2017.)

Mono-alustaa on siirretty jo useaan eri arkkitehtuuriin ja vain pieni määrä kirjoitettua koodia tarvitaan sen siirtämiseen. Tällä hetkellä Mono koostuu noin 200 000 rivistä koodia ja se vaatii vain noin 5000 riviä C-koodia siirrettävyyden toteutumiseen. Mono:n dokumentointisivuilta löytyy hyviä ohjeita siirrettävyyteen liittyen. (Porting 2017.)

3.2.2 .NET Core

.NET Core on Microsoftin tukema avoimen lähdekoodin järjestelmäriippumaton kehitysalusta, jonka Microsoft lisäsi .NET perheeseen vuonna 2014. Järjestelmäriippumattomuutensa ansiosta se toimii Windowsin lisäksi myös MacOs-alustalla ja usealla Linuxin jakelulla. Se on lisäksi taipuvainen kehitysalusta, jolla on kattavat komentorivityökalut ja joka on yhteensopiva .NET Frameworkin, Xamarinin ja Monon kanssa.

.NET Core rakentuu muutamasta eri osiosta. Sen pääosat ovat .NET Runtime, kokoelma ohjelmistokehyksen kirjastoja, SDK-työkalut, kielien kääntäjät sekä .NET Core -sovelluksien käynnistämiseen tarvittava “host” eli isäntä. Kielenään se käyttää tällä hetkellä C#- ja F#-kieliä sekä pian myös Visual Basicia. .NET Core on tavallaan järjestelmäriippumaton versio .NET Frameworkista. Koska se on .NET-toteutus, se toteuttaa .NET Standard Library -spesifikaation. .NET Core tarjoaa joukon ohjelmointirajapintoja, jotka ovat myös saatavilla .NET Frameworkissa, Monossa ja Xamarinissa. .NET Coren voi hankkia kahdella eri tapaa: lataamalla NuGet-pakettimanagerin kautta tai yksittäisinä distribuutioina Microsoftin sivuilta. (Carter, Dykstra, Lander & Onderka 2016.)

.NET Coren malli sekä arkkitehtuuri koostuvat pohjimmiltaan moduuleista. Runtime, kirjasto ja kääntäjät ovat kaikki itsenäisiä kokonaisuuksia, jotka kommunikoivat toistensa kanssa rajapintojen välityksellä. Tämän ansiosta .NET Coressa pystytään käyttämään tai olla käyttämättä haluamiaan komponentteja ja kirjastoja. (Carter & Knezevic 2016.)

.NET Core sekä Mono ovat molemmat järjestelmäriippumattomia sekä avoimen lähdekoodin kehitysalustoja, mutta ovat kuitenkin erilaisia. Mono tukee tiettyjä .NET Frameworkin sovellusmalleja, kuten Windows Formia, toisin kuin .NET Core. Se tukee myös suurta määrää .NET Frameworkin ohjelmointirajapintoja ja toimii useammalla

laitealustalla. Ne kuitenkin keskittyvät eri asioihin: Mono keskittyy mobiilialustoille kun taas .NET Core keskittyy pilveen. (Carter, Dykstra, Lander & Onderka 2016.)

3.2.3 AngularJS ja Angular 2

AngularJS

AngularJS on JavaScript-kieltä käyttävä ohjelmistokehys. Sitä käytetään yhdessä HTML-kielen kanssa tekemään nettisovelluksista dynaamisempia. AngularJS laajentaa HTML-syntaksia direktiivien avulla, jotka tekevät ohjelman komponenteista selvempiä ja ytimekkäämpiä. Kyseinen ohjelmistokehys on ihanteellinen erilaisille palvelinpuolen teknologioille, koska tuotettu koodi suoritetaan kokonaan selaimen puolella. (Developer Guide: Introduction n.d.) AngularJS on nykyään Googlen ylläpitämä, mutta alunperin sen kehittivät Misko Hevery ja Adam Abrons vuonna 2009 (Angular JS - Overview n.d.).

AngularJS tarjoaa sovelluskehittäjälle monia mahdollisuuksia. Sillä voi rakentaa suuria ja helposti hallittavissa olevia sovelluksia vaivattomasti sekä pienellä määrällä kirjoitettua koodia. Muita positiivisia puolia ovat muun muassa seuraavat: HTML-koodin puolella voidaan käyttää datakytkentää, koodia pystytään yksikkötestaamaan eli testaamaan koodia osissa, siinä käytetään riippuvuuksien injektointia (dependency injection) sekä uudelleenkäytettäviä komponentteja ja sovelluksen näkymät kirjoitetaan puhtaasti HTML-koodia käyttäen. (Angular JS - Overview n.d.)

Angular 2

AngularJS ei kuitenkaan ole uusin versio vaan tuorein sekä lopullisin versio uudesta Angular 2-kehyksestä julkaistiin 15. syyskuuta 2016 (Kremer 2016). Angular 2 on uudelleenkirjoitettu versio AngularJS:tä eikä pelkkä päivitys. Se on kirjoitettu kokonaan käyttäen TypeScript-kieltä ja se on huomattavasti nopeampi kuin edeltäjänsä. Angular 2 on mobiilipainoitteinen sekä se tarjoaa enemmän valinnanvaraa kielissä, sillä sitä voi käyttää ECMAScript5-, ECMAScript6-, TypeScript-

ja Dart-kielellä. Se toteuttaa web standardit komponenttien tavoin ja on jättänyt kontrollerien käytön kokonaan pois, sillä ne sekä $scope on korvattu komponenteilla. Angular 2:n arkkitehtuuri on siis komponenttipohjainen ja se on suuri etu verrattuna edelliseen AngularJS-kehykseen. (Sharif 2016.) Komponentit, tarkalleen web-komponentit, ovat web-ohjelmointirajapintoja, joiden avulla voidaan luoda kustomoituja, uudelleenkäytettäviä ja tiivistettyjä HTML-merkkejä eli ”tageja” käytettäväksi nettisivuilla sekä web-sovelluksissa (Introduction: What are web components? n.d.).

- **Angular 1.x Controller**

JavaScript 6 lines

```
var myApp = angular
.module("myModule", [])
.controller("productController", function($scope) {
        var prods = { name: "Prod1", quantity: 1 };
        $scope.products = prods;
});
```

- **Angular 2 Components using TypeScript**

TypeScript 10 lines

```
import { Component } from '@angular/core';

@Component({
    selector: 'prodsdata',
    template: '<h3>{{prods.name}}</h3>'
})

export class ProductComponent {
    prods = {name: 'Prod1', quantity: 1 };
}
```

Angularjs 2 Component

Kuvio 3. Angular 2:n vertailua Angular 1.x-versioon liittyen kontrolleriin ja sen korvaamiseen (Sharif 2016).

Angular 2:n myötä on tapahtunut useita muita muutoksia verrattuna Angular 1.x -versioon. Angular 2 käyttää HTML DOM-mallin tapahtumia (event) sekä elementtien ominaisuuksia suoraan. Ennen tämä on tapahtunut sisäänrakennettuja ng-direktiivejä käyttäen. Yhdensuuntainen sekä kahdensuuntainen datakytkentä on kokenut päivityksen myötä pieniä muutoksia. Syntaksimuutoksia on tapahtunut myös paikallisille muuttujille, jotka määritetään nykyään #-etuliitteellä, sekä rakenteellisille direktiiveille, joista esimerkkinä ng-repeat on korvattu ngFor. Sisäänrakennetuissa direktiiveissä käytetään nykyään myös camelCase-tapaa. Näiden monien muutoksien myötä siirtyminen Angular 1.x -versioista Angular 2:n pariin voi olla hieman hankalaa, mutta sen arvoista. (Sharif 2016.)

Angular 2 mahdollistaa sovelluksen kaltaisten toteutuksien luomisen sekä natiivien sovellusten tekemisen Ionicin, NativeScriptin sekä React Nativen kaltaisia kehyksiä käyttäen. Sen avulla voidaan myös luoda eri käyttöjärjestelmille tietokonesovelluksia, jotka käyttävät käyttöjärjestelmien omia ohjelmointirajapintojaan (API). Nopeutta Angular 2:ssa lisää uusi reitityskomponentti, joka lataa vain sillä hetkellä tarvittavat koodit. Ohjelmointiympäristössä sovelluskehittäjää auttavat heti näkyvät virheviestit, muun palautteen antaminen sekä “ennustava syöttö” eli automaattinen täydennys koodia kirjoittaessa. (Angular: Features and Benefits n.d.)

3.2.4 Ionic Framework, Cordova ja Ionic 2

Ionic Framework ja Apache Cordova

Ionic Framework on avoimen lähdekoodin SDK (Software Development Kit), jonka Ben Sperry, Adam Bradley ja Max Lynch ovat alunperin luoneet. Ionic huolehtii pääasiallisesti sovelluksen ulkonäöstä sekä sen saumattomasta vuorovaikutuksesta käyttöliittymän kanssa. Ionic yksinkertaistaa front-endin luomisprosessia toimimalla ohjelmistokehysten kanssa yhdessä. Ionicin avulla sovelluskehittäjä pystyy luomaan korkealaatuisia mobiilisovelluksia käyttäen tuttuja web-teknologioita kuten HTML-, CSS- ja JavaScript-kieliä. Ionic tarvitsee kuitenkin AngularJS-ohjelmistokehyksen hyödyntääkseen täyttä potentiaaliaan, mutta sen CSS-osaa pystytään käyttämään

ilman Angularia. Tällöin kuitenkin moni ominaisuus jää puuttumaan, kuten vahvat interaktiot käyttöliittymän kanssa sekä animaatiot. (Core Concepts n.d.) Nykyään Ionic on maailman suosituin järjestelmäriippumattomien mobiilisovellusten luontiin tarkoitettu työkalupakki (All about Ionic n.d.).

Ionic Frameworkia voidaan käyttää komentorivin kaltaisella CLI-työkalulla (lyhenne sanoista Command Line Interface), joka sisältää paljon käyttäjälle hyödyllisiä komentoja. Muita ominaisuuksia Ionic Frameworkilla on muun muassa uudelleenkäytettävät komponentit sekä “theming”. Ionicin komponentit toimivat sovelluksen rakennuspalikoina, jotka sopeutuvat niihin alustoihin, joilla sovellus pyörii. Ne on tehty käyttäen HTML-, CSS- sekä joskus JavaScript-teknologioita. Ominaisuudella nimeltä “theming” viitataan siihen, että teemat vaihtuvat sovelluksessa sen mukaan mitä laitealustaa milloinkin käytetään. Tämä tekee sovelluksesta käyttäjäystävällisen, koska teemat luovat tutun näköisen näkymän joka sopii esimerkiksi Android- tai iOS-ympäristöön. (Core Concepts n.d.) Koska Ionic on HTML-ohjelmistokehys, se tarvitsee avukseen Apache Cordovan toimiakseen natiivina sovelluksena (Chapter 1: All about Ionic). Cordova on eräänlainen päällys (englanniksi native wrapper) jonka Ionic tarvitsee ladatakseen internetnäkymään HTML-, JavaScript- ja CSS-osiot silloin kun mobiilisovellus käynnistyy. Cordova tarjoaa myös lisäosia, jotka antavat sovelluksen kommunikoida JavaScript-kieltä käyttäen mobiililaitteen omien toimintojen kanssa. Näitä toimintoja on esimerkiksi kamera ja yhteystietolista. (Biharisingh 2016.)

Ionic 2

Ionic Frameworkin ensimmäisen version jälkeen julkaistiin Ionic 2, joka on rakennettu Angular 2 -ohjelmistokehyksen päälle (Migration Concepts n.d.). 25. tammikuuta 2017 julkaistiin kehyksen viimeisin versio, Ionic 2.0.0, joka on vakaa versio Ionic 2 -kehyksestä. Uuden version kehitys sai alkunsa vuonna 2015, kun JavaScript-kieltä alettiin kehittämään.Tällöin Angular-ohjelmistokehyksen tehnyt tiimi otti yhteyttä Ionic-tiimiin tarkoituksenaan kysyä mahdollisuudesta kehittää Ionic-kehystä uuden Angular 2.0 -version päälle. Ionic-kehyksen kehitys oli hankalaa, koska Angular 2 -

kehyksen saaminen tarpeeksi vakaaseen pisteeseen kesti yllättävän kauan Angular-tiimin kohtaamien vaikeuksien vuoksi. Lopulta he kuitenkin pääsivät haluttuun lopputulokseen. Ionic 2 -kehyksessä on useita uusia ominaisuuksia verrattuna edellisiin versioihin. Näitä ominaisuuksia käydään seuraavaksi läpi lyhyesti.

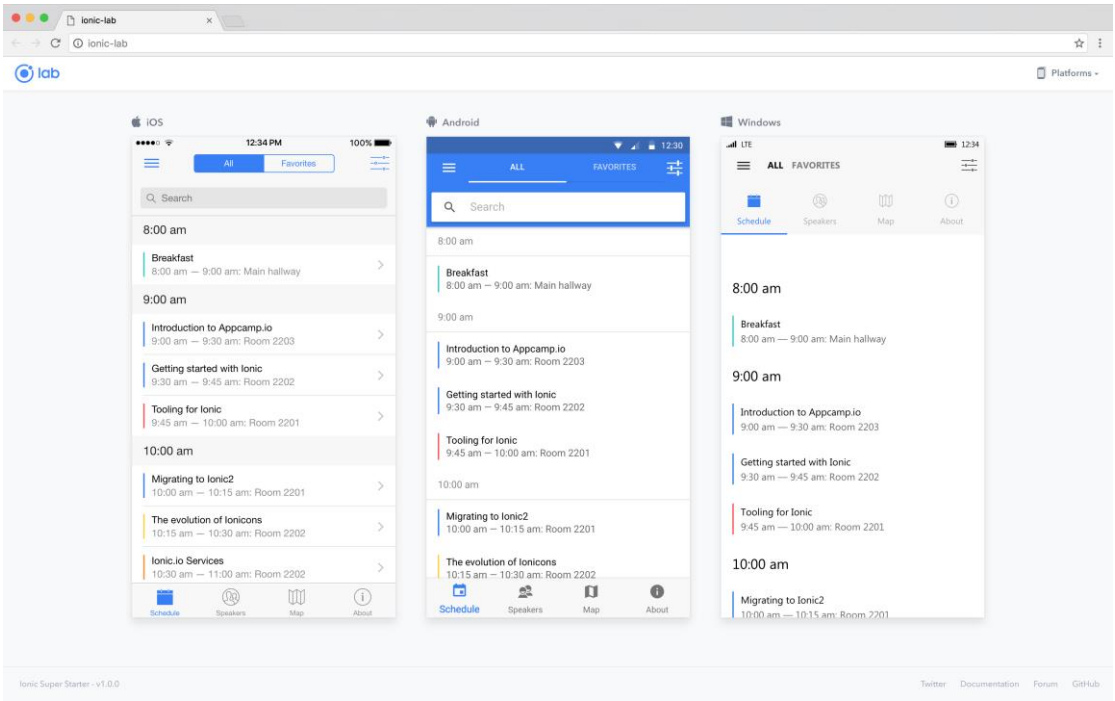
Ionic 2 sisältää enemmän järjestelmäriippumattomien sovelluksien rakentamiseen käytettäviä komponentteja ja muun muassa enemmän modaaleja, valikkoja ja kokonaan uudenlaisen navigointisysteemin. Ionic Native -liitännäissysteemi on myös uusi lisäosa, joka sisältää 70 uutta web API:n lailla käytettäviä ominaisuuksia.

Sovelluksen ulkonäön muokkaaminen eli "theming" on kokenut myös muutoksia. Siitä on tehty helpompaa ja haluamansa tuloksen saa nykyään vähemmällä määrällä kirjoitettua koodia kuin ennen. Tämä sovelluksen ulkonäön muokkaaminen on osa Ionicin Platform Continuity -ominaisuutta, jonka avulla sovellukset sopivat ulkonäöltään ja toiminnoiltaan niihin laitealustoille, joissa sovellus on ladattuna. iOS, Material Design ja Windows ovat Ionic 2:n tukemat.

Ionic-tiimin tavoitteena on pyrkiä parantamaan sovelluksen suorituskykyä ja uudessa Ionicissa sen huomaakin tehostuneen paljon. Siinä ei käytetä enää esimerkiksi JavaScript-rullausta näytöissä, joten sekä Androidin että iOS:n rullaaminen tapahtuu nyt 60 FPS-nopeudella. Uuden Virtual Scroll -toteutuksen avulla pystytään selaamaan suuriakin listoja läpi sen vaikuttamatta negatiivisesti suorituskykyyn. Myös se, että Ionic 2 on rakennettu Angular 2-kehyksen päälle, vaikuttaa suorituskykyyn, sillä Angular 2 on paljon nopeampi kuin sen edeltävä versio.

Ionic 2 -versiossa on myös uudistunut virheenkäsittelytyökalu ja paranneltu Ionic Serve Lab. Ionicin ensimmäisessä versiossa julkaistiin työkaluketju nimeltä App Scripts, joka auttoi sovelluksen rakentamisessa, testaamisessa sekä käynnissä pitämisessä. Nyt Ionic 2 tarjoaa uusia ominaisuuksia virheiden käsittelyyn sekä niiden löytämiseen. Nämä ominaisuudet löytyvät nyt rakennettuna suoraan käyttäjän

luomaan sovellukseen. Esimerkiksi “ionic serve”-komennon avulla virheen yksityiskohtaisemmat tiedot saadaan suoraan selaimen ikkunaan näkyville. Parannellun Ionic Serve Lab -ominaisuuden myötä eri laitealustojen näkymien esikatselu vieretysten selainikkunassa on helpottunut. Yksi Ionic-tiimin tulevaisuudensuunnitelmista on parantaa Ionicin työkaluja ja natiivin SDK:n simuloimista, että ohjelmistokehittäjät pystyisivät luomaan 99% sovelluksestaan selaimen välityksellä. (Lynch 2017.)



Kuvio 4. Ionic Serve Lab, joka näyttää selaimessa eri alustojen näkymiä vierekkäin (Lynch 2017).

3.2.5 TypeScript

TypeScript on Microsoftin luoma kieli, joka on julkaistu käyttäen avoimen lähdekoodin Apache 2.0 –lisenssiä. TypeScript on kirjoitettu pääjoukko JavaScript-koodia, joka käännetään yksinkertaiseksi JavaScript-koodiksi. Tämä tekee projekteista helposti siirrettäviä, koska ne toimivat melkein millä alustalla vain. (Fenton 2014.) TypeScript polveutuu JavaScript-kielestä, jolloin kaikki JavaScript-projektit ovat samalla TypeScript projekteja. TypeScript käyttää JavaScriptin

kirjastoja, joten TypeScript sisältää koko JavaScript-kielen ja vielä lisäksi kokoelman lisätoimintoja. TypeScriptiä käytetään esimerkiksi Angular JS 2.0 -ohjelmistokehyksen kanssa, johon sitä heidän sivuillaan (https://angular.io/) suositellaan. (Hejlsberg, Mohamed, Saeeduddin & Zhong 2016)

Typescriptistä löytyvät piirteet voidaan jakaa kolmeen osaan niiden JavaScript-suhteiden mukaan (Ks. Kuvio 5.). ECMAScript 5 ja 6 -versiot ovat kielimäärittelyjä, jotka ovat sukua JavaScriptin viralliselle määrittelylle, ECMA-262 ECMAScriptille. ECMAScript 5 luo TypeScriptin pohjan sekä sisältää suurimman osan kielen toiminnoista, kun taas ECMAScript 6 sisältää moduulit koodin organisoimiseen ja luokkapohjaiseen olio-orientoitumiseen. Kolmas toimintosetti sisältää asioita, joita ei olla suunniteltu lisättäväksi osaksi ECMAScript-standardia. Tarvittaessa kaikki toiminnot voidaan kääntää ECMAScript 5 –kielimäärittelyksi. (Fenton 2014.)

Kuvio 5. TypeScriptin toimintojen lähteet (Fenton 2014).

Web-kehittäjä Ewerlöf kertoo julkaisemassaan artikkelissa (2016) lyhyesti TypeScriptin kannattavuudesta JavaScriptiin verrattuna. Yksinkertaisessa pankkitili-esimerkissä nähdään, miten luokan luonti eroaa TypeScriptillä ja JavaScriptillä (Ks. Kuvio 6.). Vaikka koodin merkkien määrässä on eroa, vie TypeScript enemmän tilaa kuin JavaScript. Käyttäen ohjelmoinnissa ohjelmistoympäristönä esimerkiksi Visual Studio Codea, TypeScript auttaa antaen älykkäitä vinkkejä, jotka tarjoavat mahdollisia täydennyksiä ja myös auttavat välttämään yleisiä virheitä ilmoittamalla niistä. Tällaista JavaScript ei tällä hetkellä tee. TypeScript voi olla järkevä vaihtoehto

JavaScriptin sijaan silloin, kun projektia työstää useampi kuin yksi henkilö, henkilöt ovat kokeneempia koodaajia, projektin valmistumisella on kiire tai jos projektissa on jo suuri määrä pohjustuskoodia valmiina. (Ewerlöf 2016)

TypeScript

```
class BankAccount {
    balance = 0;
    deposit(credit: number) {
        this.balance += credit;
        return this.balance;
    }
}
```

JavaScript

```
var BankAccount = (function () {
    function BankAccount() {
        this.balance = 0;
    }
    BankAccount.prototype.deposit = function(credit) {
        this.balance += credit;
        return this.balance;
    };
    return BankAccount;
})();
```

Kuvio 6. TypeScriptin eroavaisuus JavaScriptiin käyttäen yksinkertaista pankkitili-esimerkkiä (Hejlsberg, Mohamed, Saeeduddin & Zhong 2016).

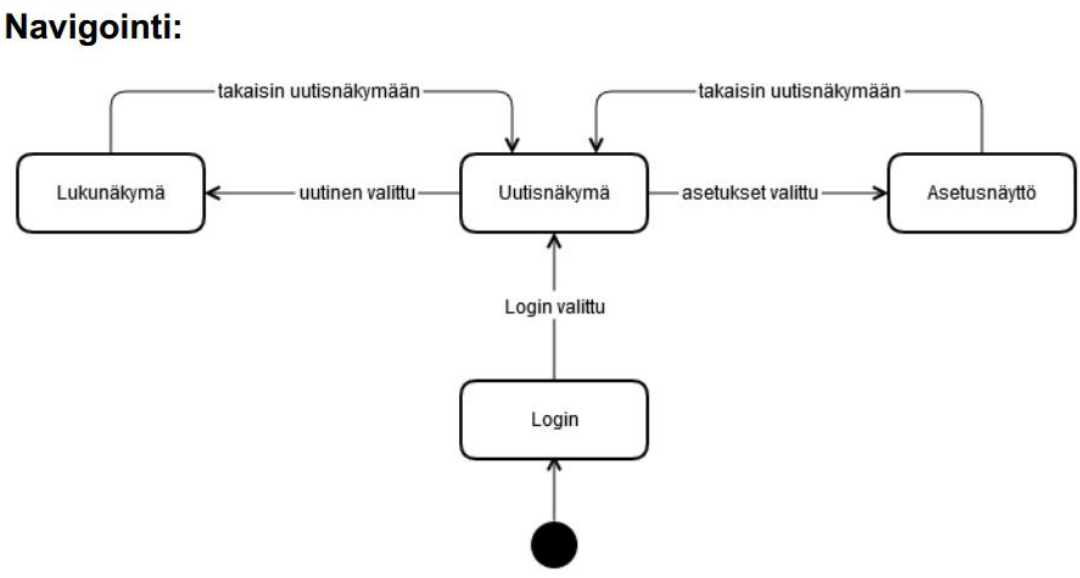
4 Sovellus

4.1 Suunnittelu

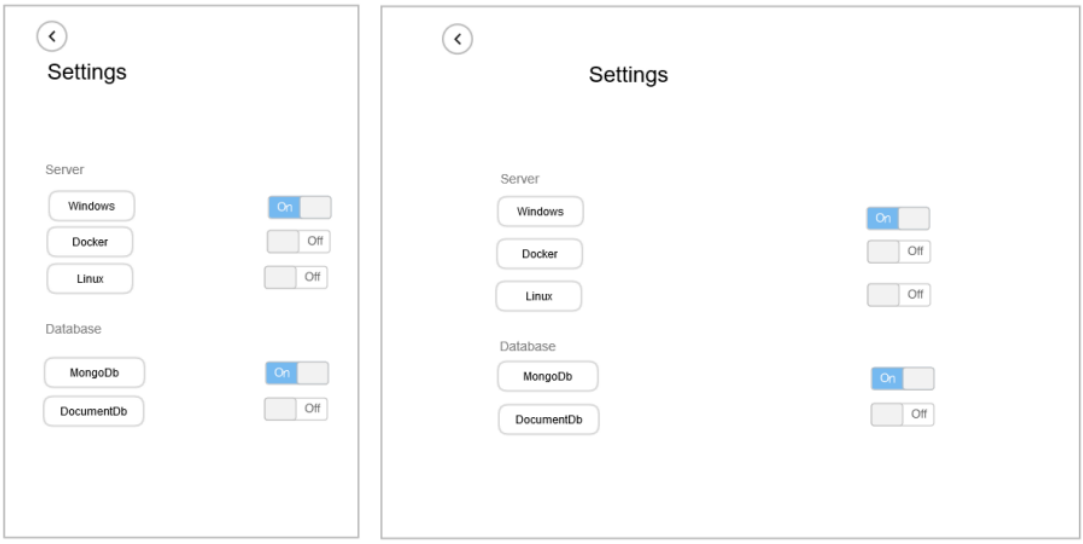
4.1.1 Käyttäjätarina ja käyttöliittymäkuvat

Ennen sovelluksen toteutusta aloitettiin sen suunnittelu. Suunnitteluun sisältyi käyttäjätarinan ja käyttöliittymäkuvien luonti sekä arkkitehtuurin mallintaminen

UML-kaavioiden avulla. Suunnitteluvaihe aloitettiin luomalla käyttäjätarina. Käyttäjätarina on selkeä ja lyhyt kertomus sovelluksen toiminnoista käyttäjän näkökulmasta (Tolvanen 2013). Käyttäjätarina sisälsi kuvaukset neljästä sovelluksen päänäkymästä käyttäjän näkökulmasta: kirjautumis-, uutislista-, luku- sekä asetusnäkymästä. Kirjautumisnäkymässä käyttäjä kirjautuu valmiiksi määritellyillä AD-tunnuksilla sovellukseen. Käyttäjällä on myös mahdollisuus kirjautua ulos. Tästä siirrytään uutislistanäkymään, jossa käyttäjä pystyy selaamaan uutisia ja siirtymään halutessaan valikkoon. Klikkaamalla yksittäistä uutista käyttäjä siirtyy lukunäkymään, jossa hän näkee valitsemansa uutisen kokonaisuudessaan. Tästä hän pystyy palaamaan edelliseen näkymään. Käyttäjä voi mennä valikon kautta asetusnäkymään, jossa hän pystyy valitsemaan mitä kolmesta palvelimesta ja kumpaa kahdesta tietokannasta hän haluaa käyttää. Käyttäjätarinan kirjautumisosuus ei ole kuitenkaan oleellinen, koska toteutetussa sovelluksessa ei ehditty luoda AD-kirjautumista ollenkaan. Käyttöliittymäkuvat sekä navigointimalli luotiin käyttäen käyttäjätarinaa pohjana. Kuvien suuren määrän vuoksi vain muutama on lisätty opinnäytetyöhön esimerkeiksi.



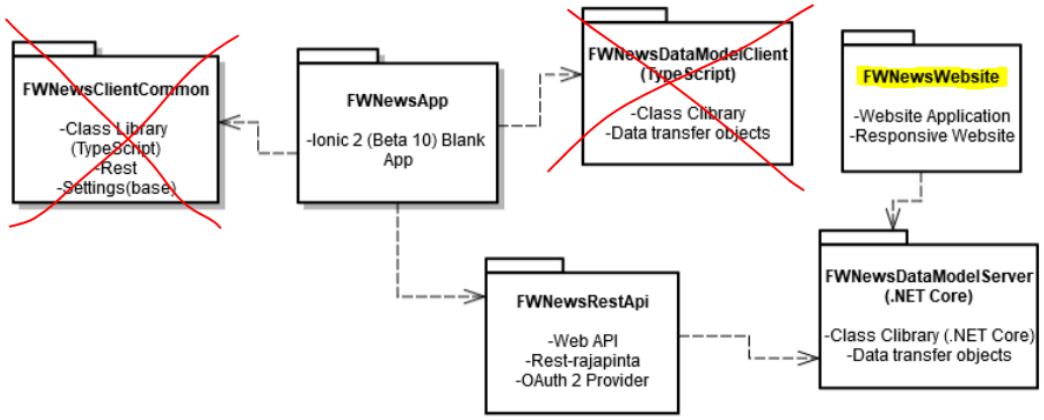
Kuvio 7. Sovelluksen navigoinnista luotu malli.



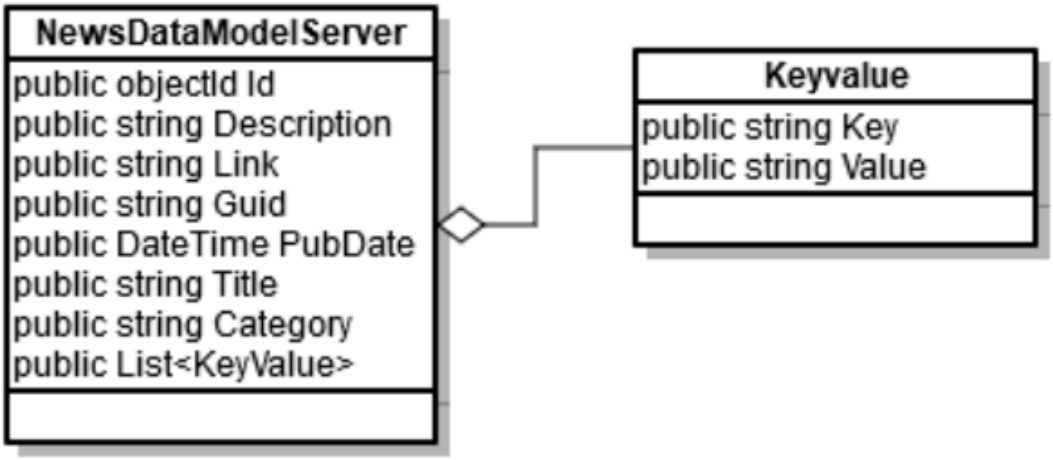
Kuvio 8. Sovelluksen asetussivun web- ja mobiiliversioista luodut käyttöliittymäkuvat.

4.1.2 Luokkakaaviot

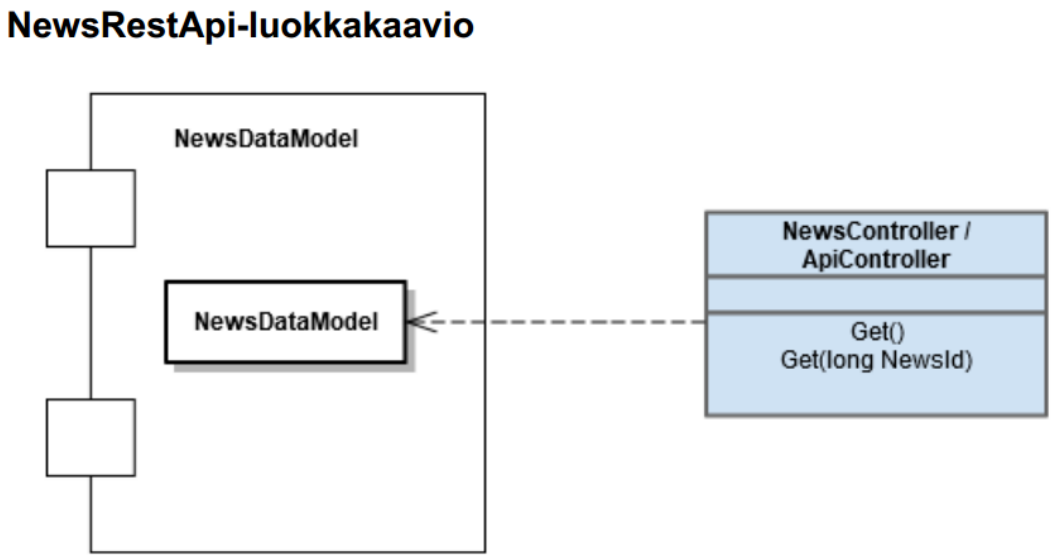
Sovelluksen arkkitehtuurin suunnittelussa käytettiin apuna erilaisia diagrammeja, esimerkiksi UML-malleja. Työnantajan käyttämä organisaatiowikiohjelmisto nimeltä Confluence sisälsi diagrammiliitännäisen, jolla luotiin sekä sovelluksen käyttöliittymäkuvat että luokkakaaviot ja sekvenssikaavio. UML on ohjelmistokuvaustekniikka eli se määrittelee kaaviotyypit, joilla toteutetaan muun muassa ohjelmiston rakenteen kuvaaminen (Luukkainen ja Laine 2010, 7). Arkkitehtuurista tehdyt mallit kokivat radikaaleja muutoksia sovellusta luodessa, joten enää pieni osa niistä toteutuu sovelluksen lopullisessa versiossa. Esimerkiksi FWNewsClientCommon- ja FWNewsDataModelClient-luokat on poistettu ja FWNewsWebsite on korvattu FWNewsAspNetCore-luokalla. Kaikkia muutoksia ei ole korjattu UML-malleihin, mutta pääidea niistä näkyy kyllä (Ks. Kuvio 9.). Myös tehty sekvenssikaavio on tällä hetkellä turha, koska kirjautumista ei luotu. Mallit kuitenkin auttoivat sovelluksen luomisessa selventämällä sovelluksen arkkitehtuurin perusideaa.



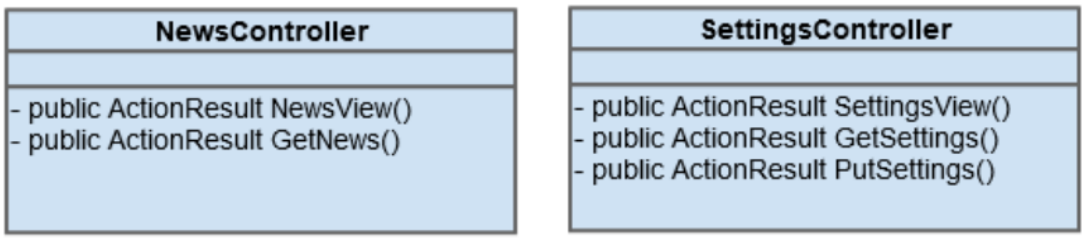
Kuvio 9. Sovelluksen luokka-arkkitehtuurin alkuperäinen mallinnus hieman paranneltuna.



Kuvio 10. DataModelServer-luokkakaavio.



Kuvio 11. NewsRestApi-luokkakaavio.



Kuvio 12. FWNewsWebsite- eli FWNewsAspNetCore-luokkakaavio.

4.2 Toteutus

Opinnäytetyön sovelluksen toteutuksen teoriaosuuteen on kirjoitettu tälle opinnäytetyölle tärkeimmät kohdat, koska muuten dokumentoinnista tulisi liian pitkä. Tässä käydään läpi tietokantojen ja palvelinten siirrettävyys, sovelluksen toteutuksessa ilmenneet ongelmat sekä sen vaikeimmat osuudet.

4.2.1 Tietokannat

Opinnäytetyössä käytettiin testikantoja, joista toinen, MongoDB-kanta, luotiin lokaaliksi Linuxille ja toinen, DocumentDB-kanta, Azuren palvelimelle. Docker- ja Windows-palvelimet voivat kutsua Linuxilla olevaa MongoDB-kantaa. DocumentDB-kanta lisättiin vasta myöhemmässä vaiheessa.

Projektin DataModelServer-kirjastoon tehtiin normaalit kantahaut, jotka mahdollistivat tietokantojen kanssa kommunikoimisen. Siellä asetettiin kannoille myös sopivat nimet, kuten “Description”, “Link”, “Tags” ja niin edelleen.


```
{
        "_id" : ObjectId("57fc82729fe70cd89104a6d5"),
        "description" : "mongodb manual",
        "link" : "https://docs.mongodb.com/manual/",
        "quid" : "https://docs.mongodb.com/manual/",
        "pubdate" : "2016-09-18T14:54:42.9272429Z",
        "title" : "mongodb",
        "category" : "database",
        "tags" : [
                {
                        "key" : "mongodb",
                        "value" : "a1"
                },
                {
                        "key" : "database",
                        "value" : "a2"
                }
        ]
}
```


Kuvio 13. Esimerkki MongoDB-kannan sisällöstä.

4.2.2 Kantojen vaihto: Local storage

Local storage eli eräänlainen paikallinen tiedontallennus luotiin, kun sovellus oli kutakuinkin koossa. Tätä paikallista tiedontallennusvälinettä käytettiin asetusnäkymän ominaisuuksien, palvelimen ja tietokantojen vaihdon, luonnissa. Paikallisen tiedontallennuksen avulla tallennetaan käyttäjän valitsemat vaihtoehdot käyttäjän koneelle. Tätä sovellettiin siten, että pystyttiin vaihtamaan kantoja ja

palvelinta. Ongelmaksi muodostui tosin se, että tietokannan arvoja ei viedä toiselle palvelimelle asetuksia vaihdettaessa.

Tietokannan vaihto tapahtuu if-else -lauseella. Ensin haetaan localStorage.get()-koodilla varastoon tallennettu arvo, joka on asetusnäkymässä oleville tietokannoille. Mobiilipuolen localStorage ei hyväksynyt enum-arvoja, joten ne tuli laittaa string-muodossa. LocalStorageen tallennettu arvo kertoo lähetetäänkö REST-luokkaan vievä osoite MongoDB- vai DocumentDB –tietokannalle, kun se asetetaan palauttamaan arvoa vastaava osoite useUrl()-metodissa. Tämä vastaus taas siirretään REST-kutsuun, jossa pyydetään useUrl()-metodin tuomaa vastausta. Näin oikean kannan tiedot saadaan käyttöliittymälle. Jos arvoa ei ole asetettuna, metodi palauttaa ehtolauseen else-osion.

```
 getmongoUrl = '/api/NewsData/Get/';
 getdocumentUrl = '/api/NewsDocument/Get/';

useUrl() {
    let value = this.localStorage.get('database');
    if (value == 0) return this.getdocumentUrl;
    else return this.getmongoUrl;
}
 getAll(): Observable<News[]> {
     return this.http.get(this.useUrl()).map(response => response.json());
 }
```

Kuvio 14. Tietokannan vaihtaminen Local Storagen avulla.

4.2.3 REST

Sovellukselle luotiin REST-kutsut RestApi-kirjastoon, joka kommunikoi DataModelServer-kirjaston kanssa. Kun sovellus laitetaan päälle, se tekee kutsun RestApi-luokkaan joka taas kutsuu DataModelServer-kirjastoon tehtyä kantakutsua. Tämä palauttaa kannan tiedot sovellukselle ja sitä kautta käyttäjälle käyttöliittymän puolelle. Seuraavissa kuvioissa (Ks. kuviot 15, 16, 17 ja 18) näytetään, miten data kulkee sovelluksen eri osissa.

```
getData() {
    this.newsService.getAll().subscribe(newsLoaded => { this.news = newsLoaded });
}
```

Kuvio 15. TypeScript-luokassa ”home.components” eli sovelluksen listanäkymässä kutsutaan newsService-luokkaa.

```
 getmongoUrl = '/api/NewsData/Get/';
 getdocumentUrl = '/api/NewsDocument/Get/';

useUrl() {
    let value = this.localStorage.get('database');
    if (value == 0) return this.getdocumentUrl;
    else return this.getmongoUrl;
}
 getAll(): Observable<News[]> {
     return this.http.get(this.useUrl()).map(response => response.json());
 }
```

Kuvio 16. Datan käsittely newsService-luokassa, joka palauttaa NewsData-kontrollerin Get-metodin.

```
[Route("api/[controller]")]
public class NewsDataController : Controller
{
    [HttpGet("[action]")]
    public ActionResult Get()
    {
        return Json(NewsRepository.Get());
    }
```

Kuvio 17. NewsDataController-luokka palauttaa JSON-muodossa NewsRepositoryn.

```
public class NewsRepository
{
    public static IMongoCollection<News> GetCollection()
    {
        //käytössä olevan mongodb kannan asettaminen
        var connectionString = "mongodb://40.68.198.173";
        var client = new MongoClient(connectionString);
        var database = client.GetDatabase("news");
        var collection = database.GetCollection<News>("news");
        return collection;
    }

    public static IEnumerable<News> Get()
    {
        try
        {
            var collection = GetCollection();
            return collection.Find(new BsonDocument()).ToListAsync().Result;
        }

        catch (Exception ex)
        {
            //todo lisää lockaus
            throw ex;
        }
    }
```

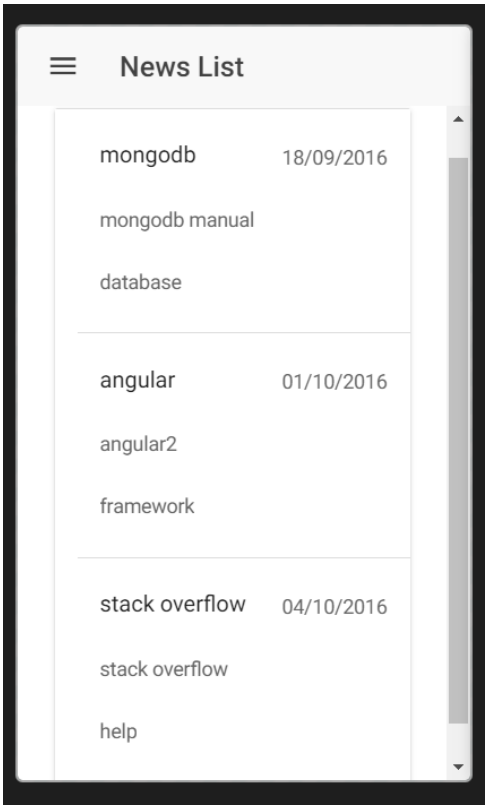
Kuvio 18. NewsRepository-luokka, jossa kommunikoidaan MongoDB-kantojen kanssa.

4.2.4 Mobiilisovellus

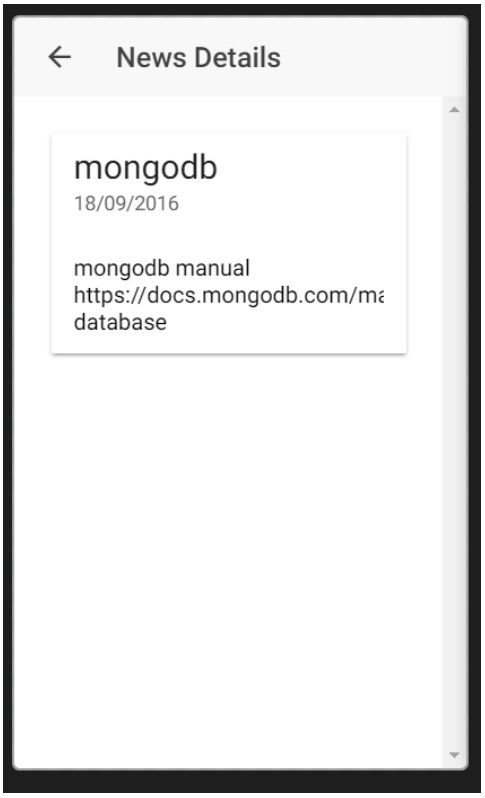
Alunperin sovelluksen mobiilipuoli luotiin käyttämällä Visual Studiossa ”Apache Cordova Ionic 2 RC – Sidemenu” -pohjaa. Sille luotiin välikädeksi RestApi-kirjasto, joka siis kutsuu DataModelServer-kirjaston kantahakuja. Tämän jälkeen luotiin uutislista-, luku-, sekä asetusnäkymät. Uutislistanäkymään (Ks. kuvio 19) tuotiin tietokannasta kaikki tiedot, lukunäkymään (Ks. kuvio 20) tuotiin id:n perusteella tietyn uutisen tiedot ja asetussivulle (Ks. kuvio 21) luotiin ominaisuudet, joilla voitiin vaihtaa tietokantaa ja palvelinta. Asetussivun ominaisuudet lisättiin kuitenkin vasta myöhemmin, koska siirrettävyys-ominaisuudet tehtiin vasta loppupäässä.

Mobiilipuolella tuli vaikeuksia siirtää sovellus web-puolelle ilman suuria muutoksia,

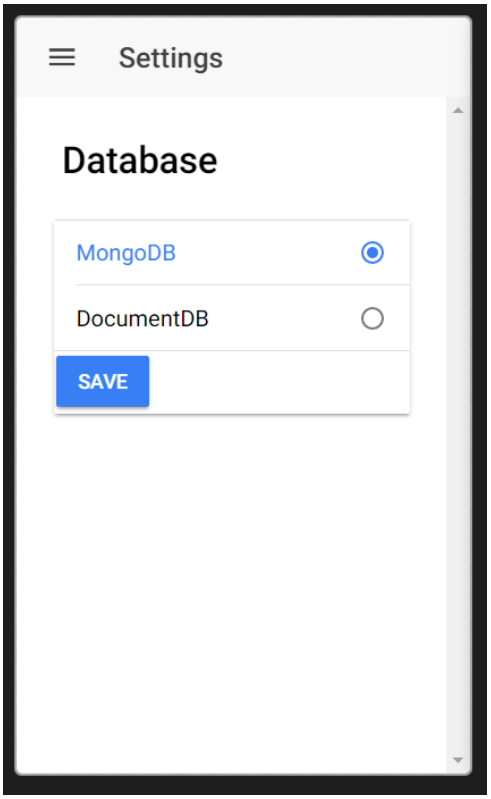
joten oli järkevämpää aloittaa web-sovelluksen luonti uudelta pohjalta. Luodun ASP.NET Core web-sovelluksen manuaalinen muuttaminen Angular 2 -sovellukseksi ei onnistunut, joten luotiin valmis projekti Yeoman-työkalulla. Yeoman on eräänlainen generaattorina toimiva liitännäinen, joka luo projektiin juuri ne osat, joita sovelluskehittäjä tarvitsee (What's Yeoman? n.d.). Opinnäytetyössä olisi voitu käyttää myös Visual Studion ASP.NET Core template -lisäosaa uuden projektin luomisessa, mutta tässä työssä päädyttiin kuitenkin Yeomaniin. Core template olisi luotu Visual Studiolla, kun taas Yeoman-projekti luotiin komentorivin kautta. Yeoman-projektiin siirrettiin kaikki vanhan projektin luokat ja ne toimivat lähestulkoon moitteettomasti uudessa ympäristössä. Tässä vaiheessa sovelluksen arkkitehtuuriin tehtiin kuitenkin suuria muutoksia, sillä uusi projekti ei jostain syystä suostunut ottamaan yhteyttä sovelluksen RestApi- ja ModelClient-osioihin. Tästä syystä ne kopioitiin pääprojektin sisälle.



Kuvio 19. Uutislistanäkymä mobiiliversiossa.



Kuvio 20. Lukunäkymän mobiiliversiossa.



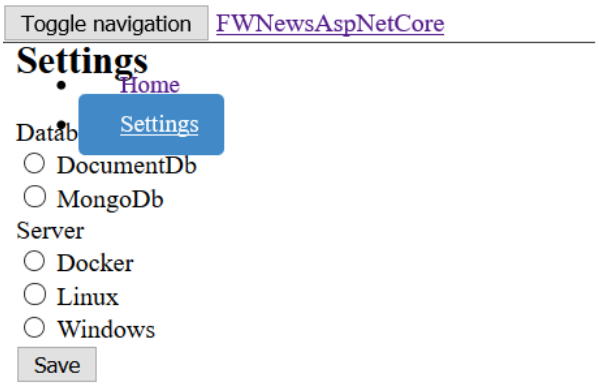
Kuvio 21. Asetusnäkymä mobiiliversiossa.

4.2.5 Linux

Kun kantojen välillä vaihtaminen oli saatu toimimaan sekä Windows-palvelimen web-puolella että mobiilipuolella, alettiin keskittyä Linux-puoleen. Luotu web-sovellus, eli FWNewsAspNetCore, kopioitiin Linuxille käyttäen WinSCP:tä. WinSCP:n päätoiminto on siirtää tiedostoja lokaalin tietokoneenkoneen ja toisen koneen välillä (Introducing WinSCP, 2014). Sovellukseen piti tehdä muutoksia, että se toimisi, joten työssä seurattiin Scott Hanselmanin blogissa julkaistun tutoriaalin (2016) ohjeita.

Ohjeen muiden lisäysten lisäksi sovellukselle tehtiin Nginx-niminen käänteinen välityspalvelin (englanniksi reverse proxy). Käänteinen välityspalvelin sijaitsee yleensä yksityisessä verkossa ja sieltä käsin ohjaa asiakkaan pyynnöt oikeille back end -palvelimille (What is a Reverse Proxy Server? N.d.). Tässä tilanteessa sillä siis saadaan Linux-sovellus (ja myös Docker-sovellus) näkymään verkon yli avatussa selaimessa lokaalilla Windows-koneella. Jostain syystä tämä hävittää Linuxissa sovelluksen muotoilut eikä virhettä pystytty ratkaisemaan. Muotoilun häviämisen takia sovelluksen asetuksien toimivuutta ei voida kokeilla, sillä sivuvalikko estää valintojen tekemisen asetussivulla. Sivuvalikko estää myös yksittäisen uutisen valitsemisen eli siirtymisen lukunäkymään. Linux-sovelluksesta tiedetään siis vain se, että se tuo onnistuneesti testikannan tiedot näkyville uutislistanäkymään.

Kuvio 22. Muotoilut menettänyt uutislistanäkymä Linux-sovelluksen web-versiossa.



Kuvio 23. Muotoilut menettänyt asetusnäkymä Linux-sovelluksen web-versiossa.

4.2.6 Docker

Sovelluksessa piti olla dockerfile Docker-alustaa varten (Ks. kuvio 24). Sen avulla luotiin sovelluksesta Docker-vedos (englanniksi image), joka pystyttiin tehdä Windowsin puolella, johon oli Docker asennettuna. Tämän jälkeen se siirrettiin Docker Hub -palveluun, jonka kautta se pystyttiin vetämään Linuxin puolelle. Opinnäytetyömme sovelluksessa Dockeria piti alunperin käyttää Linuxilla, jonka takia sitä käytettiinkin Windowsilla vain projektin vedoksen luonnissa.

```
FROM microsoft/dotnet:1.1.0-sdk-projectjson

RUN apt-get update
RUN wget -qO- https://deb.nodesource.com/setup_6.x | bash -
RUN apt-get install -y build-essential nodejs

WORKDIR /app
COPY . /app
RUN ["dotnet", "restore"]
RUN ["dotnet", "build"]
EXPOSE 5050/tcp
ENV ASPNETCORE_URLS http://*:5050
ENTRYPOINT ["dotnet", "run"]
```

Kuvio 24. Docker-vedoksessa käytetty Dockerfile.

Myös Dockerille tehtiin käänteinen välityspalvelin. Se tehtiin Dockerin hostiin eli isäntään asennettuun Nginx-alustalle. Isäntänä tässä tapauksessa toimii Linux-

palvelin. Sovelluksesta luotiin toinen Docker-vedos, joka laitettiin ajettaessa kuuntelemaan porttia 4010. Dockerfile-tiedostossa määritetty portti oli alunperin 5050 ennen muutosta. Nginx taas kuuntelee porttia 81, joka ohjattiin kuuntelemaan sovelluksen Docker-konttia. Docker kuuntelee eri porttia kuin Linux, mutta ne käyttävät samaa IP-osoitetta.

```
server {
listen 81;

server_name localhost;

location / {
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header Host $host;
proxy_pass http://127.0.0.1:4041;
}
location /home {
proxy_pass http://127.0.0.1:4041/home;
}
location /counter {
proxy_pass http://127.0.0.1:4041/counter;
}
location /fetch-data {
proxy_pass http://127.0.0.1:4041/fetch-data;
}
}
```

Kuvio 25. Dockerin Nginx-konfigurointi, joka on hieman erilainen kuin Linuxin konfigurointi: Docker ei käytä Linuxin tavoin Supervisoria.

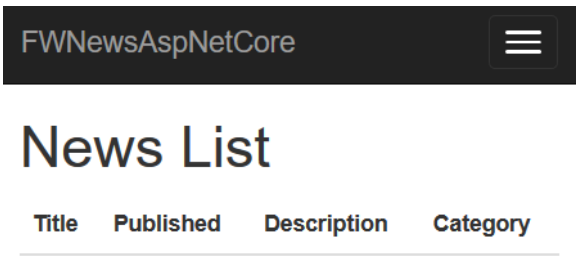
```
        server_name localhost;

        location / {
proxy_pass http://localhost:5123;
proxy_http_version 1.1;
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection keep-alive;
proxy_set_header Host $host;
proxy_cache_bypass $http_upgrade;

                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                try_files $uri $uri/ =404;
        }
location /home {
proxy_pass http://localhost:5123/home;
}
location /counter {
proxy_pass http://localhost:5123/counter;
}
location /fetch-data {
proxy_pass http://localhost:5123/fetch-data;
}
```

Kuvio 26. Linuxin Nginx-konfigurointi.

Dockerin puolella ei tapahtunut muotoilun katoamista välityspalvelimen lisäämisen jälkeen. Tietokanta kuitenkin palauttaa tyhjää ja ilmoittaa virheestä, kun sovelluksen päänäkymä avataan selaimella. Tämä virhe tapahtuu verkon ylityksessä, joka tuntemattomasta syystä estää kannan tietojen välityksen. Kannan tiedot saadaan näkyville terminaalin kautta pelkällä Dockerin IP:llä kokeiltaessa. Linuxin puolella tuli myös virheviesti “internal server error”, jos sovelluksen avasi kirjoittamalla pelkästään IP-osoitteen osoiteriville. Jos osoitteeksi kirjoitti “IP-osoite/home” Linuxilla, ei virhettä tullut. Virheen aiheuttaa luultavasti jokin jälkeenpäin tehty muutos projektissa, koska sitä ei tullut tyhjää pohjaa käyttämällä.

Haettaessa sovelluksen päänäkymää Dockeria käyttäen, ASP.NET toi selaimeen oman virheilmoituksensa. Toinen .css-tiedostoa koskeva virheilmoitus, “Response with status: 200 Ok for URL: null”, tuli terminaalissa. Tämä virhe ei kuitenkaan vaikuttanut sovelluksen toimintaan mitenkään, toisin kuin virhe, joka tuli avaamalla päänäkymän seaimessa. Tämä outo bugi käyttäytyi seuraavasti: kun Dockerin puolella avataan

web-sovelluksen asetusnäkymä osoitteesta ”IP/counter” ja siirtyy sieltä sivumenun kautta päänäkymään, ei tule virhettä ollenkaan. Tämä kuitenkin palauttaa vain päänäkymän otsikon eikä mitään muita tietoja (Ks. kuvio 27). Kun avataan suoraa sovelluksen päänäkymä, ponnahtaa virheilmoitus esille. Dockerilla kuitenkin pystyttiin testaamaan asetuksia. Koodin senhetkisen sisällön perusteella palvelinten välinen vaihto näyttää toimivan.



Kuvio 27. Dockerin web-sovelluksen tyhjä uutislistanäkymä.

4.2.7 Palvelimen vaihto: Local storage

Palvelinpään vaihtaminen tapahtuu samankaltaisella if-else-lauseella kuten kannan vaihtaminen. Kuvassa (Ks. kuvio 28) oleva toiminto suoritetaan, kun klikataan tallenna-nappia asetussivulla. OnSave()-toiminnossa tallennetaan local storageen arvot kannalle ja palvelimelle, jotka valittiin käyttöliittymässä olevilla radionapeilla. Se palauttaa serv()-metodin, joka valikoi palvelimen if-else-lausetta käyttäen.

```
serv() {
 let value = this.localStorage.get('server');
        if (value == 0) return window.location.href='http://40.68.198.173:81/home';
        else if (value == 1) return window.location.href='http://40.68.198.173/home';
        else return window.location.href='http://localhost:62499';
}

  onSave(s:Settings) {
    this.localStorage.set('database', s.database);
    this.localStorage.set('server', s.server);
return this.serv();
  }
```

Kuvio 28. Palvelimen vaihtaminen Local Storagen avulla.

5 Tulokset ja niiden arviointi

Opinnäytetyön tuloksena syntyi teoriaosuus Microsoft-ympäristössä käytettävistä tekniikoista sekä yksinkertainen siirrettävä sovellus. Teoriaosuudesta saa suurpiirteisen käsityksen käytetyistä tekniikoista ja sovellus täyttää osittain siirrettävyyden vaatimukset. Sovelluksen rakenne muuttui työn aikana useasti: luokkakaavioista jouduttiin poikkeamaan virheiden takia sekä osittain käytännöllisistä syistä. Myös AD-kirjautuminen jouduttiin ajan puutteen vuoksi jättämään toteuttamatta.

Siirrettävyys Microsoft-ympäristössä on todella saamassa uutta tuulta purjeisiinsa. Monon rinnalle tullut .NET Core ei korvaa Monoa eikä ole sama asia kuin .NET Framework, vaan se tukee sekä sitä että Monoa ja on tavallaan järjestelmäriippumaton versio .NET Frameworkista. .NET Coren kanssa käytettävät Angular 2 - ja Ionic 2 -tekniikat toimivat hyvin yhteen, sillä Ionic 2 on rakennettu Angular 2:n päälle. Angular 2 on komponenttipohjainen, mikä tekee sovellusten luomisesta yksinkertaisempaa ja tekee niistä nopeampia. Ionic 2 taas sisältää monia uusia komponentteja ja ominaisuuksia, jotka auttavat järjestelmäriippumattomien sovellusten luonnissa ja huolehtii front endin saumattomasta vuorovaikutuksesta käyttöliittymän kanssa. Viimeinen, muttei vähäisin, opinnäytetyössä esitelty tekniikka on TypeScript-kieli, joka on tiivistetty joukko JavaScript-koodia. Tämän takia se toimii monella eri alustalla. Nämä uudet tekniikat eivät välttämättä ole täydellisiä, mutta ne

vievät Microsoft-ympäristöä uuteen suuntaan siirrettävyyden luomisessa sekä sen mahdollisessa helpottamisessa.

Siirrettävyyden toteutus opinnäytetyön sovelluksessa kuvataan suhteellisen hyvin. Sovelluksesta luodut arkkitehtuurimallit antavat hyvän käsityksen siitä, mitä ollaan rakentamassa, vaikka arkkitehtuuri muuttuikin työn aikana. Luodussa sovelluksessa käytetään jokaista teoriaosassa läpikäytyä tekniikkaa, mutta myös muita, joista kerrotaan sovelluksen toteutus-osiossa. Apuna käytetään esimerkiksi Nginx-välityspalvelinta sekä local storagea eli eräänlaista tiedontallennusvälinettä.

Siirrettävyys toteutuu osittain opinnäytetyön sovelluksessa. Arkkitehtuurin jokainen osa toimii oikein sulavasti Windows-palvelimella eikä sen toteuttaminen tuottanut sen suurempia ongelmia. Linux- ja Docker- alustoilla ongelmia taas löytyi. Linux-palvelimella pyöriessään sovelluksen muotoilu katosi kokonaan. Tämä aiheutti sen, että sivuvalikko esti asetussivun käytön, eikä siirrettävyyden toteutumista voitu testata. Docker-alustalla koettiin myös vastoinkäymisiä, joista suurimmaksi koitui virheilmoitusten ilmaantuminen, kuten Linux-palvelimen puolella, ja tyhjän uutislistan palauttaminen. Dockerilla pystyttiin kuitenkin testaamaan siirrettävyyden toteutus: se onnistui moitteettomasti. Vaikka luodun sovelluksen tulokset eivät olleet täysin sitä mitä odotettiin, suoriutui se suurimmaksi osaksi siirrettävyydessä hyvin.

Sovelluksen yksinkertaisuuden ansiosta jotkut osiot ovat suhteellisen helposti sovellettavissa, esimerkiksi REST-puolen perustoiminta on hyvin kuvattu. Sovellettaessa näitä tuloksia on kuitenkin oltava tarkka ja valikoiva, sillä jotkin tämän työn sovelluksen ongelmat saattavat tulla dokumentointia seuraamalla. Siirrettävyyden sovellettavuutta haittaa dokumentoinnin vajavaisuus opinnäytetyössä. Jos se olisi yksityiskohtaisempaa, se tekisi siirrettävyydestä helpommin toteutettavaa lukijalle. Dokumentoinnista ei tullut niin selkeää muun muassa siksi, koska sovelluksen arkkitehtuuri koki niin paljon muutoksia prosessin aikana. Dokumentointi olisi saattanut olla selkeämpi, jos jälkeenpäin olisi luotu UML-mallit viimeisimpien muutosten pohjalta. Tämä arkkitehtuurin sekavuus haittaa

sovellettavuutta. Dokumentointia myös jouduttiin tiivistämään, jotta opinnäytetyöstä ei olisi tullut liian pitkä ja sekava.

6 Pohdinta

Työn aihe oli ajankohtainen, mutta toteutus venyi liian pitkälle aikavälille henkilökohtaisten ongelmien sekä liian joustavan aikataulun takia. Teknologia kehittyy jatkuvasti, joten voi olla mahdollista, että tähän aiheeseen liittyen on tullut sillä välin useita uusia samankaltaisia tutkimuksia ja jopa täysin uusia versioita käytetyistä tekniikoista. Tämä taas pienentää opinnäytetyön merkittävyyttä.

Lähteisiin suhtaudutaan kriittisesti. Aiheesta ei juurikaan löytynyt kirjallisia lähteitä, jotka olisivat olleet ajan tasalla tai opinnäytetyön tekijöiden saatavilla. Tämän vuoksi työssä jouduttiin turvautumaan paljon nettilähteisiin. Osa niistä oli luotettavia, kuten työssä käytettyjen tekniikoiden omat dokumentoinnit, mutta harmillisen useasta aiheesta löytyi työn aloitusvaiheessa tietoa vain esimerkiksi yksityishenkilöiden blogijulkaisuista. Myös tulosten luotettavuuteen suhtaudutaan kriittisesti. Opinnäytetyön tekijät olivat täysin uuden asian äärellä ja se vaati hyvin paljon itseopiskelua. Jotkut toteutetut asiat olisi pystytty ehkä luomaan järkevämmin tai nopeammin, jos kyseessä olisi ollut kokeneempia toteuttajia.

Tämän kaiken huomioon ottaen opinnäytetyön tulos oli kuitenkin positiivinen: työ toteutti suhteellisen hyvin ne vaatimukset, mitä sille oli alunperin asetettu. Opinnäytetyö on avuksi niille ihmisille, jotka eivät ole tietoisia, kuinka siirrettävyys toteutuu Microsoft-ympäristössä. Se tarjoaa selvennystä siinä käytettäviin tekniikoihin, mutta ei mene liian yksityiskohtaiseksi. Koska opinnäytetyössä toteutettu sovellus on hyvin yksinkertainen, pitäisi sen sovellettavuuden olla osittain helppoa. Dokumentointi sovelluksen luomisesta jäi hieman vajaaksi, mutta se näyttää kuitenkin sovelluksen pääpiirteet.

Jos käytetyistä tekniikoista on tullut uutta tietoa tämän opinnäytetyön luomisen aikana, voisi aiheesta tehdä laajempaa jatkotutkimusta esimerkiksi tutustumalla syvemmin töissä käytettyihin tekniikoihin ja toteuttamalla monimutkaisemman sovelluksen. Koska opinnäytetyössä on tutkittu toimeksiantajan mainitsemia tekniikoita, voisi jatkotutkimusta laajentaa myös muihin tekniikoihin. Uusia tekniikoita voisi verrata keskenään ja tutkia, mitkä niistä ovat kannattavampia siirrettävyyden toteuttamisessa. Yksi toimiva idea voisi olla myös Microsoft-ympäristön ulkopuolelle astuminen ja siirrettävyyden tutkiminen muissa ympäristöissä. Jatkotutkimuksessa voisi myös tutustua enemmän siirrettävyyden historiaan ja tutkia sen kannattavuutta nykyaikaisissa sovelluksissa.

Lähteet

About Mono. 2017. Dokumentointi Monon sivuilla. Viitattu 11.2.2017. http://www.mono-project.com/docs/about-mono/.

All about Ionic. N.d. Ionicin kuvaus Ionicin sivuilla. Viitattu 10.2.2017. https://ionic.io/about.

Angular: Features and Benefits. N.d. Angularin kuvaus Angularin sivuilla. Viitattu 20.2.2017. https://angular.io/features.html.

AngularJS – Overview. N.d. Artikkeli Tutorialspoint-sivustolla. Viitattu 17.1.2017. http://www.tutorialspoint.com/angularjs/angularjs_overview.htm.

Biharisingh, A. 2016. Build Your First Mobile App With Ionic 2 & Angular 2 – Part 1. Artikkeli Gone Hybrid -sivustolla. Viitattu 12.2.2017. http://gonehybrid.com/build-your-first-mobile-app-with-ionic-2-angular-2/.

Carter, P., Dykstra, T., Lander, R. & Onderka, P. 2016. .NET CORE. Viitattu 6.02.2017. https://docs.microsoft.com/en-us/dotnet/articles/core/index.

Carter, P. & Knezevic, Z. 2016. .NET Core - .NET Goes Cross-Platform with .NET Core. Microsoft, MSDN Magazine Blog: Volume 31 Number 4. Viitattu 5.02.2017. https://msdn.microsoft.com/en-us/magazine/mt694084.aspx.

Chapter 1: All about Ionic. N.d. Dokumentaatio Ionicin sivuilla. Viitattu 11.2.2017. http://ionicframework.com/docs/guide/preface.html.

Core Concepts. N.d. Dokumentaatio Ionic Frameworkin sivuilla. Viitattu 10.2.2017. http://ionicframework.com/docs/v2/intro/concepts/.

Cross-platform Definition. 2005. Artikkeli The Linux Information Project –sivustolla. Viitattu 14.2.2017. http://www.linfo.org/cross-platform.html.

Dawson, A. 2011. Future-Proof Web Design: A Survival Guide. John Wiley & Sons. Viitattu 14.2.2017. http://janet.finna.fi, Books24x7.

Developers Guide: Introduction. N.d. Johdanto AngularJS-sivustolla. Viitattu 17.1.2017. https://docs.angularjs.org/guide/introduction.

Ewerlöf, A. 2016. When should I use TypeScript. Viitattu 24.09.2016. https://medium.freecodecamp.com/when-should-i-use-typescript-311cb5fe801b#.sdnvw383c.

Fenton, S. 2014. Pro TypeScript: Application-Scale JavaScript Development. Apress. Viitattu 4.10.2016. http://janet.finna.fi, Books24x7.

Hanselman, S. 2016. Publishing an ASP.NET Core website to a cheap Linux VM host. julkaisu Scott Hanselmanin blogissa. Viitattu 27.2.2017. http://www.hanselman.com/blog/PublishingAnASPNETCoreWebsiteToACheapLinuxVMHost.aspx.

Hejlsberg, A., Mohamed, H., Saeeduddin, A. & Zhong, P. 2016. TypeScript Language Specification. Dokumentaatio TypeScriptin GitHub-sivuilla. Viitattu 23.09.2016. https://github.com/Microsoft/TypeScript/blob/master/doc/spec.md.

Introducing WinSCP. 2014. Dokumentaatio WinSCP-sivustolla. Viitattu 27.2.2017. https://winscp.net/eng/docs/introduction.

Introduction: What are web components? N.d. Artikkeli Webcomponents.org-sivustolla. Viitattu 16.2.2017. https://www.webcomponents.org/introduction.

Kananen, J. 2012. Kehittämistutkimus opinnäytetyönä. Jyväskylä: Jyväskylän ammattikorkeakoulu.

Kananen, J. 2015. Kehittämistutkimuksen kirjoittamisen käytännön opas. Jyväskylä: Jyväskylän ammattikorkeakoulu. Viitattu 29.09.2016.

Kremer, J. 2016. Angular, version 2: proprioception-reinforcement. Julkaisu Angular-blogissa. Viitattu 19.1.2017. http://angularjs.blogspot.fi/2016/09/angular2-final.html.

Luukkainen, M & Laine, H. 2013. Luentomoniste kurssille: Ohjelmistojen mallintaminen. Viitattu 22.2.2017. https://www.cs.helsinki.fi/u/mluukkai/ohmas10/ohma.pdf.

Lynch, M. 2017. Announcing Ionic 2.0.0 Final. Julkaisu Ionicin blogissa. Viitattu 17.2.2017. https://blog.ionic.io/announcing-ionic-2-0-0-final/?_ga=1.255168395.2024880829.1486476954.

Migration Concepts. N.d. Dokumentaatio Ionic Frameworkin sivuilla. Viitattu 13.2.2017. https://ionicframework.com/docs/v2/intro/migration/.

Mono Releases. 2017. Dokumentaatio Monon sivuilla. Viitattu 11.2.2017. http://www.mono-project.com/docs/about-mono/releases/.

Mooney, J. N.d. Bringing Portability to the Software Process. Viitattu 15.2.2017. https://pdfs.semanticscholar.org/ef08/695a3e437ea58f48e247f72b4bac61140c5c.pdf.

Poikolainen, T. 2016. Sähköpostilla lähetetyt sovellusarkkitehtikuvat. Viitattu 29.09.2016.

Porting. 2017. Dokumentaatio Monon sivuilla. Viitattu 10.2.2017. http://www.mono-project.com/docs/advanced/runtime/porting/.

Sharif, S. 2016. Difference between Angular 1 VS Angular 2. Julkaisu Technical Diaryn sivuilla. Viitattu 18.2.2017. https://www.technicaldiary.com/difference-angular-1-vs-angular-2/.

Tolvanen, P. 2013. Ketteryys haltuun: Yleisimmät ketterät käytännöt. Ketteryys haltuun –artikkelisarjan ensimmäinen osa Sininen Meteoriitti –sivustolla. Viitattu 22.2.2017. https://www.meteoriitti.com/2013/06/06/ketteryys-haltuun-yleisimmat-ketterat-kaytannot/.

What's Yeoman? N.d. Yeoman-sivuston etusivulla oleva johdanto. Viitattu 27.2.2017. http://yeoman.io/.

What is a Reverse Proxy Server? N.d. Käänteisen välityspalvelun merkityksen selitys Nginx-sivustolla. Viitattu 27.2.2017. https://www.nginx.com/resources/glossary/reverse-proxy-server/.