SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

MONIALUSTAINEN PILVI-POJAINEN MOBIILISOVEL-LUS

TE-KIJÄ/T: Leevi Ojala

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä(t) Leevi Ojala	
Työn nimi Monialustainen Pilvipohjainen Mobiilisovellus	
Päiväys 30.5.2018	Sivumäärä/Liitteet 26
Ohjaaja(t) Jussi Koistinen, Lehtori ja Mikko Pääkkönen, TKI-asiantuntija	
Toimeksiantaja/Yhteistyökumppani(t) Savonia-AMK	

Tiivistelmä

Opinnäytetyön tarkoituksena on luoda monialustainen ja pilvipohjainen mobiilisovellus ja tutkia samalla käyttämieni teknologioiden käytettävyyttä, sekä sovelluksen käyttöönoton haastavuutta. Ajatus on luoda tyhjästä sekä usealla käyttöjärjestelmällä toimiva mobiilisovellus, palvelinpään koodi, sekä kaikki infrastruktuuri mitä kyseisen sovelluksen ylläpitämiseen tarvitaan pilvipohjaisesti. Projektissa siis synnytetään tyhjästä tuotantovalmis sovellus, joka ominaisuuksiensa lisäksi pystyisi vastaamaan oikeasti tuotannossa tulevaa rasitusta ja muista haasteita, kuten skaalattavuutta.

Käytetyt teknologiat Microsoft Azure ja Xamarin.Forms on valikoitu niin, että mahdollisimman paljon asioita olisi kehittäjän kannalta jo valmiiksi olemassa (kuten tietoturva), eikä näihin asioihin pitäisi kiinnittää huomiota. Tämä tekee kehityksestä sulavampaa kehittäjän näkökulmasta, kun kehittämisessä voi keskittyä pelkästään toiminnollisuuteen. Tämä myös säästää kehittämisen resursseja, kun työntekijä voi keskittyä olennaiseen. Myös kriittisten virheiden määrä pienenee, kun toiminnallisuutta on ”automatisoitu” teknologioihin.

Opinnäytetyön lopputuloksena sovelluksen lopullinen toteutus täytti kaikki ennalta suunnitellut kriteerit niin tuotantovalmiuden, kuin myös toiminnallisuuden osalta. Valikoidut teknologiat olivat kuitenkin lupaavista ominaisuuksistaan huolimatta käytännössä vaikeakäyttöisiä ja materiaali niistä vanhentunutta. Aikaa kului paljon kehitysympäristön säätämiseen kehitystyön sijaan. Kuitenkin tulee huomioida se, että teknologiat ovat vielä suhteellisen nuoria ja kehittyvät nopeasti ja juuri tuo nopea kehitys selittää osan kehitysympäristön kanssa olleista ongelmista.

Avainsanat
sovelluskehitys, Xamarin, pilvipohjainen, monialustainen

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Leevi Ojala			
Title of Thesis Multi-Platform Cloud Based Mobile Application			
Date	30 May 2018	Pages/Appendices	26
Supervisor(s) Mr. Jussi Koistinen, Senior Lecturer and Mr. Mikko Pääkkönen, ROI Specialist			
Client Organisation /Partners Savonia University of Applied Sciences			

Abstract

The purpose of this thesis was to create a multi-platform and cloud based mobile application while studying the usability of the technologies, and the challenge of deploying the application. The idea was to create from blank a multi-platform mobile application, a server-side code, and all the infrastructure needed to maintain that application cloud-based. The project thus generates an empty production-ready application that, in addition to its own characteristics, would be able to really respond to future strain and other challenges such as scalability.

The used technologies Microsoft Azure and Xamarin.Forms were selected in a way that as many things as possible exist already for the developer (such as security), and these issues should not be considered. This makes the development smoother from the developer's point of view, when development can focus solely on functionality. This also saves the development resources when the employee can focus only on the essentials. Also, the number of critical errors is reduced when the functionality is "automated" in technologies.

As a result of this thesis the final implementation of the application fulfilled all the pre-designed criteria in terms of production readiness and functionality. However, despite the promising features, the selected technologies were virtually ineffective, and the material outdated. It took a lot of time to adjust the development environment to the required development stage. However, it should be noted that the technologies are still relatively young and are developing rapidly, and that rapid development explains some of the problems with the development environment.

Keywords
software developement, Xamarin, cloud based, multi-platform

SISÄLTÖ

1 JOHDANTO

Opinnäytetyön tarkoituksena on kehittää monialustainen mobiilisovellus. Käytännössä tämä tarkoittaa sitä, että sovelluksesta on versionsa useammalle erilaiselle mobiilikäyttöjärjestelmälle. Tässä tapauksessa nuo kyseiset käyttöjärjestelmät ovat Ios ja Android. Sovelluksen tulisi myös olla tuotantovalmis, eli sen tulisi olla tietoturvan, vakauden ja skaalattavuuden osalta sellainen, että se kestäisi julkaistuna sovelluksena. Tämän takia työkaluiksi valittiin Microsoft Azuren palvelut, joilla kyseinen toteutus on mahdollista toteuttaa ilman todella syvällistä asiantuntemusta palvelinpuolesta, sekä Xamarin.Form, jolla voidaan toteuttaa monialustainen sovellus helposti ilman koodin toistamista.

Sovelluksen toiminnallisuudeksi valittiin chat ”seinä”, jonne käyttäjät voivat kirjoittaa anonyymisti kirjoituksia. Kirjoituksiin sisällyetään myös äänestysmahdollisuus, jolloin paljon positiivisia ääniä saaneet kirjoitukset näytetään ensin ja negatiivisia ääniä keränneet taas viimeisenä. Ennen sovelluksen käyttöä käyttäjän täytyy myös tunnistautua jotenkin ja tämän toiminnon toteutukseen valittiin Facebook kirjautuminen.

2 TYÖKALUT JA PALVELUT

Seuraavassa kappaleessa käydään läpi koko kehityskaaren aikana käytettävät työkalut ja palvelut. Sovelluskehityksessä on yleisesti erittäin tärkeää, että kehitysympäristö on ajantasainen ja kunnossa, mutta tässä kehitystilanteessa käyttöympäristön ja sen työkalujen ajantasaisuus korostuu entisestään. Syynä tähän on Xamarin teknologian nopea kehitys, joka ei välttämättä toimi halutulla tavalla, jos käytetään vanhentuneita tai muuten vääriä kehitystyökaluja.

2.1 Kehitystyökalut

Kehitystyökaluilla tarkoitetaan tässä kappaleessa työkaluja, joilla on rakennettu käyttöliittymän sovellusta sekä palvelinpuolen koodia.

2.1.1 C#

C# on Microsoftin keittämä ohjelmointikieli. Se julkaistiin vuonna 2000 ja sen tarkoituksena oli toimia Microsoftin .Net konseptin pääohjelmointikielenä. Kieleen haluttiin saada C++:n tehokkuus ja Javan helppokäyttöisyys, joten luonnollisesti kyseessä on oliopohjainen ohjelmointikieli. kyseinen ohjelmointikieli on myös vahvasti tyypitetty, mikä yhdessä Visual Studio IDE:n kanssa helpottaa olio- ja tyyppivirheicen karsimista koodista. C# toimii myös ohjelmointikielenä projektissa käytettävissä Xamarin ja Visual Studio ympäristöissä. (Osborn, 2000)

2.1.2 Visual Studio

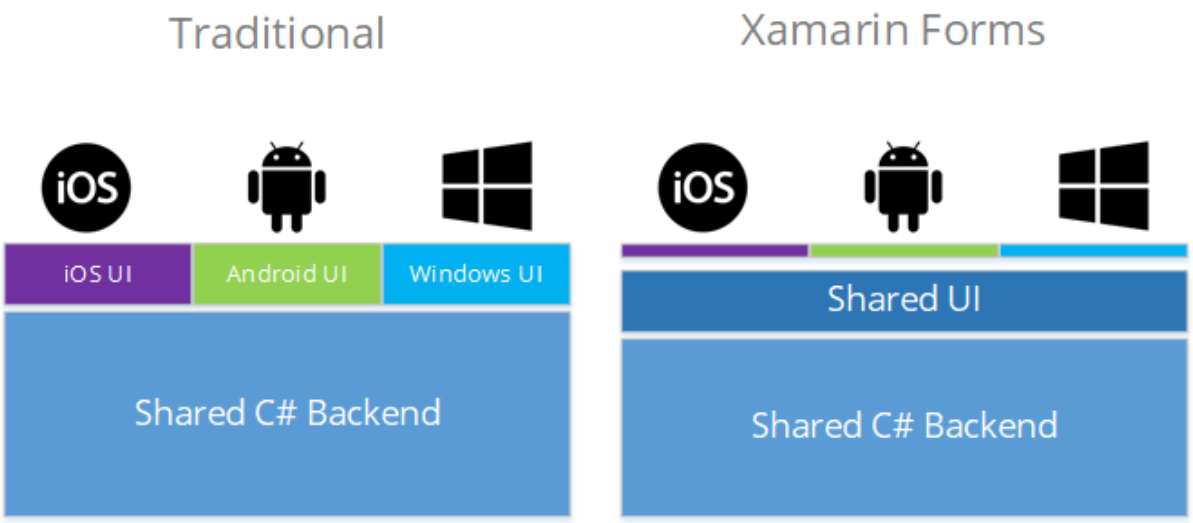
Visual Studio on Microsoftin kehittämä ohjelmankehitysympäristö(IDE), joka tukee useita ohjelmointikieliä. Visual Studio käyttää Microsoftin sovelluskehitysalustoja, kuten .NET, Windows Forms ja Microsoft Silverlight. Juuri nämä sovelluskehitysalustat yhdistettynä kattavaan määrään kolmansien osapuolien lisäosia tekevät Visual Studiosta paljon laajemman kokonaisuuden kuin vain koodieditorin. Esimerkkinä tästä voisi mainita Microsoftin Entity Frameworkin, joka on objektien relaatiokartoitukseen käytettävä kehitysalusta, maanläheisemmin se siis hoitaa tietokantayhteyksiä helposti suoraan kooditasolta. Tämä on vain yksi esimerkki monista kehitysmahdollisuuksista, joita Visual Studio tarjoaa.

Visual Studion uusin version on tällä hetkellä Visual Studio 2017, josta käytetään tässä kehityksessä *Community* versiota, joka on yksityisille kehittäjille ilmainen. Visual Studiosta on myös saatavilla *Professional* ja *Enterprise* versiot, jotka ovat tarkoitettu ominaisuuksien ja lisensointinsa puolesta isommille yrityksille ja organisaatioille.

2.1.3 Xamarin.Forms

Xamarin Form on Xamarin nimisen yhtiön tuote, joka mahdollistaa monialustaisen sovelluskehityksen. Sen, kuten myös muiden Xamarinin tarjoamien tuotteiden ideana on tarjota jaettu koodikanta, joka mahdollistaa natiivien Mobiilisovellusten kehittämisen C# ohjelmointikielellä. Tuotteena juuri

Forms tarjoaa mahdollisuuden kehittää bisneslogiikan lisäksi myös käyttöliittymän jaetulla koodilla, eli monialustainen ohjelma voidaan kehittää alusta loppuun yhdellä kerralla ja muuntaa sitten kullekin mobiilikäyttöjärjestelmälle natiiviksi sovellukseksi.



kuva 1. Xamarinin jaettu UI

Tämä säästää ohjelmistokehityksessä resursseja, kun sovellus tarvitsee kehittää vain kerran, eikä jokaiselle alustalle erikseen. Myös eri teknologioiden opettelu voi nousta haasteeksi mobiilikehityksessä, koska kolmelle suurimmalle mobiilikäyttöjärjestelmälle Androidille, iOS:ille ja Windowsille sovelluksia tehtäessä ohjelmointikieli on aina eri. Xamarin helpottaa tässä ohjelmoijaa siten, että kaikki koodi voidaan tehdä Microsoftin C# kielellä. Kuvassa 1. on havainnollistettu perinteisen sovelluskehityksen ja Xamarin.Forms kehityksen eroa. Kuvan vasemmalla puolella kuvatussa perinteessä sovelluskehityksessä jokainen käyttöjärjestelmä vaatii erikseen tehdyn käyttöliittymän, kun taas oikealla puolella kuvattu Xamarin.Forms vaatii vain yhden käyttöliittymätoteutuksen, joka on jaettu kaikille käyttöjärjestelmille.

2.2 Microsoft Azure ja palvelut

Microsoft Azure on Microsoftin tarjoama pilvipalvelu, joka tarjoaa Iaas ja Paas palveluita, sekä tuotteita, joissa näitä molempia on yhdistetty. Käytännössä siis puhutaan alustasta, jonne oman sovelluksensa palvelinpään voi laittaa niin, että se on internetin välityksellä aina saatavilla. Tämä vapauttaa myös kehittäjän kaikesta omasta infrastruktuurista, koska omia palvelimia ei tarvita, vaan käytetään virtuaalikonetta joka saa laskentatehonsa jostakin Microsoftin konesalista.

2.2.1 Iaas ja Paas

IaaS ja PaaS ovat pilvipalvelutyyppejä, joita molempia Microsoft Azure tarjoaa. IaaS on lyhenne sanoista *Infrastructure as a service*, Mikä tarkoittaa sitä, että pilvipalelu tarjoaa mahdollisuuden pystyttää oman sovellus infrastruktuurin pilveen. Pilvipalveluun voidaan siis laittaa internetin kautta saataville palvelimia tai muuta palveluinfraa, jolloin vapaudutaan täysin omien laitteiden hankinnasta. Yleensä IaaS palvelut eivät ole ilmaisia ja kustannukset voivat olla sovellustasolla merkittäviä, mutta

enenevissä määrin myös suurille käyttäjämäärille kannattavien pilvipalveluita löytyy, eikä oman infran fyysinen pystytys ole kannattavaa. (Eronen, 2016)

PaaS puolestaan on lyhenne sanoista *Platform as a service*, mikä tarkoittaa, että pilvipalvelu tarjoaa jonkinlaisen sovellusalustapalvelun. Tällaisesta on hyvä esimerkki tässäkin projektissa käytetty Azuren *App Service*, joka tarjoaa sovelluskehitysalustan, jolla mobiilikehittäminen on helpompaa ja yksinkertaisempaa. Siinä myös yhdistyvät Microsoftin Iaas ja Paas palvelut, kun *App Servicen* pohjalla käytetään pilveen itse luotua infrastruktuuria, eli IaaS palvelua. (Microsoft, 2018)

3 KEHITYKSEN KULKU

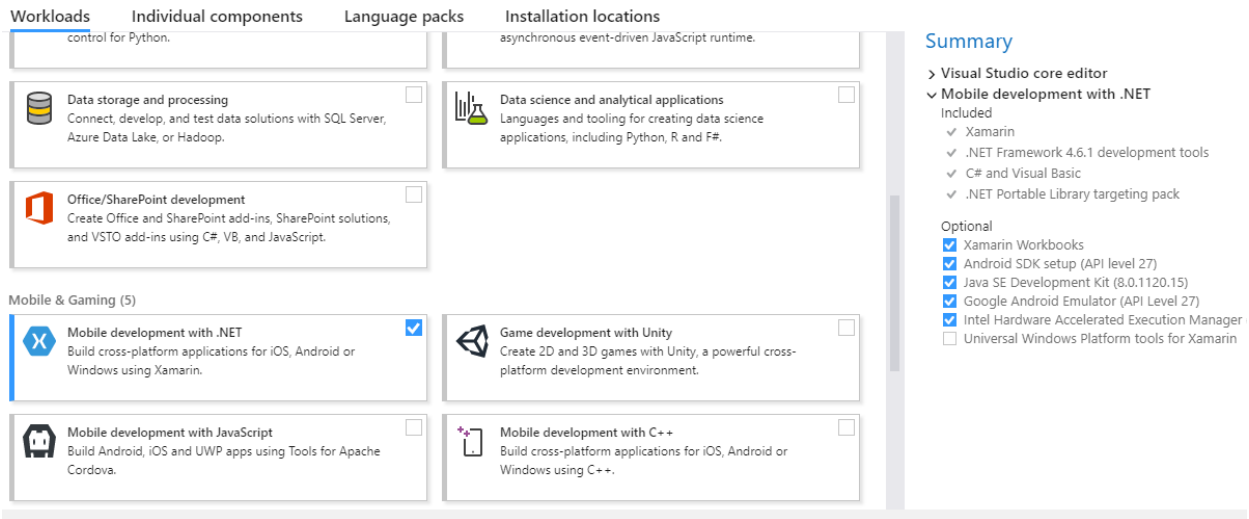
Tässä kappaleessa käydään läpi hieman valituilla tekniikoilla toteutetun sovelluksen kehityksen kulkua juuri siitä näkökulmasta, että teknologiat ovat aikaisemmin suhteellisen tuntemattomia. Tarkoitus on eritellä seikkoja, jotka näkisin positiivisiksi tehokkaan kehityksen kannalta, sekä niitä asioita jotka voivat olla näiden tekniikoiden kompastuskiviä.

3.1 Kehitys ympäristön asennus

Xamarinilla ja Azurella toimivaa sovellusta kehittäessä on tärkeää, että tarvittavat kehitystyökalut, sekä palvelut ovat kunnossa. Käytännössä tämä tarkoittaa sitä, että halutaan pilvipalvelun ja siellä olevan infrastruktuurin olevan kunnossa jo kehityksen alkuvaiheessa, jotta pystytään kehittämään myös palvelinpään koodia reaaliaikaisesti ja testaamaan muutoksemme saman tien. Näin vältytään ikäviltä bugeilta myöhemmässä vaiheessa ja pidetään kehitys ketteränä.

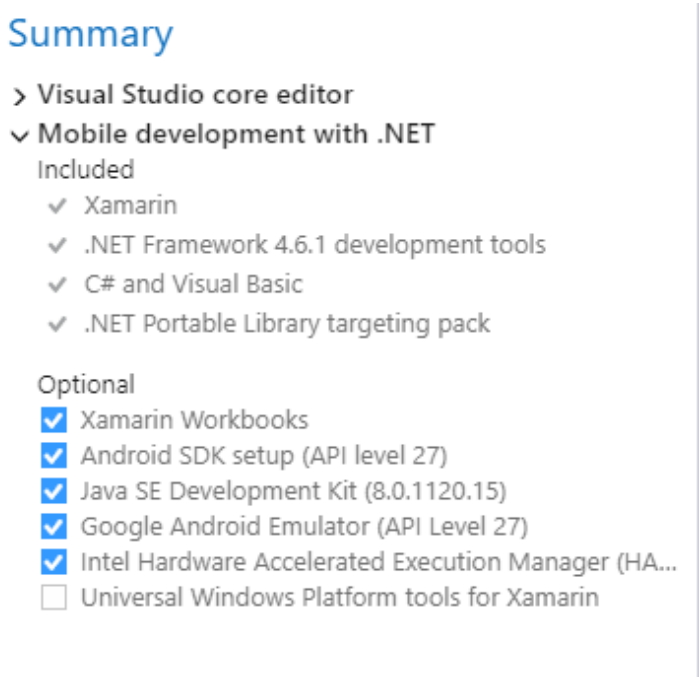
3.1.1 Sovelluskehityksen kehitysympäristö

Sovelluskehitykseen vaadittavien työkalujen käyttöönotto on tässä tapauksessa perin helppoa. Microsoftin ostaessa Xamarinin se lisättiin Visual Studion ominaisuudeksi, jonka voi asennuksen mukana ottaa käyttöön vain yhdellä klikkauksella. Jo asennettuun Visual Studioon Xamarinin työkalut saa käyttöönsä helposti käyttämällä Visual Studio Installer työkalua, jolloin jo tehtyä asennusta voi muokata. Näkymä näissä molemmissa asennustilanteissa on kuitenkin sama ja se on näytetty kuvassa 2.



Kuva 2. Visual Studio Installerin asennusikkuna

Kuvassa olevan asennuksen yhteenvedon kohdalla (kuva 2.) nähdään myös, miten asennuksen yhteydessä työasemalle asentuu myös Androidin vaatimat riippuvuudet, jolloin näiden asennuksesta ei tarvitse enää huolehtia, kun kehitysympäristö on jo pystyssä. Tämän jälkeen jäljelle jää vain asennuksen valmistumisen odottaminen ja tämän jälkeen käyttöympäristö on käyttöönottoa vaille valmis.

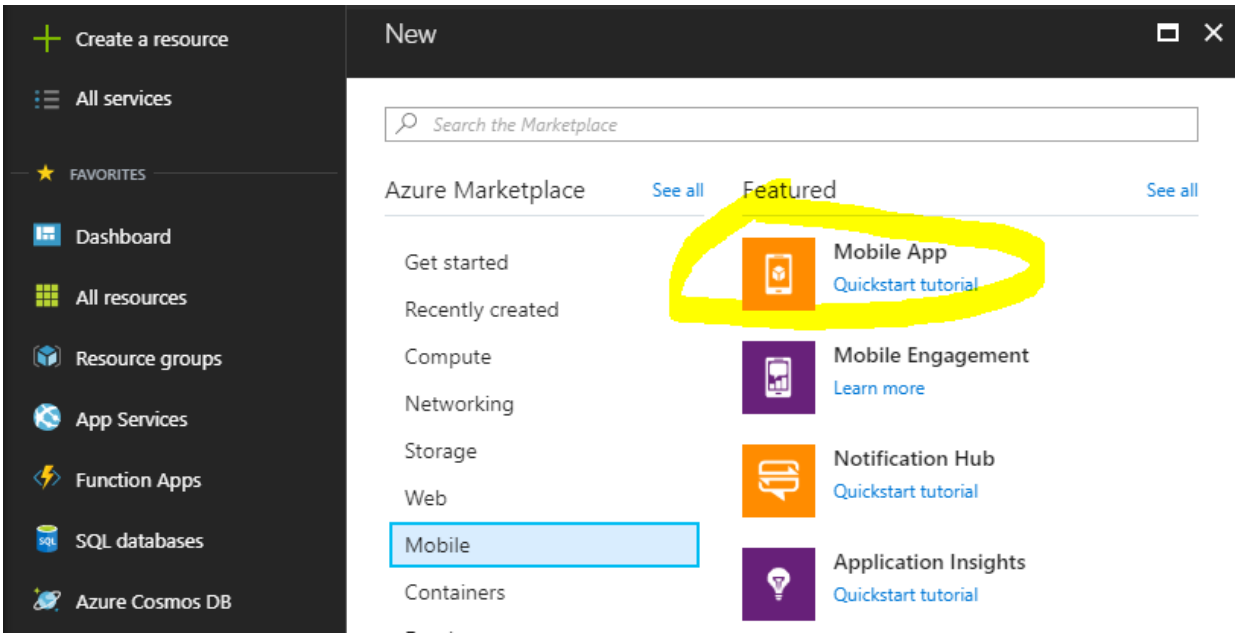


kuva 3. Yhteenveto Xamarin asennuksen sisällöstä

Xamarin kuitenkin vaatii uusimman 2017 version Visual Studiosta, jotta kaikki ominaisuudet toimivat oikein. Uusimman version Community version on kuitenkin pienille tiimeille sekä yksittäisille käyttäjille ilmainen, joten näiden kehitystyökalujen käytöstä pienessä mittakaavassa ei tule kehittäjälle kustannuksia.

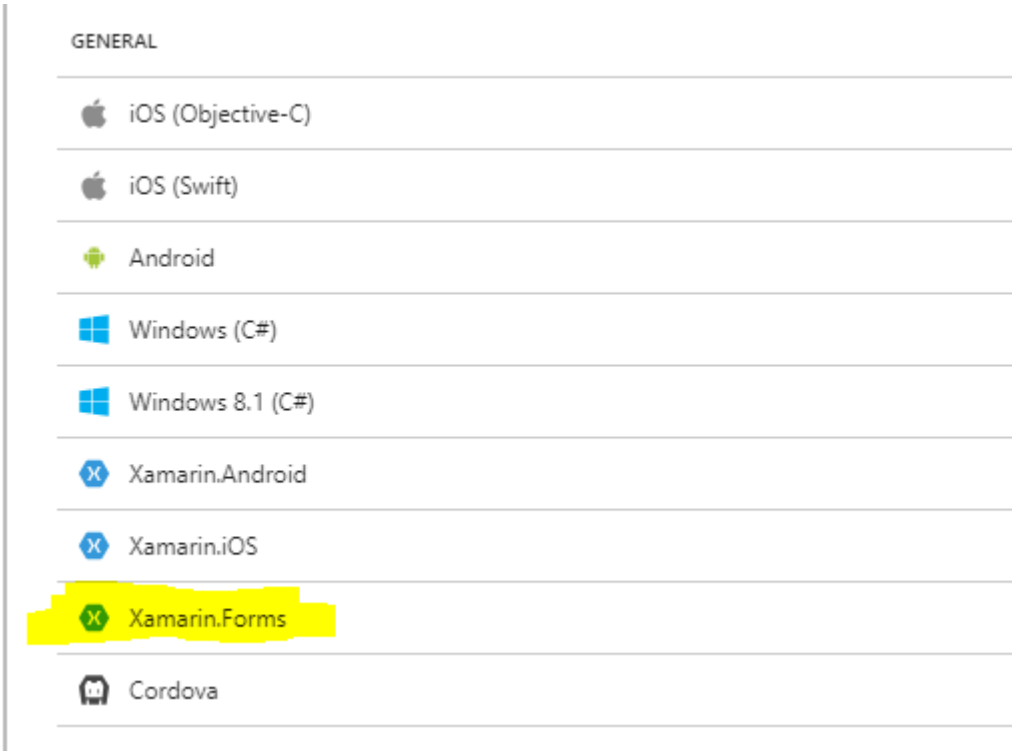
3.1.2 Pilvipalveluiden käyttöönotto

Jotta saadaan palvelinpuolen toiminnallisuus käyttöön, on otettava käyttöön pilvipalvelu, jonne sovelluksen palvelimen infrastruktuuri asennetaan. Microsoft Azurella koko palvelinkokonaisuuden asennus ja käyttöönotto onkin perin helppoa ja ohjattu käyttöönotto mahdollistaa palvelimen oikeaoppisen asennuksen, vaikkei tietämystä palvelininfrastruktuurista kauheasti olisikaan.



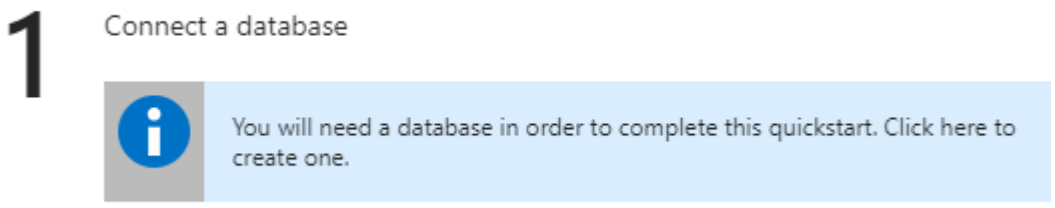
kuva 4. Azuren resurssinäkymä

Koska luodaan mobiilisovellusta, aloitetaan palvelinympäristön käyttöönoton luomalla *Mobile App* resurssin kuvan 4. mukaisesti. Tämän jälkeen mennään Azuren portaalissa luotuun resurssiin, joka voidaan *quickstart* toiminnon avulla ottaa käyttöön helposti. Ensiksi on kuitenkin valittava teknologia, jonka mukaisesti Azure palvelinpään rakentaa.



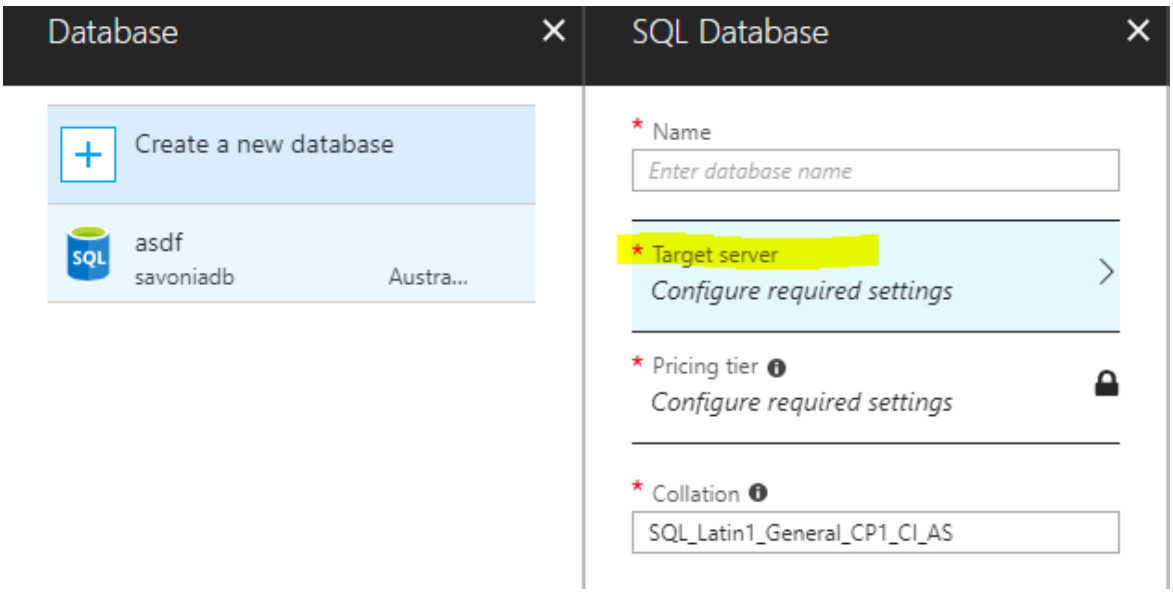
kuva 5. Mahdolliset Mobile App asennusvaihtoehdot

Kuvassa 5. nähdään kaikki mahdolliset teknologiat, joille Azure osaa suoraan palvelinkonfiguraation tehdä, mutta valitaan *Xamarin.Froms* teknologia. Vaikka tarjolla on Xamarinin Androidille ja iOS:ille suunnatut versiot, valitaan Forms, joka mahdollistaa kehityksen molemmille näille ja sen lisäksi myös Windows alustalle.



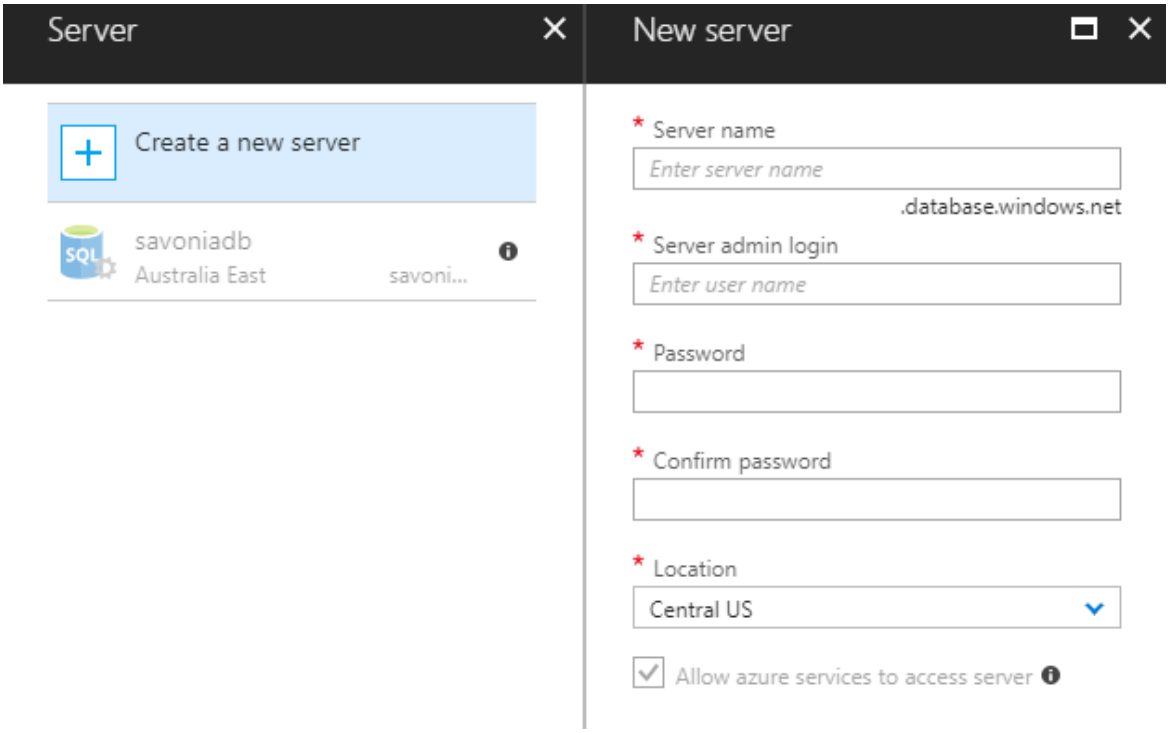
kuva 6. Tietokannan puuttumisesta johtuva ilmoitus

Seuravaksi tullaan sivulle, jossa saadaan kuvan 6. mukaisesti huomautus, että ei ole tietokantaa, johon äsken luotu resurssi voitaisiin yhdistää. Onneksi kyseistä resurssia klikkaamalla Aukeaa ikkuna, jonka kautta pääsee sen luomaan.



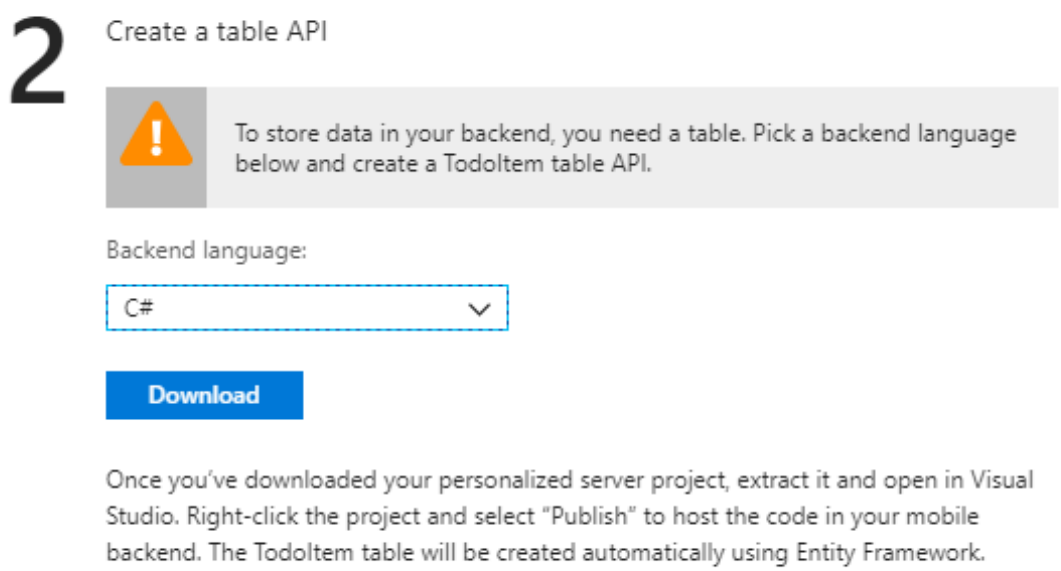
kuva 7. Tietokannan luominen

Vielä kuitenkin puuttuu palvelin, jonne tietokannan voisi luoda. Vaikka kuvan 7. mukaisesti on auki näkymä, jossa tietokantaa konfiguroidaan, on tiedoissa kuitenkin *Target Server* kohta, jotka klikatessa palvelimen puuttuminen noteerataan.



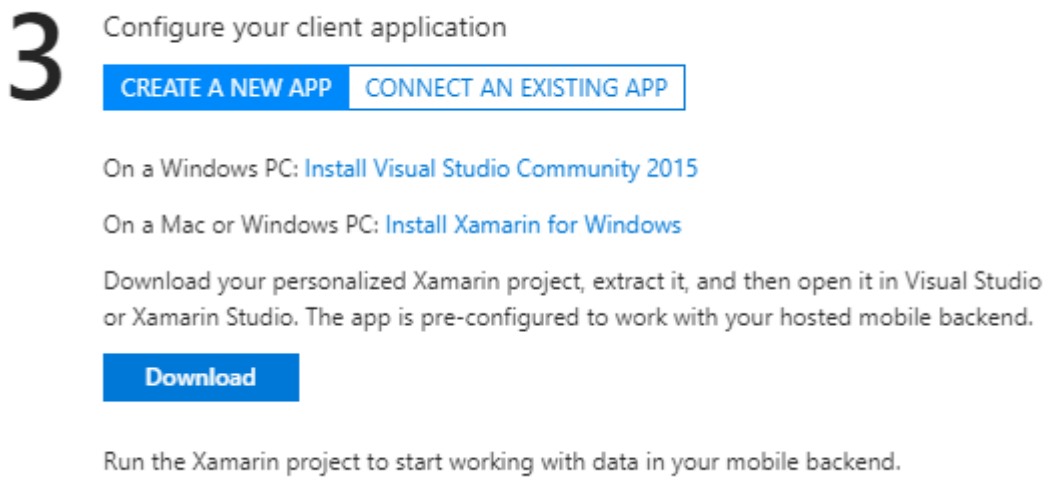
Kuva 8. Palvelimen luominen

Azure kuitenkin tarjoaa suoraan mahdollisuuden lisätä puuttuva palvelin pilvi infrastruktuuriin. Palvelimen luominen on erittäin yksinkertaista ja syöttämällä kuvassa 8. näkyvät perustiedot hoitaa Azure palvelimen asennuksen. Tämän jälkeen mennään vain takaisin kuva 7. ikkunaan jossa syötetään tietokannan tiedot loppuun ja luodaan se. Sitten onkin valmista ja voidaan siirtyä seuraavaan vaiheeseen.



Kuva 9. Palvelinpään ohjelmointikielen valinta

Seuraavassa vaiheessa jää tehtäväksi valita palvelinpään koodin ohjelmointikieli. Tarjolla olevat vaihtoehdot ovat C#, sekä Node.js. Tähän projektiin ohjelmointikieleksi valikoitui C# sen ennalta osaamisen vuoksi. On myös huomattavasti helpompaa kehittäjän kannalta, kun palvelinpään, sekä käyttöliittymäpään ohjelmointikieli on sama. Kun kieli on valittu, voi käyttäjä ladata palvelinpään koodit itselleen. Oletuksena palvelinpään koodi on tehty Microsoftin omalle *TodoItem* esimerkkisovellukselle ja koodi tuleekin muokata vastaamaan omaa sovellusta. Datayhteydet pilvipalvelun kanssa on automaattisesti konfiguroitu ladattuun koodiin, joten tehtäväksi jää vain Olioiden ja toiminnollisuuden muuttaminen omia tarpeita vastaavaksi.



kuva 10. Clientin lataamisen mahdollisuus

Kuvassa 3 nähdään myös Azuren tarjoama kolmas vaihe, jos myös käyttöliittymäpään esimerkkikoodin voisi ladata koneelle, jolloin datayhteys pilvipalveluun olisi valmiina. Tämä esimerkkikoodi on kuitenkin ankarasti vanhentunutta, eikä se yrityksistäni huolimatta mennyt edes järkevästi kääntäjästä

läpi. Kannattaakin luoda Visual Studiossa uusi Xamarin.Forms sovellus ja yhdistää se itse Azuressa luomaamme *Mobile Serviceen*. Ohjeet tähän löytyy Xamarinin dokumentaatiosta, joita seuraamalla sovelluksen saa suhteellisen pienellä vaivalla toimimaan yhteen pilvipalvelumme kanssa. (Microsoft, 2016)

3.2 Dokumentaatio ja materiaali

Xamarinin käyttöönotossa ja kehityksen muissakin vaiheissa on otettava huomioon sen nopea kehitys viimevuosien aikana. Esimerkiksi monessa muussa kehitystilanteessa hyödylliset YouTube videot osoittautuivat moneen kertaan hyödyttömiksi juuri nopean kehityksen takia, eli niissä neuvottavat asiat olivat vanhentuneita eivätkä siksi käyttökelpoisia. Myös Microsoftin omaa dokumentaatiota peratessa tulee olla varuillaan, ettei lue vanhentunutta tutoriaalia, joihin törmättiin jatkuvasti kehityksen edetessä. Kuitenkin, kun jaksaa etsiä ja on päivämääräkriittinen, löytyy niin Microsoftin dokumentaatiosta, sekä myös muista lähteistä hyviä tutoriaaleja kehityksen ja Xamarinin ympärillä olevien teknologioiden tueksi.

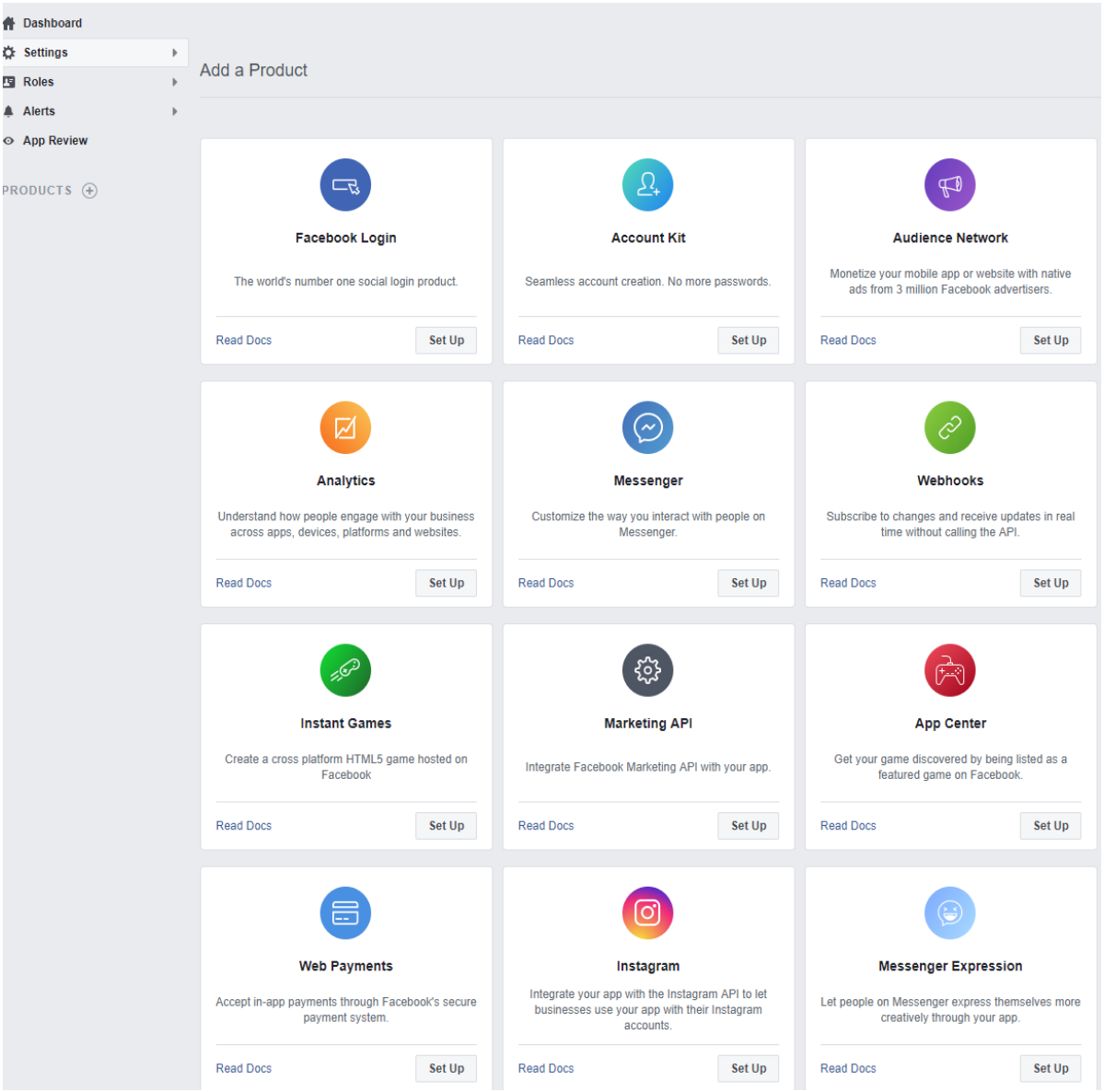
4 SOVELLUS

Sovelluksen pääominaisuus on toimia chat ”seinänä”, jonne kaikki sovellusta käyttävät ihmiset voivat kirjoittaa postauksia. Seinällä jokainen käyttäjä voi äänestää kirjoituksia positiivisilla tai negatiivisilla äänillä. Kaikki viestiminen sovelluksessa tehdään anonyyminä, vaikkakin sovelluksen käyttämiseen vaaditaan kirjautuminen Facebook palveluun. Seuraavissa kappaleissa käydään läpi sovelluksen toiminnallisuuden ja niiden taustalla oleva tekniikka.

4.1 Sisäänkirjautuminen

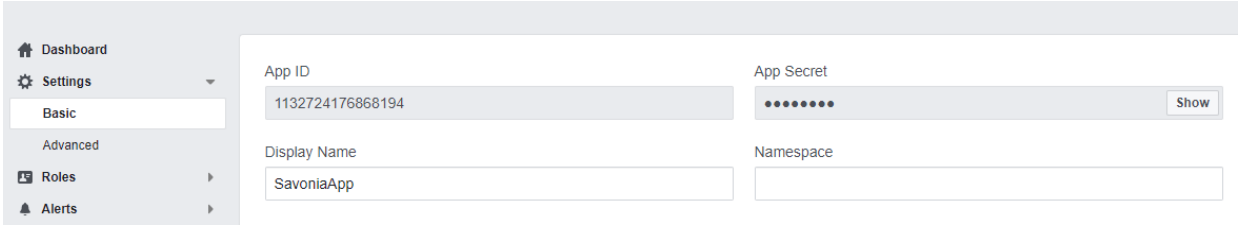
Sovelluksen kirjautuminen tehtiin käyttäen Xamarin.Auth Nuget palvelua. Se mahdollistaa käyttäjäautentikoinnin yleisimpien suurien palveluntarjoajien kanssa, kuten Facebook, Linkedin ja Google. (Xamarin) Näin saavutetaan sovellukselle tavoitteeksi asetettua tietoturvaa, koska valmis palvelu hoitaa kaiken tietoliikenteen ja palauttaa vain vastauksen siitä, että onnistuiko kirjautuminen.

4.1.1 Facebook sovelluksen rekisteröinti



Kuva 11. Facebookin saatavilla olevat kehittäjäsovellukset

Ennen kuin Xamarin.Auth palvelua voi käyttää, on rekisteröitävä sovellus käyttäen Facebookin sovelluskehittäjäportaalia. Tämä onnistuu perin helposti kirjautumalla ensin omalla Facebook käyttäjällään portaaliin ja valitsemalla luo uusi sovellus. Tämän jälkeen Aukeaa näkymä kaikkiin tarjolla oleviin sovelluksiin, kuten kuvassa 11. näkyy.



Kuva 12. Facebookiin luodun sovelluksen App ID

Tässä tapauksessa haluttu sovellus on *Facebook Login* ja sitä klikkaamalla pääsee luomaan sovellukseemme Facebook autentikaation. Kun kirjautuminen on otettu käyttöön luodulle sovellukselle Facebookin portaalissa, täytyy enää kaivaa tarvittavat kirjautumistunnukset sovelluksemme asetuksista. Nämä löytyvät suhteellisen helposti asetusten takaa, kuten kuvassa 12 nähdään.

Huomioitavaa kuitenkin on, että App ID ja varsinkin App Secret ovat vain kehittäjän omaan käyttöön ja niitä ei tule luovuttaa kenellekään ulkopuoliselle. Esimerkiksi julkisissa versionhallinnoissa kannattaa olla tarkkana, ettei vahingossa julkaise lähdekoodin mukana näitä henkilökohtaiseksi tarkoitettuja tunnuksia.

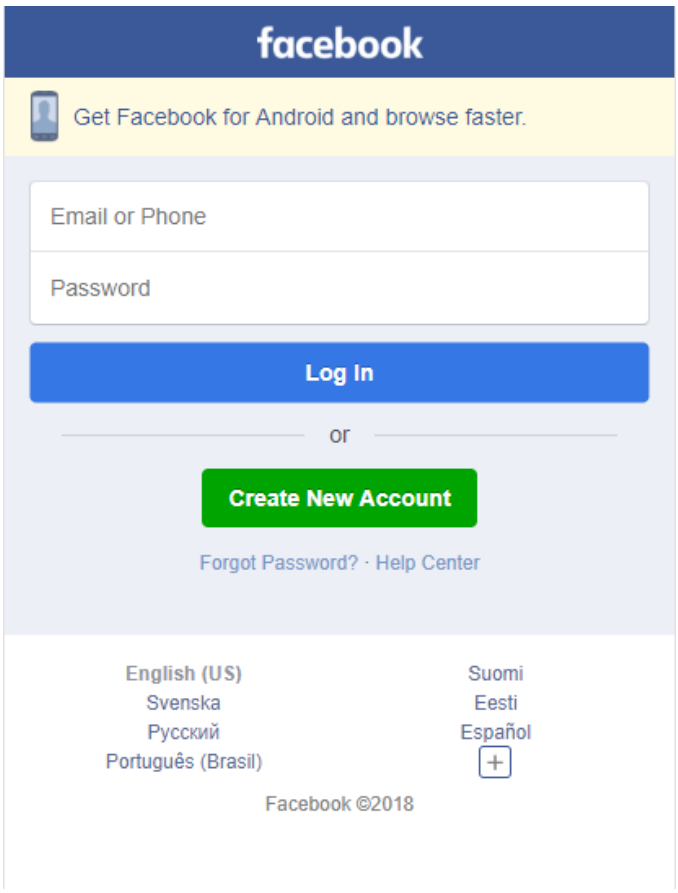
4.1.2 Autentikaation lisäys sovellukseen

Nyt kun sovellus on rekisteröity Facebookin päähän, tarvitaan enää sovelluksen id:n, jotta Xamarin.Auth palvelun voi otaa käyttöön. Kuvassa 13 on näytetty, että miten autentikointi otetaan käyttöön, sekä mitä tietoja siihen vaaditaan. *AuthorizeURL* ja *redirectUrl* ovat lähes aina samoja ja ne voi kopioida suoraan omaan lähdekoodiin, mutta *clientId* on aina sovelluskohtainen ja sen saa käyttöönsä edellisessä kappaleessa kuvaillulla tavalla.

```
OAuth2Authenticator auth = new OAuth2Authenticator
(
    clientId: "App ID from https://developers.facebook.com/apps",
    scope: "",
    authorizeUrl: new Uri("https://m.facebook.com/dialog/oauth/"),
    redirectUrl: new Uri("http://www.facebook.com/connect/login_success.html"),
    // switch for new Native UI API
    //      true = Android Custom Tabs and/or iOS Safari View Controller
    //      false = Embedded Browsers used (Android WebView, iOS UIWebView)
    //  default = false  (not using NEW native UI)
    isUsingNativeUI: use_native_ui
);
```

Kuva 13. Facebook autentikaation rekisteröinti

Myöskään käyttäjätietoja ei tarvitse käsitellä, kun laskeutumissivuina kirjautumisnappia painaessa toimii autentikoinnin tarjoavan sivuston oma kirjautumissivu. Sovelluksen käynnistyessä käyttäjälle näytetään sisäänkirjautumisikkuna, jossa ainut mahdollinen kirjautumismenetelmä on kirjautuminen Facebookin kautta. Kyseistä nappia painaessaan käyttäjä ohjataan Facebookin omalle kirjautumissivulle.



Kuva 14. Facebookin kirjatumis- ja rekisteröitymisikkuna

Facebookin omalla kirjautumissivulla on myös mahdollisuus luoda uusi käyttäjä, mikäli sovellusta käyttävällä henkilöllä ei sitä jo ennestään ole. Myös uuden käyttäjän luonti tehdään täysin Facebookin omalla sivustolla, joka poistaa kehittäjältä tässä tapauksessa paljon työtä ja päänvaivaa. Kun kirjautuminen on onnistunut, palautetaan käyttöliittymäkoodissa tieto onnistuneesta kirjautumisesta, sekä vähän käyttäjätietoja tunnistautumiseen.

4.2 Yhteyden avaaminen

```
public class AzureCloudService : ICloudService
    {
        MobileServiceClient client;

        public AzureCloudService()
        {
            client = new MobileServiceClient("https://my-backend.azurewebsites.net");
        }

    }
```

Kuva 15. Yhteyden avaaminen Mobile Serviceen

Ennen kuin tietokantaan pystytään ottamaan yhteyttä, täytyy meidän luoda käyttöliittymän koodissa *MobileServiceClient*, jonka avulla voimme ottaa helposti yhteyden Azuren Mobile Serviceen. Tämä luotu client huolehtii käytännössä kaiken tiedonsiirron tietokannan ja käyttöliittymän välillä. Kehittäjän tehtäväksi jää vain luoda oikeanmuotoisia *IMobileServiceTable* tauluja, joiden avulla tiedetään

mitä taulua tietokannassa kutsutaan. Tämä hakuprosessi on käyty läpi yksityiskohtaisemmin seuraavissa kappaleissa. Luotu client siis huolehtii tiedonsiirrosta ja kehittäjän tehtävä on vain kutsua sitä, kun tehdään CRUD-operaatioita.

4.3 Kirjoitusten hakeminen

```
IMobileServiceTable<T> table;
public AzureCloudTable(MobileServiceClient client)
    {
        this.client = client;
        this.table = client.GetTable<T>();
    }
```

Kuva 16. GetTable pyyntö

Kirjoituksia haettaessa kutsutaan clientin *GetTable* metodia, jolle annetaan *IMobileServiceTablen* muoto, joka tässä tapauksessa on *Posts*. Koodista löytyy myös samanmuotoinen muuttuja, *table.* Oletuksena on, että tietokannasta löytyy muotoa vastaava taulu. Taulut haetaan table muuttujaan, jonka jälkeen voidaan muuntaa table muuttujan sisältöä tai muotoa tarvitulla tavalla, jos se on tarpeellista. tämän muuttujan ylläpito on kuitenkin tärkeää siinä mielessä, että kun tehdään kutsuja tämä on taulu, jonka perusteella *Moblie Service* ylläpitää tietokantaa. Tämä tulee huomioida etenkin siinä vaiheessa, kun tietokantaan laitetaan tietoa tai sitä muokataan.

Kuva 17. Haetut kirjoitukset käyttöliittymässä.

Kuvassa 17. näkyy miten kirjoitukset näkyvät listanäkymänä käyttöliittymässä kirjautumisen jälkeen. Kirjoitukset ladataan kerralla käyttöliittymän avautuessa, mikä voi aiheuttaa sovelluksen hitautta kirjoituksien määrästä riippuen. Kehityskohtana olisi siis ladata kirjoituksia pienemmissä osissa, kun listaa rullataan eteenpäin.

4.4 Kirjoitusten lisääminen

```
public async Task<Post> CreateItemAsync(Post item)
{
    await table.InsertAsync(item);
    return item;
}
```

Kuva 18. Kirjoituksen lisääminen

Käyttöliittymän alareunassa on myös tekstikenttä ja plus ikoni, jonka avulla käyttäjä pystyy lisäämän oman viestinsä sovellukseen. Koska viestit tulevat näkyviin äänestyksen mukaan, eikä kronologisessa järjestyksessä, voi oma viesti hukkua helposti viestitulvaan ja sitä pääseekin tarkastelemaan myöhemmin käyttöliittymän yläreunassa olevan *My Posts* välilehden kautta. Sinne tulevat siis kaikki käyttäjän omat kirjoitukset. Kuvassa 18. on näytetty miten yksinkertaisella metodilla *item*, joka sisältää siis uuden kirjoituksen sisältävän olion lisätään tietokantaan asynkronisesti.

4.5 Kirjoitusten reaktioiden tallennus

```
public async Task<Post> UpdateItemAsync(Post item)
{
    await table.UpdateAsync(item);
    return item;
}
```

Kuva 19. Kirjoitusten muokkaaminen

Jokaisen kirjoituksen yhteydessä on myös reagointipainikkeet, joilla käyttäjä pystyy ilmaisemaan, onko kirjoituksen sisältö hänen mielestään positiivista vai negatiivista. Kun käyttäjä on antanut kirjoitukselle äänen, näkyy sininen ikoni sen mukaan, minkä äänen hän on antanut, eikä ääntä voi jälkeenpäin muuttaa. Ääni tallennetaan kirjoitukseen kuvan 19. mukaisessa metodissa, jossa kantaan tallennetaan kirjoitusolion sisältävä item muuttuja. Aina kun kirjoitus saa positiivisen tai negatiivisen arvon, muutetaan *item* oliossa olevaa *popularity* arvoa yhdellä numerolla sen mukaisesti ja muutos tallennetaan tietokantaan.

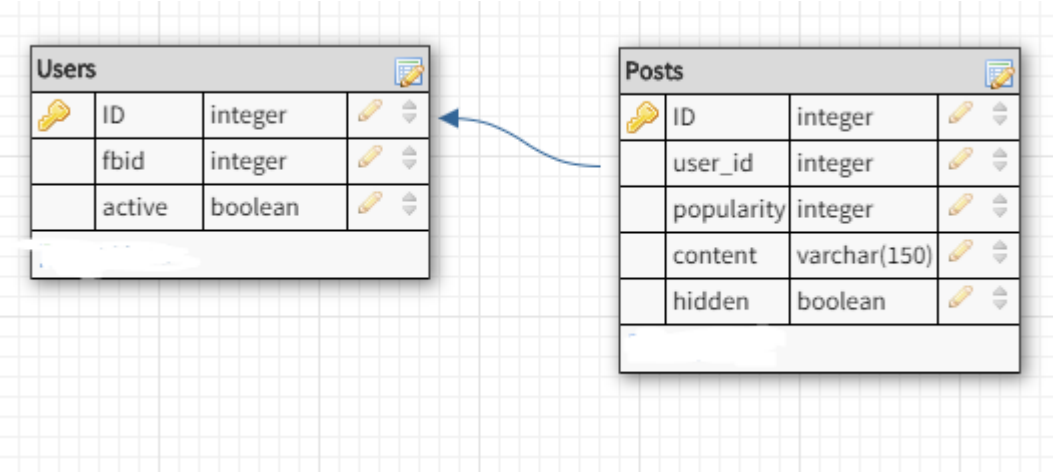
4.6 (Kehityskohta) Kirjoitusten automaattinen moderointi

Microsoft Azure tarjoaa tekoälypalveluita nimeltään *cognitive services*, joihin kuuluu myös tekstintunnistus. Jatkokehityksessä olisinkin toteuttanut ominaisuuden, jolla jokaisesta *Mobile Servicen* kautta kulkevasta viestistä suodatetaan esimerkiksi tiettyjä sanoja, joka auttaa tunnistamaan haitalliset viestit. Valitettavasti aika ei tässä yhteydessä riittänyt, mutta kyseessä olisi suhteellisen helposti lisättävä ominaisuus joka tekisi suuren vaikutuksen sovelluksen käyttömukavuuteen. (Analytics, 2018)

5 TIETORAKENNE

Sovelluksen tietorakenne on tarkoituksella perin yksinkertainen, koska tarkoituksena oli testata teknologioiden käytettävyyttä ja nopeaa käyttöönottoa, ennemmin kuin luoda erittäin laajaa kokonaisuutta.

5.1 Tietokanta



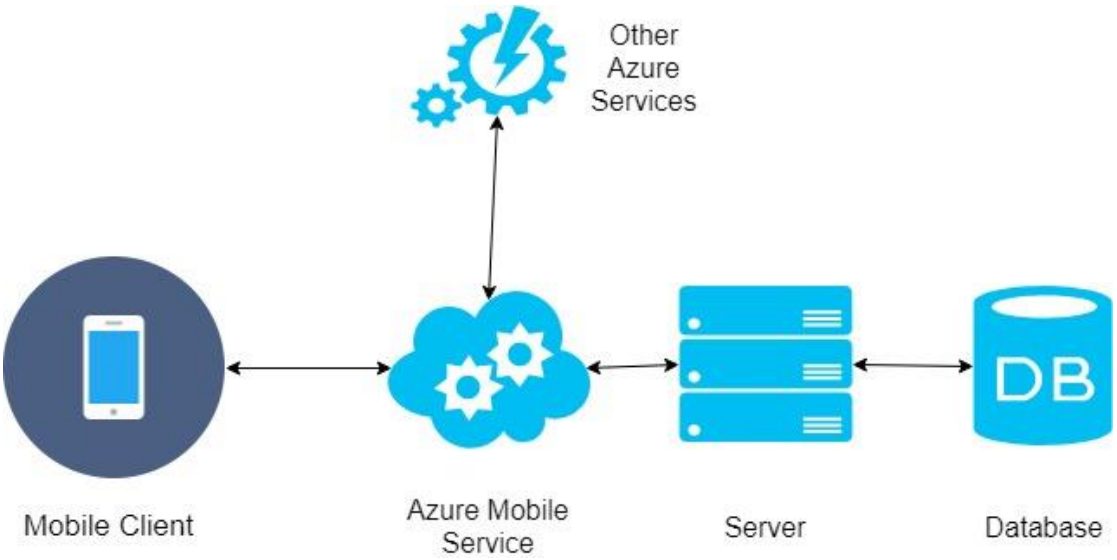
kuva 20. Tietokannan kuvaus

Kuvassa 20. on kuvattu tietokannan rakenne kaikessa yksinkertaisuudessaan. Vaikka kirjautuminen tehdään Facebookin kautta, rekisteröidään aina ensimmäisellä kirjautumisella käyttäjä kantaan, jota vasten kirjautuminen jatkossa aina tehdään Facebookin jokaiselle käyttäjälle tarjoamaa ID numeroa vastaan. Näin tarvittaessa tietyn kirjautujan käyttö voidaan estää *active* kentän avulla, jonka ollessa *false* arvoinen, viestien lähetys ei mene kantaan saakka onnistuneesti.

Jokaisella kirjoituksella on myös omistaja, vaikkakaan mitään henkilöön yhdistettävää tietoa ei lähetetä käyttäjille. Käyttäjä yhdistetään kirjoitukseen *user_id* viiteavaimen avulla. Näin viestien hallinnointi ja monitorointi helpottuu, kun jokaiseen kirjoitukseen on helposti yhdistettävissä sen kirjoittanut henkilö. *Post* oliolla on myös popularity kenttä, jonka avulla seurataan paljonko positiivisia tai negatiivisa ääniä kukin kirjoitus on saanut. Mikäli määrätty negatiivien äänien määrä ylittyy, eli kirjoituksen *popularity* arvo on liikaa miinuksella, vaihdetaan *hidden* kenttään arvo *true*. Tällöin kirjoitusta ei enää näytetä muille käyttäjille, vaikkakin se säilyy tietokannassa viestin monitoroinnin helpottamiseksi.

Tietokanta on tällä esimerkissä todella pelkistetty, eikä luultavasti riittäisi kovin pitkälle tuotannossa, koska se ei kata kuin hyvin yksinkertaisen käyttäjien ja kirjoitusten hallinnan. Kantaan voisi muun muassa jatkokehityksen kannalta tehdä oman taulun *hidden* arvon omaaville negatiivisiksi äänestetyille kirjoituksille, sekä lisätä kirjoituksiin enemmän yksilöiviä tunnisteita, jottei kirjoitusmäärän kasvaessa kaikille käyttäjille näytettäisi kaikkia kirjoituksia, vaan pelkästään heille maantieteellisesti tai kiinnostuksen mukaisesti merkittävät kirjoitukset.

5.2 tiedonkulku



Kuva 21. Sovelluksen tiedonkulku

Kuvassa 21. on esitetty kaavio siitä, miten tieto pilvipohjaisessa sovelluksessa tässä tapauksessa liikkuu. Sovellus ottaa yhteyden azuren *Mobile Service*en, joka hoitaa kaiken liikenteen serverille ja tietokantaan. Tämä helpottaa ohjelmointia suuresti, kun pyyntöjä ja kutsuja ei tarvitse itse manuaalisesti tehdä, vaan pelkkä moblile servicen kutsuminen riittää.

6 YHTEENVETO

Opinnäytetyönä toteutettu sovellus valmistui projektisuunnitelmien mukaan, vaikkakin teknologioiden osalta työn etenemisessä tuli monia yllätyksiä. sovellus kuitenkin valmistui, eikä ominaisuuksista jouduttu tinkimään. Kehittämisestä valituilla tekniikoilla jäi lopuksi hyvin ristiriitainen kuva. Vaikka teknologiat olivat Microsoftin tuotteita ja niiden takana on suuri yritys, jätti moni seikka ainakin aloittelevan sovelluskehittäjän näkökulmasta paljon kysymyksiä ilmaan.

6.1 Teknologioiden käytettävyys

Vaikka Microsoftin muut tuotteet toimivat varmasti ja helposti kuin junan vessa, jätti Xamarin todella ristiriitaisen kuvan kyseisetä alustasta. Kyseessä ei selkeästi ole ainakaan vielä kovin suosittu sovelluskehittämistapa ja ehkä juuri siitä syystä tarjolla olevaa tietoa oli todella rajoittuneesti. Myös internetissä olevat esimerkit olivat aina kovin pintapuolisia ja paikoin myös Xamarinin oma dokumentaatio vanhentunutta. Esimerkkinä tästä on Azuren *quickstart* ominaisuuteen jääneet vanhat versiot palvelinpään, sekä asiakaspään koodeista. Aikaa käytettiin paljon kehitysympäristön tutkimiseen virheilmoitusten vyöryessä konsoliin, koska oletuksena on, että suoraan Microsoftin palvelusta ladattava koodi on toimivaa ja ajan tasalla.

Xamarin.Form teknologiasta jääkin sellainen kuva, että se on kuolleena syntynyt idea, joka ei ole oikein ottanut tuulta alleen. Voi myös, että sen kehitys on niin alkuvaiheessa, että se ei ole vielä saanut suosiota jonka sen tekniset ominaisuudet ja käytettävyys ansaitsisivat, mutta missään vaiheessa sovelluskehitystä ei tule sellainen olo, että käytössä olisi teknisesti veitsenkärjellä oleva teknologia.

6.2 Vaihtoehtoiset teknologiat

Monialustainen sovelluskehittäminen on tällä hetkellä ajankohtaista ja sen ympärille on syntynyt monia ratkaisuja. Xamarin.Formsin tarjoama ratkaisu on siinä mielessä uniikki, että lopullinen koodi on natiivia koodia siinä ympäristössä, jonne sovellus muunnetaan. Kuitenkin varsinkin kevyemmissä ja yksinkertaisimmissa sovelluksissa on käytössä web teknologioihin pohjaavia ratkaisuja, joissa käytännössä kaikki sovelluskehitys tehdään web teknologioilla ja lopputuotos ajetaan ”kehyksiin”, joissa se käyttäytyy mobiililaiteella kuten normaali sovellus. Sovelluksen muuttuessa monimutkaisemmaksi, web teknologiat voivat kuitenkin osoittautua haastaviksi, sekä suorituskyky voi kärsiä. (Richardson, 2017) Molempien teknologioiden testauksen jälkeen voidaan todeta web teknologioiden helppous, joissa nykyisellään toteutettu monialustainen sovellus on todella paljon yksinkertaisempi tehdä, sekä kehitys ei vaadi lähellekään niin paljon resursseja.

6.3 Lopullinen sovellus ja jatkokehitys

Lopullisen sovellukseni toiminnallisuus vastasi täysin sitä, mitä opinnäytetyön suunnitelmassa oli kirjattu. Ylitsepääsemättömiä teknisiä vaikeuksia ei tullut vastaan, mikä oli osaksi senkin ansiota, että

toiminnallisuus pidettiin tarkoituksella yksinkertaisena. Tavoitteena ollut monialustainen ja pilvipohjainen sovellus toteutui, vaikka varsinkin paljon tarjolla olleita sovellukseen kytkettäviä pilvipalveluita jäi kokeilematta.

Jos sovellusta jatkokehitettäisiin, niin teknologiat voisi vaihtaa web ohjelmointikieliin pohjautuvaan *Ionic* alustaan, jolla koko toteutuksen voisi tehdä alusta. Vaikka Azureen kytkettyyn Xamarin.Froms sovellukseen olisi teoriassa mahdollista kytkeä vaikka minkälaisia mielenkiintoisia palveluita, kuten tekstin- tai kuvantunnistusta, ei käytännössä kyseiset ominaisuudet ole tarpeeksi houkuttavia käytön jatkamiseksi. Jos kyseessä olisi laajempi ja suorituskykykriittisempi kokonaisuus, saattaisi Xamarin.Forms silloin olla varteenotettavampi vaihtoehto, vaikkakin siinä vaiheessa tulisi arvioida, että kannattaako raskas sovellus toteuttaa suoraan natiivisovelluksena ja jokaiselle alustalle erikseen.

7 LAINATUT LÄHTEET

Analytics, Text. 2018. Text Analytics. [Online] 2018. https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics/?v=18.05.

Eronen, Heidi. 2016. [Online] 15. 3 2016. https://blog.planeetta.net/iaas-paas-saas.

Microsoft. 2018. [Online] 2018. https://docs.microsoft.com/en-us/azure/app-service/.

—. 2016. Consuming an Azure Mobile App. [Online] 20. 9 2016. https://docs.microsoft.com/en-us/xamarin/xamarin-forms/data-cloud/consuming/azure.

Osborn, John. 2000. windowsdevcenter. [Online] 01. 08 2000. http://www.windowsdevcenter.com/pub/a/oreilly/windows/news/hejlsberg_0800.html.

Richardson, Lee P. 2017. Code Project. [Online] 28. 3 2017. https://www.codeproject.com/articles/1079101/xamarin-vs-ionic-a-mobile-cross-platform-shootout.

Xamarin. Xamarin.Auth. [Online] [Viitattu:] https://github.com/xamarin/Xamarin.Auth.