Petteri Lillberg

Mobiiliapplikaation ja verkkosivuston suunnittelu ja toteutus

Opinnäytetyö
Tieto- ja viestintätekniikan koulutus

2020

Kaakkois-Suomen ammattikorkeakoulu

Kaakkois-Suomen ammattikorkeakoulu

Tekijä/Tekijät	Tutkinto	Aika
Petteri Lillberg	Insinööri (AMK)	Joulukuu 2020
Opinnäytetyön nimi		
Mobiiliapplikaation ja verkkosivuston suunnittelu ja toteutus		49 sivua 3 liitesivua
Toimeksiantaja		
Xamk GameLab		
Ohjaaja		
Lehtori Niina Mässeli		

Tiivistelmä

Opinnäytetyön tarkoituksena oli tuottaa helppokäyttöinen mobiiliapplikaatio ja verkkosivusto. Mobiiliapplikaation tulisi helpottaa laittomien kaatopaikkojen ja vieraslajien ilmoitusta. Mobiiliapplikaation kohderyhmänä ovat ulkona liikkuvat ihmiset. Lähetettyjä ilmoituksia pitäisi saada tarkastella ja hallinnoida verkkosivustossa.

Projektin alkuvaiheessa ei ollut vielä tietoa verkkosivuston tarpeesta, joten mobiiliapplikaatio suunniteltiin ja toteutettiin ensin. Verkkosivuston tarpeen myötä tuli tarve tietokannalle ja ohjelmointirajapinnalle. Tietokantaan tallennettaisiin ilmoitukset ja ohjelmointirajapinta lähettäisi mobiiliapplikaatiosta tulevat ilmoitukset tietokantaan.

Tämä opinnäytetyö tehtiin kehitystyönä ja se toteutettiin ketterän ohjelmistokehityksen periaatteita noudattaen. Toteutus keskeytyi useasti projektin tarpeiden muuttuessa, keskeytyksien aikana suunniteltiin toteutuksen seuraavaa vaihetta, jonka jälkeen toteutus aloitettiin uudestaan. Yhtenä esimerkkinä verkkosivuston lisäys projektiin, jolloin mobiiliapplikaation suunnitelma muuttui ja sen ilmoituksen lähetyksen hoitava osa piti suunnitella uudestaan.

Työn tuloksena syntyi toimiva mobiiliapplikaatio, verkkosivu ja ohjelmointirajapinta. Verkkosivulla voi tarkastella ja hallinnoida tietokannassa olevia ilmoituksia ja mobiiliapplikaatiolla voi rakentaa ilmoituksen suoraviivaisesti. Mobiiliapplikaation ainut keskeneräinen osa on datan kerääminen ja lähettäminen oikeassa muodossa ohjelmointirajapinnalle.

Kokonaisuuden kehitys tulee jatkumaan opinnäytetyön valmistumisen jälkeen niin kauan, kunnes se on valmis julkaistavaksi. Verkkosivuston lopullisen ulkonäön tulee mukailla mobiiliapplikaation teemaa. Verkkosivustolla ilmoituksen tekoon ja hallintaan tarvittavat sivut tulevat muuttumaan ja kummallakin tulee olla oma yksittäinen sivunsa. Kokonaisuutta tullaan mainostamaan joko sosiaalisen median voimin tai mahdollisen ostajan haluamalla tavalla.

Asiasanat

mobiiliapplikaatio, verkkosivusto, ohjelmointirajapinta, sovelluskehitys, älypuhelin

South-Eastern Finland University of Applied Sciences

Author (authors)	Degree	Time
Petteri Lillberg	Bachelor of Engineering	December 2020
Thesis title		
Design and implementation of a website and mobile application		49 pages 3 pages of appendices
Commissioned by		
Xamk GameLab		
Supervisor		
Niina Mässeli, Senior Lecturer		

Abstract

The objective of this thesis was to develop an easy-to-use mobile application and a website. The mobile application should facilitate the reporting of illegal dumping and invasive species. The target group of this mobile application is people moving outdoors. Submitted reports should be viewable and manageable on the website.

At the beginning of the project, there was no need for a website, so the mobile application was designed and implemented first. When the need for a website arose, the need for a database and programming interface was apparent. Notifications would be stored in the database and the programming interface would send the notifications from the mobile application to the database.

This thesis was done as a development assignment and it was implemented according to the principles set by the Agile Manifesto. As the needs of the project changed, the implementation was interrupted on numerous occasions, those interruptions were used to plan out the next stages of the implementation, after which, the implementation started anew. One example would be the addition of the website into the project, which changed the design of the mobile application, the part in charge of sending the data had to be redesigned due to that.

The results of this thesis were a working mobile application, a website and programming interface. The only part left unfinished in the mobile application, was the one that assembles the data and sends it to the application programming interface in the correct format. Reports can be managed and viewed on the website. The development of this project will continue until it is ready to be released.

Keywords

mobile application, website, API, software development, smartphone

SISÄLLYS

TERMIT JA LYHENTEET

Android	Googlen kehittämä käyttöjärjestelmä
CRUD	Create, read, update ja delete. Tietokannan yleiset toiminnot
HTML	Hypertext Markup Language, eli hypertekstin merkintäkieli. Standardoitu kuvauskieli
Käyttöliittymäkehys	Ohjelmoinnin apuväline, joka tarjoaa valmiiksi rakennettuja käyttöliittymän osia
Ohjelmointirajapinta	Määritelmä ohjelman keskenään toimivista osista
Sovelluskehys	Ohjelmoinnin apuväline, joka tarjoaa valmiiksi rakennettuja ohjelmiston osia
Verkkosovelluskehys	Ohjelmoinnin apuväline, joka tarjoaa valmiiksi rakennettuja verkkosovelluksen rakentamiseen tarvittavia osia.
XML	Extensible Markup Language, merkintäkielen standardi

1 JOHDANTO

Opinnäytetyössä on käyty läpi mobiiliapplikaation ja verkkosivuston toteutuksen ja suunnittelun vaiheet. Opinnäytetyön tarkoituksena on esitellä mobiiliapplikaation ja verkkosivuston kehitystyön vaiheita, tuloksia ja ongelmia.

Työn toimeksiantaja on Kaakkois-Suomen ammattikorkeakoulun Kotkan kampuksen GameLab. Gamelab.fi-sivuston (2019) mukaan ”GameLab on peliohjelmoinnin insinöörikoulutuksen laboratorio. Koulutus on rakennettu tiiviissä yhteistyössä peliteollisuuden ja siellä toimivien yritysten ja ammattilaisten kanssa.”

Opinnäytetyössä tuotettu mobiiliapplikaatio tulee olla helppokäyttöinen ja ilmoituksen teko ja lähettäminen tulisi olla nopeaa. Käyttäjille näkyy vain ylläpitäjien hyväksymät ilmoitukset. Käyttäjille ei näytetä verkkosivustolla muita sivuja kuin Google Maps -näkymän sisältämä sivu ja verkkosivustosta yleistä tietoa sisältäviä sivuja. Ylläpidolla tulisi taas tämän lisäksi olla työkalut ilmoituksien luomiseen, muokkaamiseen ja poistamiseen.

Opinnäytetyössä on kuusi osaa. Ensimmäisen osan aiheena on opinnäytetyön teoria, jossa käydään läpi perusteita mobiiliapplikaation ja verkkosivuston käyttöliittymien suunnittelusta, samalla käydään läpi samankaltaisia mobiiliapplikaatioita. Toisessa käydään läpi opinnäytetyössä käytetyt menetelmät, kolmannessa taas mobiiliapplikaation, verkkosivuston, ohjelmointirajapinnan ja tietokannan suunnittelua. Opinnäytetyön neljäs osa kattaa mobiiliapplikaation, verkkosivuston ja ohjelmointirajapinnan toteutuksen. Viides osa kattaa projektissa vastaan tulleet ongelmat ja niiden mahdolliset ratkaisut. Opinnäytetyön viimeisessä osassa käydään läpi työn lopputulos, jatkokehitysideat ja verrataan lopputulosta muihin sovelluksiin.

2 TEORIA

Tässä luvussa tarkastellaan vastaavanlaisia mobiiliapplikaatioita, sekä teorioita mobiiliapplikaatioiden ja verkkosivustojen käyttöliittymien suunnittelusta. Vastaavanlaisia mobiiliapplikaatioita on etsitty Google Search -hakukoneella

ja Google Play -kaupan hakuominaisuudella. Vastaavanlaisista mobiiliapplikaatioista on tehty kartoitus, jossa käydään lyhyesti läpi, mitä jokainen mobiiliapplikaatio tekee ja miten. Mobiiliapplikaatioiden ja verkkosivustojen käyttöliittymien suunnittelussa käsitellään, miten hyvä käyttöliittymä tulisi suunnitella.

2.1 Vastaavanlaisten mobiiliapplikaatioiden kartoitus

Suunnitellun applikaation kaltaista ei suunnittelun alkupuolella Suomen puolelta löytynyt. Applikaatioita haettiin kahdesta paikasta, Google Search -hakukoneesta ja Google Play -kaupasta. Termeinä hakuihin käytettiin ”jäte”, ”jäte ilmoitus”, ”kaatopaikka”, ”jätteenkeräys”, ”roska”, ”roskaaminen”, ”puhtaaksi”. Google Search -hakukoneen puolella hakuja tehtiin lisätermeillä ja ilman. Lisätermeinä käytettiin termiä ”mobiili” ja ”app”. Google Play -kaupan puolella ei lisätermejä käytetty. Kummastakaan ei hakutermeillä suunnitelmia vastaavaa applikaatiota löytynyt. Muualta maailmalta applikaatioita etsittiin termillä ”Illegal dumping” ja ”Landfills”, joilla hyvin nopeasti löytyi Report Illegal Dumping-, Clean The Creek-, Stop Wild Dumps- ja TrashOut-applikaatiot. Näiden lisäksi ”Report”-termillä löytyi bbits-kehittäjän kehittämät Love-etuliitteellä ja Report It -päätteellä nimetyt applikaatiot.

Report Illegal Dumping on Teksasin ympäristölaatukomission rahoittama Android-applikaatio, joka on tarkoitettu laittomien kaatopaikkojen ilmoittamiseen Teksasin alueella. Report Illegal Dumping -applikaatio lähettää valitukset oikean toimivallan tutkittavaksi. (East Texas Council of Governments 2020.) Clean The Creek on myöskin applikaatio, joka keskittyy laittomien kaatopaikkojen ilmoittamiseen, mutta tieto lähtee tällä kertaa kanadalaisille voittoatavoittelemattomille organisaatioille. Applikaatiossa on myös Facebook-integraatio ja näkymät, joissa ”likaiset sijainnit” näkyvät kartalla tai listassa. Facebook-integraation avulla käyttäjät voivat seurata toisten ”Kudos”-pisteitä, aktiviteettiä ja jakaa tehty ilmoitus itse Facebook-sivustolla. (Red Cherry 2017.)

TrashOut-applikaatio toimii hyvin samalla tavalla kuin Report Illegal Dumping -applikaatio. Se kuitenkin tähtää maailmanlaajuisempaan suuntaan, erittelee laittomat kaatopaikat kolmeen eri kokoluokkaan ja lisää ilmoitukseen tagit, joilla kerrotaan, minkälaista roskaa kaatopaikalla on. (TrashOut 2020.) Stop

Wild Dumps -applikaatiolla voi Bernardin (2020) mukaan nähdä ilmoitetut laittomat kaatopaikat, nähdä mitkä niistä on jo kerätty pois, nähdä ilmoituksiin liitetyt kuvatiedostot ja lähettää uusia ilmoituksia. Ilmoituksiin voi position, henkilötietojen ja kuvien lisäksi määrittää TrashOut-applikaation kaltaisesti laittoman kaatopaikan koon ja tyypin. Applikaatio on tarkoitettu käytettäväksi Ranskassa. Iso-Britannialainen bbits on kehittänyt kaupunkikohtaisesti monia eri applikaatioita, joiden suurin ero toisiinsa nähden on applikaation teema ja se, minne ilmoitus laittomasta kaatopaikasta lähtee (Google Play s.a.).

2.2 Mobiiliapplikaation käyttöliittymän suunnittelu

Android-applikaation käyttöliittymän tulisi olla selkeä. Hyvän käyttöliittymän tulee kommunikoida käyttäjän kanssa ja käyttäjälle tulisi olla selvää, mitä näkymä käyttäjältä haluaa. Liian selkeä käyttöliittymän ei kuitenkaan tule olla, eikä käyttöliittymän jokaista osaa tarvitse selittää perin pohjin, sillä käyttäjän näkökulmasta liiallinen neuvominen voi aiheuttaa ärsytystä. Tehokkaan käyttöliittymän pitää tasapainottaa ytimekkyys ja selvyys. Hyvin suunnitellussa käyttöliittymässä käyttäjä tietää, että plus-ikoni musiikkia toistavassa applikaatioissa tarkoittaa uuden musiikkikappaleen lisäämistä soittolistaan ilman että sitä tarvitsee turhalla tekstillä selittää. (Thornsby 2016, 12.)

Applikaation sisällöllä ei ole mitään väliä, jollei applikaation käyttöliittymä reagoi käyttäjän kosketuksiin kunnolla ja hidastelee. Kukaan ei sitä silloin tule käyttämään. Käyttöliittymän tulisi myös reagoida painikkeiden painalluksiin; jos käyttäjä täyttää lomakkeen ja näpäyttää ”Lähetä”-painiketta, applikaation tulisi vastata siihen jotenkin muuten kuin käyttäjää hämmentävällä hiljaisuudella. Käyttäjälle riittää vastaukseksi yksinkertainen ponnahdusikkuna ja lyhyt asiasta ilmoittava lause. Loppujen lopuksi käyttöliittymän tulee kuitenkin olla kaunis. Käyttöliittymän painikkeiden tulee olla järjestyksessä, eikä elementtien tule olla vinossa. Näillä kaikilla asioilla on väliä, koska emme elä täydellisessä maailmassa. (Thornsby 2016, 12–13.)

2.3 Verkkosivuston käyttöliittymän suunnittelu

Verkkosivuston muotoilun tulee olla yhdenmukainen ja jokaisen tehdyn elementin tulisi noudattaa verkkosivuston yhtenäistä teemaa, jonka olet itse määrittänyt. Käyttäjät muistavat yksityiskohdat, oli se sitten tietoista tai ei, he tunnistavat linkin sen väristä ja muodosta. (Hong 2018, 55.)

Verkkosivuston visuaalinen puoli ei ole ainoa asia, jonka pitää olla yhdenmukainen, vaan myös verkkosivuston sisällön. Verkkosivustolla käytetyn teemoituksen tulisi kuvastaa brändiä. Jos kyseessä on yrityksen verkkosivusto, joka myy tuotteita toisille yrityksille, verkkosivuston sisällön tulisi silloin sekä tuntua että näyttää ammattimaiselta. (Hong 2018, 56.)

Toimintojen on oltava yhdenmukaisia koko verkkosivustolla ja verkkosivuston elementtien pitää aina reagoida samalla tavalla. Jos esimerkiksi avataan Dropbox-sivustolla tiedosto, niin painike, josta mennään takaisin päin, on aina ylävasemmalla. Painikkeen yhdenmukaisuuden takia käyttäjän ei tarvitse opetella käyttöliittymää aina uudestaan. (Hong 2018, 56.)

3 KÄYTETYT MENETELMÄT

Tässä luvussa käsitellään projektissa käytetyt menetelmät. Käytetyistä ohjelmointikielistä, verkkoselaimista, palveluista ja kehyksistä kerrotaan yleisesti mitä ne ovat ja mihin niitä on projektissa käytetty. Jos menetelmän käytön aikana on tullut vastaan ongelmia, ne käydään läpi lyhyesti.

3.1 Kehitystyökalut

C#-ohjelmointikieltä lukuun ottamatta melkein kaikki projektissa käytetyt menetelmät, palvelut, rakenteet ja tekniikat olivat projektin toteuttajalle uutta, joten projektin työkaluiksi haluttiin mahdollisimman tutut sovellukset. Näihin kuului Visual Studio Community, Microsoft SQL Server Management Studio ja Postman.

Alemmissa luvuissa käydään jokainen näistä kolmesta läpi; selitetään mikä kukin on ja mihin niitä on käytetty. Samalla kerrotaan myös, miten kyseessä oleva sovellus tai palvelu on projektin aikana toiminut.

Microsoft Visual Studio

Microsoft Visual Studio on ohjelmankehitysympäristö, joka tukee monia ohjelmointikieliä. Projektissa käytettiin pääasiassa Visual Studion Community julkaisua. Yksittäinen kehittäjä voi kehittää ja testata Visual Studio Community -kehitysympäristön kanssa ohjelmiaan, eikä sen lisenssi kiellä kehitettävän ohjelmiston myymistä eteenpäin.

Visual Studio Community toimi projektissa kaikin puolin loistavasti, vaikka 16.7.0-versio ei mobiilisovellusta saanutkaan enää käännettyä. Tätä varten vaihdettiin hetkeksi Visual Studio Enterprise -julkaisuun, koska Microsoft ei tarjonnut latauksia aikaisemmille Visual Studio Community -versioille.

Microsoft SQL Server Management Studio 18

Microsoft SQL Server Management Studio on sovellus, joka julkaistiin Microsoftin SQL Server 2005 -version kanssa, sitä käytetään Microsoft SQL Server -tyyppisten tietokantojen konfigurointiin ja hallintaan.

Projektissa Microsoft SQL Server Management Studio 18:aa käytettiin pääasiassa alustavan Azure SQL Database -palvelun tietokannan toteuttamiseen ja suunnitteluun. Sovelluksen käyttö kuitenkin jäi vähemmälle, kun projektissa siirryttiin hyödyntämään Entity Framework Core -tiedonsiirtotekniikan Code First -tapaa rakentaa SQL-tietokanta.

Sovelluksessa havaittiin yllättävän yleistä epävakautta ja hitautta operaatioissa, joista Visual Studio -kehitysympäristön Server Explorer -moduuli suoriutui selvästi nopeammin. Vastaan tulleet ongelmat johtivat uusien vaihtoehtojen etsimiseen ja lopulta aiemmin mainitun Code First -tavan opetteluun. Microsoft SQL Server Management Studio 18 -sovelluksen käyttö väheni sen mukaan, mitä pidemmälle projekti eteni. Projektin loppupuolella sovellusta ei käytetty muuhun, kuin tietokannan pikaiseen tarkasteluun.

Postman

Postman on yhteistyöhön keskittyvä alusta, joka on tarkoitettu ohjelmointirajapintojen kehitykseen (Postman 2020). Alustalla voi luoda, jakaa, testata ja dokumentoida ohjelmointirajapintoja.

Postman-alustaa hyödynnettiin projektiin liittyvän nettisivun ohjelmointirajapinnan POST-, GET- ja DELETE-pyyntöjen testaamiseen, kehittämiseen, vikojen etsintään ja niiden toimivuuden varmistamiseen.

3.2 C#

C# on tyyppiturvallinen olioperustainen ohjelmointikieli, jonka avulla kehittäjät voivat rakentaa turvallisia ja vankkoja sovelluksia, jotka toimivat .NET-ekosysteemissä (Microsoft 2015). C#-ohjelmointikielellä voit luoda Windows-asiakassovelluksia, XML-verkkopalveluja, hajautettuja komponentteja, asiakas-palvelinsovelluksia, tietokantasovelluksia ja paljon muuta (Microsoft 2015).

Projekti toteutettiin pääasiassa C#-ohjelmointikielellä. C#-ohjelmointikielen dokumentaatio on laaja ja se on projektissa käytettävien Xamarin.Forms-käyttöliittymäkehyksen ja ASP.NET Core-sovelluskehyksen yksi tuettu kieli.

3.3 JavaScript

JavaScript on kevyt, tulkattu ja just-in-time-käännetty ohjelmointikieli. Vaikka se tunnetaan parhaiten verkkosivujen skriptikielenä, myös muut kuin selainympäristöt käyttävät sitä, esimerkiksi Node.js, Apache CouchDB ja Adobe Acrobat. JavaScript on prototyyppipohjainen, moniparadigmainen, yksisäikeinen, dynaaminen kieli, joka tukee oliokeskeisiä, imperatiivisia ja deklaratiivisia (esim. toiminnallinen ohjelmointi) tyylejä. (MDN Contributors 2020).

JavaScript-ohjelmointikieltä käytettiin nettisivun Google Maps -kartan näyttävää sivua toteutettaessa. Toteutuksen aikana opeteltiin JavaScript-ohjelmointikieltä sen verran, että tarvittavat toiminnot saatiin toteutettua. Oppimiseen auttoi sen samankaltaisuus C#- ja C++-ohjelmointikielien kanssa.

3.4 Ketterä ohjelmistokehitys

Ketterä ohjelmistokehitys on joukko periaatteita, joita käytetään ohjelmistotuotantoprojekteissa. Agile Alliance -sivuston (2001) mukaan ketterässä ohjelmistokehityksessä arvostetaan yksilöitä enemmän kuin menetelmiä ja työkaluja. Toimivaa ohjelmistoa taas arvostetaan enemmän kuin kattavaa dokumentaatiota. Asiakasyhteistyötä arvostetaan enemmän kuin sopimusneuvotteluja. Viimeisenä ketterässä ohjelmistokehityksessä arvostetaan vastaamista muutokseen enemmän kuin pitäytymistä suunnitelmassa. Ketterän ohjelmistokehityksen periaatteiden mukaan jälkimmäisillä asioillakin on arvoa, mutta ensiksi mainittuja arvostetaan enemmän.

Ketterän ohjelmistokehityksen käytetyimpiin menetelmiin kuuluu Measeyn ym. (2015, 125–164) mukaan mm. SCRUM, Extreme Programming (XP) ja Kanban. Measey ym. (2015) toteaa SCRUM-menetelmän perustuvan empiirisen prosessikontrollin kolmeen pylvääseen, avoimuuteen, tarkasteluun ja sopeutumiseen (Schwaberin & Sutherlandin s.a., mukaan). Measy ym. (2015) toteaa Extreme Programming -menetelmän ottavan toimivaksi todistetut käytännöt ja soveltaa niitä puhtaimmassa tai äärimmäisimmässä muodossa. Siinä ihmiset kehittävät pääasiassa ohjelmistoa, eivätkä vain ohjelmoi (Beck 2004). Kanban-menetelmä on lähestymistapa palvelujen jatkuvaan parantamiseen. Sillä korostetaan työn sujuvuutta ja sen nopeaa kulkua. Kanban-menetelmää käytetään tietotyön palvelujen toimittamiseen, työnkulkuun ja kaikkeen luovaan. (Measy ym. 2015, Andersonin 2010, mukaan.)

Projektissa tuli vastaan paljon uusia asioita, joten ketterän ohjelmistokehityksen mukaan toteutettiin erimittaisia pyrähdyksiä. Pyrähdykset jaettiin projektisuunnitteluun, vaatimusanalyysiin, ohjelmistosuunnitteluun, ohjelmointiin, testaukseen ja dokumentaatioon, jotka ovat kooders.fi-blogin (2018) mukaan julkaisemiseen tarvittavat tehtävät.

3.5 Kehykset

Työssä käytettiin kahdenlaista kehystä: sovelluskehystä ja käyttöliittymäkehystä. Verkkosivusto toteutettiin ASP.NET Core -sovelluskehyksellä ja mobiiliapplikaatio Xamarin.Forms-käyttöliittymäkehyksellä.

Xamarin.Forms

Xamarin.Forms on Microsoftin (2020a) dokumentaation mukaan avoimen lähdekoodin käyttöliittymäkehys, joka mahdollistaa Xamarin.Android-, Xamarin.iOS- ja Windows-applikaatioiden rakentamisen samalla koodialustalla.

Xamarin.Forms-käyttöliittymäkehyksen valinta perustui harjoittelun toimeksiantajan suositukseen ja projektin kehittäjän kokeiluihin. Microsoftin hyvä dokumentaatio ja työlleen omistautuneet ja yhteisöissä aktiivisesti mukana olevat Xamarin.Forms-käyttöliittymäkehyksen kehittäjät auttoivat projektin edistymistä todella paljon.

Xamarin.Forms-käyttöliittymäkehyksen takana on tuttu ja turvallinen C#-ohjelmointikieli ja käyttöliittymän puolella taas käytetään XAML-merkintäkieltä. Käyttöliittymän voi myös toteuttaa C#-ohjelmointikielellä, mutta itse projektissa käyttöliittymä toteutettiin XAML-merkintäkielellä ja kaikki muu C#-ohjelmointikielellä.

ASP.NET Razor

ASP.NET Razor on learnrazorpages.com-sivuston (2019) mukaan sivukeskeinen sovelluskehys dynaamisten ja tieto-ohjautuvien verkkosivustojen rakentamiseen. Järjestelmäriippumatonta ASP.NET Razor -sovelluskehystä voi käyttää ja kehittää Windows-, Unix- ja Mac-käyttöjärjestelmissä. Razor on sisältynyt ASP.NET Core -sovelluskehykseen versiosta 2.0 eteenpäin. .NET Core on saatavissa ilmaiseksi joko ohjelmistokehityspakettina tai Runtime-versiona.

Microsoftin (2020b) dokumentaation mukaan Razor-merkintäsyntaksilla sisällytetään palvelinpohjaista koodia verkkosivuihin. Razor-syntaksi koostuu Razor-merkinnöistä, C#-ohjelmointikielestä ja HTML-merkintäkielestä. Kuvassa 1 näkyy esimerkki C#-koodista, joka ajetaan HTML-merkintäkielen <td>-tagien sisällä Razor Pages -sivulla.

```
<td>
    @{
        if (item.Data.Length > 150) {
            item.Data = item.Data.Substring(0, 150) + "...";
        }
    }
    @item.Description
</td>
```

Kuva 1. Esimerkki C#-koodista Razor Pages -sivun sisällä

Projektin verkkosivuston rakentamisen alustaksi valikoitui ASP.NET Core, koska ASP.NET Core -sovelluskehys vaikutti vakaalta ja varmalta vaihtoehdolta. Projektin aikana ASP.NET Core -sovelluskehys ei aiheuttanut ongelmia epävakauden tai hitauden kanssa ja osoittautui hyvin vakaaksi. Vanhan ja samankaltaisen ASP.NET MVC -verkkosovelluskehyksen ohjeita löytyi reilusti ja ne sekoittuivat samankaltaisuuden johdosta hyvin ASP.NET Razor -sovelluskehyksen hakutulosten sekaan. ASP.NET MVC -verkkosovelluskehyksen kehittämisen lopetuksen myötä hakutulokset ASP.NET Core -sovelluskehykselle ja sen komponenteille tulevat yleistymään.

3.6 Visual Studio App Center

Visual Studio App Center on palvelu, joka tarjoaa työkaluja sovellusten testaamiseen, rakentamiseen, jakamiseen, analysointiin ja diagnosointiin. Projektissa ei App Center -palvelusta käytetty muuta kuin App Center Diagnostics -palvelua.

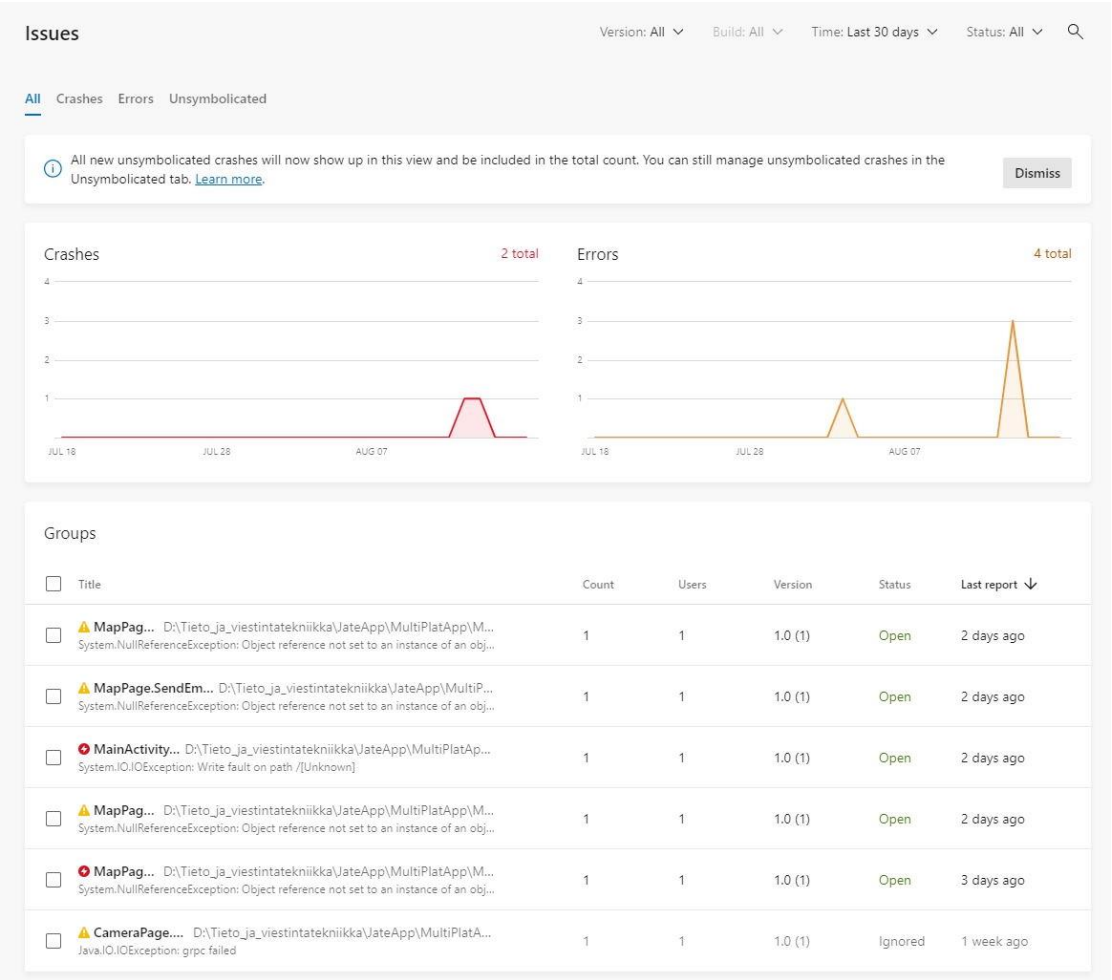
Projektin aikana virhetilanteet oli helppo saada selville, kun itse testilaite oli tietokoneeseen yhdistetty johdon kanssa. Näin ei kuitenkaan tapahdu muuta kuin testiympäristössä. Testiympäristön ulkopuolelle kaivattiin toisenlaista ratkaisua. Ratkaisun tulisi tallentaa virhetilanteet ja lähettää ne langattomasti eteenpäin. Tässä päädyttiin Visual Studio App Center -palveluun, joka oli helppo sisällyttää projektiin ja toimi vaivattomasti.

Visual Studio App Center Diagnostics on Microsoftin (2019) dokumentaation mukaan pilvipalvelu, joka auttaa kehittäjiä valvomaan sovelluksen toimivuutta ja tarvittavan datan lähettämisessä eteenpäin applikaation virhetilanteessa, oli kyse sitten testauksessa tehtävästä bugien jahtaamisesta tai julkaisun jälkeisestä oikeasta virhetilanteesta.

Visual Studio App Center -palvelu kerää try/catch-lohkojen avulla tiedot virheestä tai kaatumisesta. Crashes.TrackError()-funktio kaappaa tiedot (Kuva 2) tapahtuneesta kaatumisesta tai virhetilanteesta ja lähettää sen eteenpäin App Center -palveluun, jossa se näkyy kuvan 3 kaltaisessa listassa. Listasta voi valita haluamansa raportin ja tarkastella App Center -palvelun tallentamia tietoja.

```
} catch (Exception ex) {
    await DisplayAlert(Constants.errorProcessingTitle,
        Constants.errorImageProcessing,
        Constants.alertConfirm).ConfigureAwait(false);
    var properties = new Dictionary<string, string>
    {
        { "Category", Constants.errorCategoryImage },
        { "Exception type", ex.GetType().ToString() }
    };
    Crashes.TrackError(ex, properties);
}
```

Kuva 2. Try/catch-lohkon catch-osassa ilmoituksen näyttäminen ja virhetilanteen tallennus



Kuva 3. Visual Studio App Center -palvelun lista virhe- ja kaatumisraporteista

3.7 Microsoft Azure

Microsoft Azure on Sumologic-sivuston (2019) mukaan pilvialusta sovellusten ja palvelujen rakentamiseen, julkaisemiseen ja hallinnointiin. Azure-alustalla on saatavilla monia eri tuotteita, mutta projektissa ei muita tuotteita tai palveluja käytetty kuin Azure SQL -tietokantaa ja Azure Storage -palvelua.

Kaakkois-Suomen ammattikorkeakoulun opiskelijat saavat Microsoft Azure -palveluun 100 dollarin arvosta krediittejä ja vuoden aikaa (Kuva 4) sinne rekisteröitymisestä käyttää ne. Päätös tietokantajärjestelmän valintaan perustui Microsoft-palveluiden laajuuteen ja kattavaan dokumentaatioon. Microsoft Azure Storage -palvelun käyttöä ei edes alkupuolella suunniteltu, mutta verkkosivuston kehityksen aikana se kuitenkin osoittautui tarvittavaksi.

Kuva 4. Azure for Students -tilauksen saldo projektin kehittäjän omalla tilillä

Microsoft Azure SQL Database

Microsoft Azure SQL Database on Microsoftin (2019) dokumentaation mukaan hallittu tietokantamoottori, joka on alustana toteutettu palvelu. Se hoitaa suurimman osan tietokannan hallintatoiminnoista ilman käyttäjän osallistumista.

Azure SQL Database käyttää aina uusinta SQL Server -tietokantamoottoria ja päivitettyä käyttöjärjestelmää 99,99 prosentin saatavuudella.

Palvelutasoja on saatavilla Basic, Standard ja Premium. Projektiin valittiin Azure for Student -tilauksen omaaville ilmainen Standard S0 -tason tilaus kymmenellä DT:lla. DT tarkoittaa Database Transaction -yksikköä, joka taas kuvastaa Microsoftin (2020c) mukaan sekoitettua prosessorin, muistin, tietokannan luku- ja kirjoitusmittaa.

Microsoft Azure SQL Database -palvelua käytettiin projektissa mobiiliapplikaation lähettämien raporttien tallennukseen, josta verkkosivusto hakee näytettävän tiedon käyttäjälle.

Microsoft Azure Storage

Microsoftin (2020d) dokumentaation mukaan Microsoft Azure Storage on Microsoftin oma pilvivarastointiratkaisu moderneihin tietojen tallennusskenaarioihin. Sen ydintallennuspalvelut tarjoavat skaalautuvan objektivaraston dataobjekteille, levytallennuksen Azure-virtuaalikoneille, tiedostojärjestelmäpalvelun pilvelle, viestipalvelun ja NoSQL-varaston.

Microsoft Azure Storage -palvelun ydintallennuspalveluihin kuuluu projektissa käytetty Azure Blob Storage, joka on Microsoftin (2020e) mukaan optimoitu jäsentymättömän datan tallennukseen ja suunniteltu mm. kuvien ja dokumenttien näyttämiseen verkkosivulla. Nämä seikat ja se, että Azure for Students -tilauksen omistajat saavat viisi gigatavua ilmaista tallennustilaa Blob Storage -palveluun, tekivät palvelusta projektiin sopivan valinnan.

Projektissa palvelua käytettiin mobiiliapplikaatiosta lähetettyjen kuvien säilyttämiseen. Palvelun kautta kuvatiedostot näytettiin käyttäjille verkkosivuston karttanäkymässä, pinnin inforuudussa.

3.8 GitHub

GitHub on kehitysalusta, jossa voi github.com-sivuston (s.a.) mukaan palveluun voi tallentaa omaa koodia, tarkistella koodia, hallita projekteja ja luoda

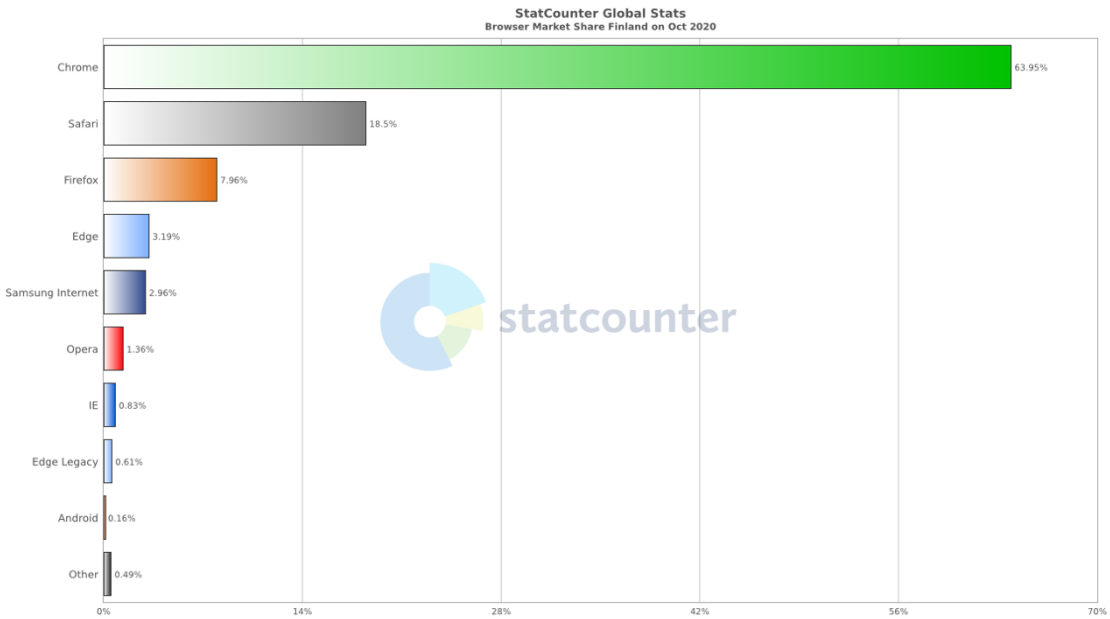
sovelluksia 50 miljoonan muun kehittäjän kanssa. GitHub käyttää hyväkseen hajautettua versionhallintajärjestelmää nimeltä git.

Projektin kehityksessä käytettiin GitHub-kehitysalustaa tehtyjen muutoksien seuraamiseen ja verkkosivuston ja mobiiliapplikaation kuuluvien projektitiedostojen helppoon varmuuskopiointiin. Visual Studio Team Explorer -liitännäisen avulla kummatkin projektit oli helppo lisätä GitHub-kehitysalustalle, josta jokaisesta kehittäjän lisäämästä muutoksesta näkyy muuttuneet tiedostot, mikä niissä muuttui ja muutokseen lisätty kommentti.

GitHub on toiminut projektissa Visual Studio Team Explorerin kanssa enimmäkseen moitteitta. Projektin GitHub-arkisto piti korjata kerran, mutta aikaa siihen ei puolta tuntia enempää tuhlaantunut. Muita ongelmia itse git-versionhallintajärjestelmän tai GitHub-kehitysalustan kanssa ei tullut vastaan.

3.9 Verkkoselaimet

Verkkosivustoa testattiin kahdella selaimella, Google Chromella ja Mozilla Firefoxilla. StatCounter-sivuston (2020) vuoden 2020 marraskuun statistiikkojen (Kuva 5) mukaan Suomessa Chrome-selainta käyttää 63,95 % väestöstä ja Mozilla Firefox -selainta 7,96 % väestöstä, Safaria käyttää taas noin 18,5 % väestöstä.



Kuva 5. Suomen verkkoselainten käyttö StatCounter-sivuston (2020) mukaan

StatCounter-sivuston (2020) antamien statistiikkojen mukaan projektin testauksen aikana vaihdeltiin Mozilla Firefox- ja Google Chrome -selaimien välillä. Google Chrome -selainta käytettiin verkkosivuston testauksessa noin 3/4 ajasta, kun taas Mozilla Firefox -selaimella vain pääasiassa testattiin, että sillä selaimella sivu näkyy ja toimii oikein.

Google Chrome -selain tuki kaikkia asioita mitä projektissa haluttiinkin käyttää. Firefox-selaimelta taas puuttui verkkosivustolla käytetyn, HTML-merkintäkielen input-elementtiin kuuluvan, datetime-local-käyttöliittymäkontrollin tuki. Aikarajoitusten takia mobiilisovellusta ja verkkosivustoa ei testattu Applen mobiililaitteilla ja Safari-selaimella.

4 SUUNNITTELU

Seuraavissa luvuissa käydään läpi asioita, jotka liittyvät mobiiliapplikaation, verkkosivuston, ohjelmointirajapinnan ja tietokannan suunnitteluun.

Projektissa suunniteltiin ensin mobiiliapplikaatio, sitten suunniteltiin verkkosivusto ja tietokanta, ja lopuksi ohjelmointirajapinta. Projektissa tuli vastaan paljon uusia asioita, joten toteutus keskeytyi hyvin useasti ja vaihtui takaisin suunnitteluun. Asian avaaminen paperille auttoi ongelmien ratkomisessa, nopeutti suunnittelua ja auttoi näkemään vielä puuttuvat osat.

Projektin aikana huomattiin, että uusia asioita tulee koko ajan vastaan ja kaikkea ei pystynyt kerralla suunnitella, joten projektissa otettiin käyttöön joitakin ketterän kehityksen periaatteita. Toimiva sovellus laitettiin dokumentaation edelle ja reagoitiin enemmän muutoksiin kuin suunnitelmien noudattamiseen.

Isommissa projekteissa käytetään perinteisesti systemaattisempaa ja dokumenttivetoista menetelmää, jossa asetetaan tarkat spesifikaatiot ja kaaviot toteutettavalle ohjelmistolle. Kaavioita mallinnetaan yleensä UML-mallinnuskielellä. Mallinnuskieli koostuu merkinnöistä, joiden tarkoitus on mallintaa ohjelmistojärjestelmän kaikki osat: data, tila, käytös, kommunikaatio, tarjotut palvelut, ajoitusominaisuudet ja käyttöönoton määritykset. Jokainen notaatio muo-

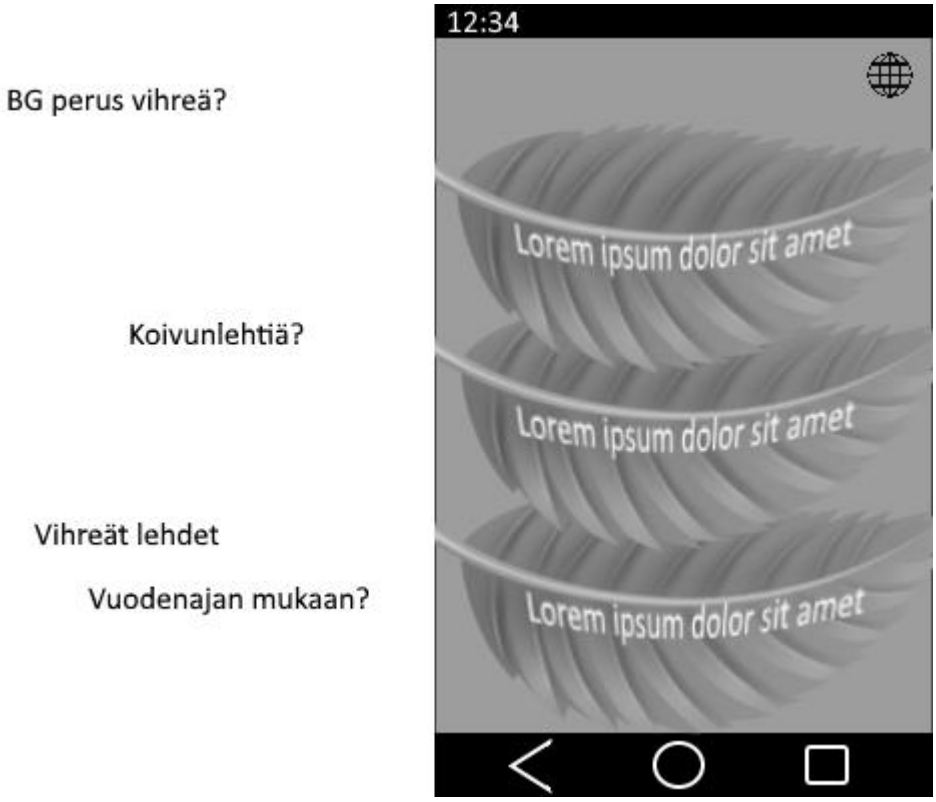
dostaa oman monimutkaisen kielensä. Notaatiot liittyvät toisiinsa ja ovat riippuvaisia toisistansa, tämä tekee yhtenäisen semantiikan tarjoamisen haastavaksi. (Lano 2009.)

4.1 Mobiiliapplikaation suunnittelu

Mobiiliapplikaation suunnittelun päämääränä oli saada aikaan helppokäyttöinen ja suoraviivainen applikaatio, jonka avulla laittoman kaatopaikan ja vieraslajin ilmoittaminen mobiililaitteella olisi nopeampaa ja helpompaa.

Ajateltiin, että applikaation päämääräinen käyttäjä olisi henkilö, joka olisi lenkillä, matkalla johonkin muualle tai muuten vain kiireinen. Käyttöliittymä piti suunnitella siten, että ilmoituksen tekemisessä ei menisi paljon aikaa, eikä käyttäjä hukkaisi tietään ilmoituksen tekemisen aikana minkäänlaisiin erillisiin näkymiin. Suoraviivaisuus ja selkeys oli prioriteettien kärkipäässä. Tärkeimpänä suoraviivaisuus, toisena selkeys, kolmantena muokattavuus ja neljäntenä kauneus.

Mobiilisovelluksen lopullisen käyttöliittymän teemaa ei kovin paljoa projektin aikana suunniteltu. Mobiiliapplikaation toteuttamisen kannalta päätettiin kuitenkin, että olisi hyvä olla jonkinlainen suunnitelma mobiiliapplikaation lopullisesta teemasta. Mobiiliapplikaation luontokeskeisyyden takia päädyttiin kuvan 6 kaltaiseen nopeaan luonnokseen. Pääväriksi valittiin teemalle vihreä ja painikkeille ja muille käyttöliittymän elementeille valittiin simppeli luontoteema.



Kuva 6. Mobiiliapplikaation käyttöliittymän nopea luonnos

Sarkkisen (2019) mukaan ilmoituksessa tulee olla sijainti, kuvaus tai kuva jätteistä ja ilmoittajan tiedot, näiden lisäksi olisi hyvä olla tieto siitä, haluaako ilmoittaja vastauksen ilmoitukseensa. Laittoman kaatopaikan ja vieraslajien ilmoittamisen yhteydessä ei tarvita muuta kuin ilmoittajan sähköpostiosoitteen tai jokin muun osoitteen, josta ilmoittajaan saa yhteyden. Ilmoituksen tekemisen asuinkiinteistölle kerääntyneistä jätteistä tarvitsee kuitenkin ilmoittajan nimen.

Ennen verkkosivuston aloitusta, applikaation osat päätettiin jakaa neljään osaan: ilmoituksen tyypin valintaan, nimen lisäämiseen, kuvaan ja/tai kuvaukseen ja sijainnin valintaan. Käyttäjälle näytettiin yksi näkymä kerrallaan, eikä käyttäjän tulisi pystyä etenemään seuraavaan näkymään, jollei tarvittavia tietoja ole lisätty.

Ensin käyttäjän tulisi valita ilmoituksen tyyppi, joten applikaation tulisi näyttää jonkinlainen näkymä, jossa näkyy kaikki mahdolliset valinnat. Seuraavassa näkymässä haluttaisiin ilmoitukseen liitettävä kuva tai kuvaus. Kuvia ei tulisi saada lisätä puhelimen muistista, vaan kuvat pitäisi ottaa paikan päällä. Jos

kyseessä on kuitenkin asuinkiinteistölle kerääntyneestä jätteestä, niin käyttäjälle tulisi ilmoituksen tyypin valinnan jälkeen näyttää näkymä, johon käyttäjä voi kirjoittaa nimensä. Viimeisessä näkymässä joko tarkastetaan kuvista saadut sijainnit, tarkastetaan puhelimelta saatu sijainti tai lisätään sijainti itse. Koska alustana on Android-käyttöjärjestelmä, niin pitää käyttäjältä kysyä sopivissa kohdissa lupaa tiedostojärjestelmän, paikannuksen ja kameran käyttöön. Mobiiliapplikaation ilmoituksenlähetysprosessin tulisi loppujen lopuksi näyttää liitteen 1 kaltaiselta.

Suunnitelmana oli, että ilmoitus lähetettäisiin karttanäkymän jälkeen sähköpostilla ympäristökeskukselle. Mobiiliapplikaatio lisäisi sähköpostiin kerätyt tiedot, ja käyttäjän ainoaksi työksi jäisi ilmoituksen tietojen tarkistus ja itse sähköpostin lähetys. Sähköpostin voisi myös tallentaa myöhempää ajankohtaa varten, jollei käyttäjän puhelimessa juuri tällä hetkellä ole kunnollista yhteyttä. Sähköpostin huonona puolena oli, että kukaan muu ei näkisi näitä ilmoituksia. Kun projektiin lisättiin verkkosivusto, mobiiliapplikaation suunnitelma ei muuten muuttunut kuin ilmoituksen lähettämisen kannalta.

4.2 Verkkosivuston suunnittelu

Verkkosivuston suunnittelu aloitettiin kartoittamalla mahdolliset sovellus- ja käyttöliittymäkehykset, joilla voisi verkkosivuston toteuttaa. Kandidaatteina oli kuitenkin loppujen lopuksi vain CakePHP, Express.js, ASP.Net Core ja Ruby on Rails. ASP.Net Core valittiin verkkosivuston tekoon, koska mobiiliapplikaation teon aikana Microsoftin dokumentaatio osoittautui kelvolliseksi ja C# oli jälleen vaihtoehtona.

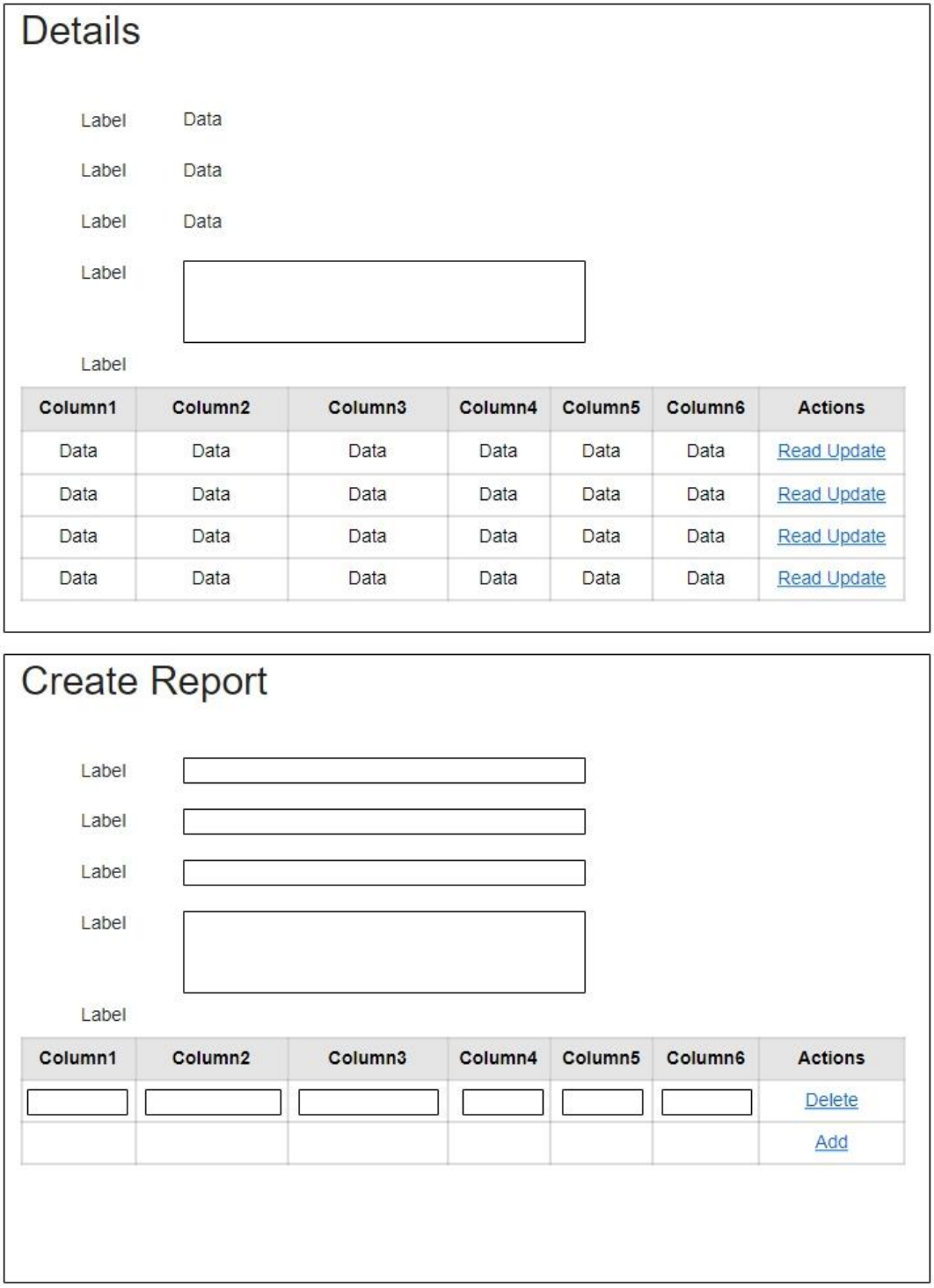
Verkkosivuston päämääräisenä tarkoituksena oli näyttää muille käyttäjille tietokantaan lähetetyt ilmoitukset. Verkkosivuston vierailijoille ilmoituksesta tulisi näyttää vain hyväksytyn ilmoituksen kuvaus, kuvat, päivämäärä ja paikka, kun taas ylläpitäjä tai vastaavanlaisen käyttäjätilin haltija näkisi ilmoituksen kaikki tiedot ja hyväksymättömät ilmoitukset.

Vierailijoille ilmoitukset näytetään vain karttanäkymän kautta. Vierailija voi vain tarkastella ilmoitusta, muut toiminnot kuuluvat rekisteröityneille käyttäjille.

Verkkosivuston suunnitteluun lisättiin karttanäkymän lisäksi neljä muuta näkymää, joissa rekisteröityneet käyttäjät voivat hallinnoida ja tarkastella tietokannassa olevia ilmoituksia; näihin ei sivustolla vierailevan käyttäjän tulisi päästä käsiksi millään tavalla.

Ilmoituksien tarkastelua varten käyttäjällä piti olla mahdollisuus suodattaa ilmoituksia sähköpostin, ID-numeron, kategorian ja päivämäärän mukaan ja lajitella päivämäärän, sähköpostin ja kategorian mukaan. Samalla suunnitelmaan lisättiin sivunumerointi, ettei mahdollisesti suuri määrä ilmoituksia näkyisi kaikki samalla sivulla. Ilmoituksia tulisi olla mahdollisuus lajitella ja suodattaa kaikilla mahdollisilla kombinaatioilla ja suodatus- ja lajitteluasetukset tulisi säilyttää, vaikka käyttäjä siirtyisi sivulta toiselle.

Käyttöliittymän suunnittelussa ei kauneuteen keskitytty, vaan haluttiin selvä ja ymmärrettävä käyttöliittymä, josta olisi helppo hallinnoida tietokannassa olevia ilmoituksia. Varhaisessa suunnitteluvaiheessa olevan verkkosivuston ilmoitusten ”Read”- ja ”Create”-sivustot suunniteltiin näyttävän kuvan 7 kaltaisilta. Aluksi ajateltiin, että jokaisella ilmoituksen käyttämällä tietokannan taulukolla tulisi olla omat CRUD-toiminnot, mutta projektin edetessä se kuitenkin osoittautui turhaksi ja suunnitelma muuttui niin, että ainakin kaikkien taulukoiden ”Read”- ja ”Create”-toiminnot olisi yhdistetty yhdeksi sivuksi, ettei käyttäjän tarvitse turhaan seikkailla neljässä eri näkymässä nähdäkseen ilmoituksen kaikki tiedot.



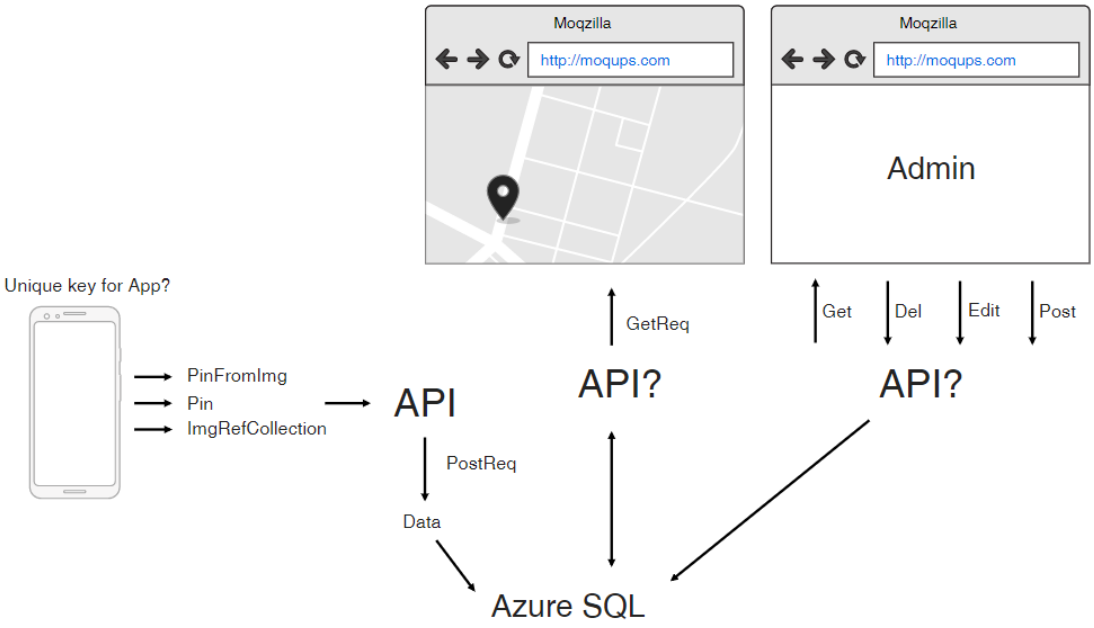
Kuva 7. Varhainen luonnos verkkosivuston Report Details- ja Create Report -sivuista

4.3 Ohjelmointirajapinnan suunnittelu

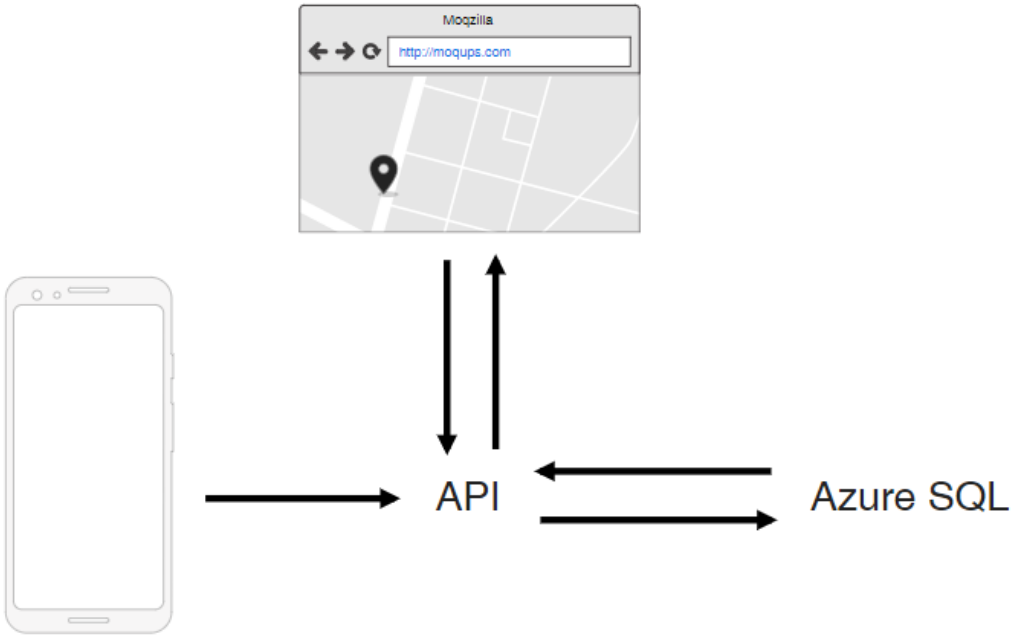
Verkkosivuston suunnittelun aikana havaittiin, että mobiiliapplikaatio tarvitsee jonkinlaisen välikäden ilmoituksen tietojen lähettämiseen tietokantaan. Ohjelmointirajapinnat eivät ennestään olleet tuttuja muuten kuin nimellisesti. Suunnittelu täten aloitettiin selvittämällä, miten ohjelmointirajapinnat toimivat, miten

sellainen toteutetaan ja mitä mahdollisuuksia ohjelmistorajapinnan toteuttamiseen ASP.NET Core -sovelluskehyksen puolella on. Huomattiin nopeasti, että ASP.NET Core -sovelluskehys sisältää omat työkalut ohjelmointirajapinnan toteuttamiseen.

Ohjelmointirajapinnan toteuttaminen ensimmäistä kertaa ei ollut helppoa. Ensin selvitettiin, miten tulevan ohjelmointirajapinnan pitäisi toimia verkkosivuston, mobiiliapplikaation ja tietokannan kanssa. Ohjelmointirajapinnan suunnittelun aikana tehtiin skitsejä, joiden piti selventää sen toimintaa muiden projektin komponenttien kanssa, esimerkkeinä kuva 8 ja kuva 9.



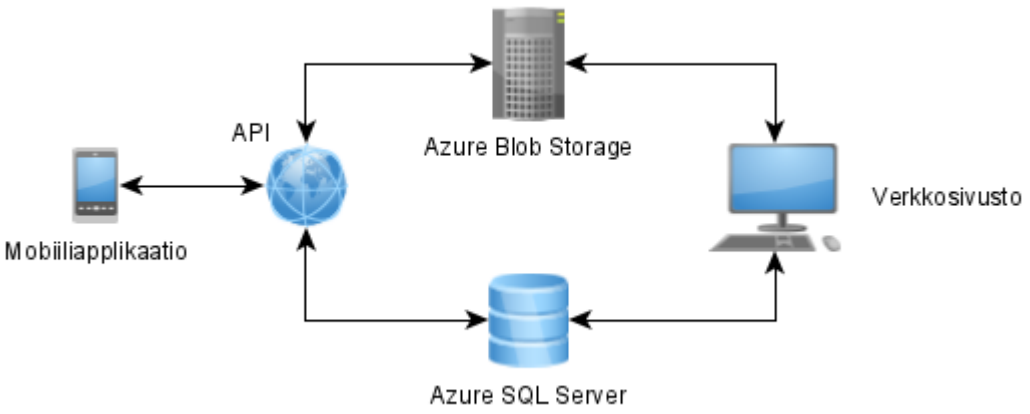
Kuva 8. Varhaisen suunnitteluvaiheen luonnos ohjelmointirajapinnan toiminnasta



Kuva 9. Toinen varhaisen suunnitteluvaiheen luonnos ohjelmointirajapinnan toiminnasta

Ohjelmointirajapinnan pitäisi vastaanottaa mobiiliapplikaatiosta tai muusta lähteestä ilmoitus, johon lisättäisiin samalla merkintä, että ilmoitusta ei ole vielä hyväksytty. Tästä ilmoitus lähetettäisiin ohjelmointirajapinnan avulla tietokantaan. Suunnitelmaan kuului ilmoituksen yhteydessä otettujen kuvien lähettäminen tiedostopalvelimelle. Tämä täytyisi tehdä ennen ilmoituksen lähettämistä, jonka jälkeen vastaukseksi ohjelmointirajapinnalta tulisi saada tiedoston lopullinen polku.

Nykyinen suunnitelma ohjelmointirajapinnan toiminnasta kirjoittamisen hetkellä oli kuvan 10 kaltainen. Nykyisen suunnitelman mukaan mobiiliapplikaatio lähettäisi ensin ilmoituksesta kerätyt kuvatiedostot ohjelmointirajapinnan kautta tiedostopalvelimelle, josta takaisin vastaukseksi tulisi kuvatiedoston osoite.



Kuva 10. Tämänhetkinen suunnitelma ohjelmointirajapinnan toiminnasta projektin muiden osien kanssa

4.4 Tietokannan suunnittelu

Verkkosivuston lisäämisen jälkeen ja ennen ohjelmointirajapinnan suunnittelua tiedettiin, että jonkinlainen tietokanta tarvittaisiin ilmoitusten tallentamiseen. Tietokannan suunnittelu lähti jälleen siitä, että kartoitettaisiin mahdolliset vaihtoehdot ja valittaisiin opiskelijan budjetille ja projektin toteuttamiselle sopivin tietokannan tarjoava palvelu tai tietokantamoottori.

Tietokantamoottoreita oli valittavana monia, yleisimpiin löytyi ilmainen lisäosa ASP.NET Core -sovelluskehykseen, joka helpotti käyttöönottoa. Tietokantamoottoreihin, joihin ilmainen lisäosa oli saatavilla, kuuluivat Oracle DB, MySQL, MariaDB, PostgreSQL, Azure Cosmos DB, SQL API, SQLite ja SQL Server. Näiden kokeilu aloitettiin etsimällä ilmaisia palveluja, jotka tarjoaisivat mahdollisimman halvan, mieluiten ilmaisen, alustan tietokannan testaamista varten. Testaus aloitettiin SQL Serveristä, koska käytössä oli jo muitakin Microsoftin sovelluksia, ohjelmistokehyksiä ja palveluja.

Kartoitettiin palveluja, jotka tarjoaisivat työn toteutuksen ajaksi SQL Server -tietokantaa ilmaiseen käyttöön. Päädyttiin Microsoft Azure-palvelun verkkosivustolle, jossa suoraan mainostettiinkin ilmaista kokeilujaksoa. Kokeilujen ja testauksien myötä selvisi, että Microsoft Azure SQL Database oli tarpeeksi hyvä projektin tarpeisiin. Valintaan vaikutti Microsoft Azure -palvelun tarjoama tilaus opiskelijoille, johon sisältyy vuoden ajaksi 100 dollaria saldoa, kaikki mitä normaali ilmainen tili sisältää ja muita hyödyllisiä palveluja.

Tietokantaan tallennettavan ilmoituksen olisi pitänyt koostua vain neljästä eri taulukosta: kategoria, ilmoitus, pinni ja kuva taulukoista. Kategoriataulukolla on yksi tai monta ilmoitusta, ilmoituksella on yksi tai monta pinniä, pinnillä taas voi olla kuvia. Suunnitelma muuttui tietokannan toteutuksen alettua ja suunnitelmaan piti lisätä yksi taulukko lisää. Microsoftin (2019) dokumentaation mukaan monen-suhde-moneen-relaatioita ei tueta, jollei niiden välillä ole taulukkoa, joka liittää nämä toisiinsa. Tämän taulukon pitäisi yhdistää taulukot toisiinsa ja muodostaa monen-suhde-moneen-relaation. Kahden taulukon yhdistävä taulukko lisätään tietokantaan kuvan 11 tapaan.

Kuva 11. Liitostaulukko tietokannassa ER-kaaviona (Crow's foot notation)

5 TOTEUTUS

Ensimmäisenä toteutettiin mobiiliapplikaatio, jonka jälkeen siirryttiin verkkosivuston toteutukseen. Verkkosivuston toteutuksen aikana toteutettiin tietokanta ja osa ohjelmointirajapintaa. Ohjelmointirajapinnan toteutuksen aikana verkkosivuston ja mobiiliapplikaation toteutukseen tehtiin muutoksia. Tässä luvussa käydään läpi verkkosivuston, mobiiliapplikaation ja ohjelmointirajapinnan toteutus.

5.1 Mobiiliapplikaation toteutus

Mobiiliapplikaatio toteutettiin Microsoftin omistamalla Xamarin.Forms-käyttöliittymäkehyksellä. Applikaation käyttöliittymä koostuu Xaml-tiedostoista, joissa käytetään pääasiassa XML-merkintäkieltä. Jokaiseen Xaml-tiedostoon on myös liitetty C#-ohjelmointikieltä tukeva Xaml.cs-tiedosto.

Sivuja applikaatiossa on kuusi kappaletta, nämä ovat CameraPage, DescriptionPage, MapPage, NameInputPage, OhjePage1 ja Page1. Ensimmäisenä sivuna on Page1, jossa applikaatio saa tiedon siitä, minkälaisen ilmoituksen käyttäjä valitsee. Tiedon saanti on toteutettu asettamalla kokonaisluku integer-muuttujaan. Kuvassa 12 näkyy, miten kokonaisluku asetetaan CameraPage- tai DescriptionPage-sivulle. Sama kokonaisluku asetetaan CameraPage- ja

DescriptionPage-sivuilla UserData-luokkaan (Kuva 13), jota käytetään lopullisen ilmoituksen rakentamisessa.

```
try {
    if (statusStorage == PermissionStatus.Granted && statusCamera == PermissionStatus.Granted) {
        //Navigate to CameraPage, set the mode to 2 and proceed without a name.
        await Navigation.PushAsync(new CameraPage(2, null)).ConfigureAwait(true);
    } else {
        //Navigate to DescriptionPage, set the mode to 2 and proceed without a name.
        //DescriptionPage does not require any permissions.
        await Navigation.PushAsync(new DescriptionPage(2, null,
            Constants.descriptionPageMode2)).ConfigureAwait(true);
    }
```

Kuva 12. Navigaatio CameraPage- ja DescriptionPage-sivuille.

```
public class UserData
{
    4 references | 1 author, 1 change
    public string Name { get; set; }
    4 references | 1 author, 1 change
    public string PersonalEmail { get; set; }
    4 references | 1 author, 1 change
    public string Description { get; set; }
    5 references | 1 author, 1 change
    public int Mode { get; set; }
}
```

Kuva 13. UserData-luokka

Jos käyttäjä valitsi ilmoituksen, joka ei tarvitse ilmoittajan nimeä, niin silloin tarkistetaan, onko applikaatiolle annettu lupa käyttää tiedostojärjestelmää ja puhelimen kameraa. Jollei näiden käyttöön ole annettu lupaa, applikaatio jatkaa seuraavaksi DescriptionPage-sivulle, joka on vain CameraPage ilman mahdollisuutta ottaa kuvia. Jos käyttäjä taas valitsi ilmoituksen tyypiksi ”Asuinalueella olevan jätteen ilmoitus”, niin applikaatio siirtyy NameInputPage-sivulle, jossa ensin tarkastetaan, että onko applikaatiolle annettu tiedostojärjestelmän käyttöön lupaa. Tällä sivulla applikaatio tallentaa käyttäjän syöttämän nimen applikaation LocalApplicationData-kansioon tekstitiedostoksi, jos tiedostojärjestelmän käyttöön on annettu lupa. Ilmoittajan nimi siirretään lopuksi seuraavalle sivulle.

CameraPage-sivulla ilmoitukseen lisätään joko kuvaus tai kuva. Sivulta ei voi edetä ennen kuin kuvauksessa on tekstiä tai yksi kuva on otettu. Kuvaus lisätään samaiseen UserData-luokkaan. Kuvanottoon mentäessä tarkistetaan James Montemagon kehittämän MediaPlugin lisäosan avulla, onko kamera

saatavilla tai kuvanotto edes tuettu mahdollisuus. Jollei ole, annetaan käyttäjälle ponnahdusikkunan avulla ilmoitus asiasta. Jos kamera on saatavilla ja kuvanotto tuettu, käytetään MediaPlugin-lisäosan TakePhotoAsync-funktiota (Kuva 14) kameran avaamiseen ja kuvan ottamiseen. Tämän jälkeen kuva lisätään listaan, joka näkyy applikaation käyttöliittymässä myös käyttäjälle. Listasta voi tämän jälkeen poistaa kuvia tai lisätä samalla tavalla seuraava kuva.

```
try {
    //Call for camera and wait for the user to take a picture.
    var photo = await CrossMedia.Current.TakePhotoAsync(new StoreCameraMediaOptions
    {
        SaveToAlbum = true,
        PhotoSize = PhotoSize.Medium,
        DefaultCamera = CameraDevice.Rear,
        Name = DateTime.Now.ToString(),
        Directory = Constants.cameraFolderName
    }).ConfigureAwait(false);

    //Add imageStream to ViewModel as a new ListImage
    if (photo != null) {
        try {
            ImageSource imageStream = ImageSource.FromStream(() =>
            {
                stream = photo.GetStream();
                return stream;
            });
            var vm = BindingContext as ListImageViewModel;

            vm.AddCommand.Execute(new ListImage() {
                ImageSource = imageStream,
                FileType = "jpg",
                FilePaths = new FilePaths {
                    AlbumFilePath = photo.AlbumPath,
                    TempFilePath = photo.Path
                },
                MediaFile = photo });
        } catch (Exception ex) {
```

Kuva 14. TakePhotoAsync ja saadun streamin lisäys applikaation listanäkymään

CameraPage-sivulla ”Seuraava”-painikkeen painallus prosessoi mahdolliset otetut kuvat kahteen eri listaan. Tässä vaiheessa asetetaan boolean-tyyppisen hasPictures-muuttujan arvoksi ”true”. Jos applikaatiolle on annettu lupa käyttää puhelimen paikannusta ja kuvasta löytyy paikannustiedot, lisätään kuvan paikannustiedot, tiedostonimi ja -polku PositionWithImageInfo-tyyppiseen listaan, muuten lisätään tiedostonimi ja -polku FilePaths-tyyppiseen listaan. Jos positionWithImageInfoList-listalla on enemmän kuin nolla jäsentä, asetetaan hasPositions-boolean muuttujan arvoksi ”true”. Jollei listalla ole PositionWithImageInfo-tyyppisiä jäseniä, tarkistetaan ja pyydetään jälleen paikannuksen

käyttöoikeudet. Jos käyttöoikeudet ovat myönnetty, kutsutaan ProcessUserLocation-metodia, joka palauttaa GeoLocation.GetLocationAsync-funktion avulla puhelimen nykyisen position. Samalla tarkistetaan, onko saatu positio Suomen rajojen sisällä, jollei ole, ei positiota lisätä mihinkään, muussa tapauksessa se lisätään PositionInfo-tyyppiseen muuttujaan. Lopuksi valitaan hasPictures- ja hasPositions-boolean-muuttujien avulla, mihin OhjePage1-sivun konstruktoreista siirrytään seuraavaksi.

OhjePage1-sivulla ei muuta tapahdu, kuin käyttäjän nopea ohjeistaminen lyhyellä pätkällä tekstiä. MapPage-sivulle siirryttyä OhjePage1 ”Seuraava”-painiketta painamalla, asetetaan kartan näkymä joko positionInfoList-listan tai positionWithImageInfoList-listan ensimmäisen jäsenen position kohdalle. Jollei positioita ole, asetetaan näkymä näyttämään ”koko Suomi”. Samalla prosessoidaan aikaisemmin mainitut PositionInfo- ja PositionWithImageInfo-tyyppiset listat Google Maps -karttanäkymän haluamaan muotoon (Kuva 15 ja Kuva 16). Samaisella sivulla voi myös lisätä pinnejä haluamiinsa paikkoihin kartan pitkällä painalluksella, jolloin ajetaan Map_Long_Clicked-metodi. Jos Locations-lista on tyhjä, ajetaan pinCount++-koodi kerran ja lisätään Google Maps Pin -merkki listaan. Pinnin Label-muuttujaksi annetaan ”Constants.googleMapsPinName + pinCount”, Constants.googleMapsPinName-muuttujan arvo on kirjoituksen hetkellä ”Pin” ja pinCount-muuttujan arvo tulee tyhjän listan aikana olemaan ”1”. Jollei lista kuitenkaan ole tyhjä, katsotaan listan jäsenien Label-muuttujista puuttuvat luvut ja lisätään uusin Google Maps Pin -merkki pienimmäksi puuttuvaksi luvuksi.

```
2 references
private void ProcessToLocations(List<PositionInfo> pil) {
    try {
        //Kutsu ListImageViewModel.cs tiedoston AddCommandia.
        //Muunna PositionInfo tyyppisen listan itemi
        //Google Maps haluamaan muotoon.
        for (int i = 0; pil.Count > i; i++) {
            pinCount++;
            ((MapViewModel)BindingContext).AddCommand.Execute(new MapPinInfo()
            {
                PositionInfo = pil[i],
                Label = Constants.googleMapsPinName + " " + pinCount,
                HasBeenManuallyAdded = false
            });
        }
    } catch (Exception ex) {
```

Kuva 15. ProcessToLocations PositionInfo-tyyppisellä listalla

```
1 reference
private void ProcessToLocations(List<PositionWithImageInfo> pwiil) {
    try {
        //Kutsu ListImageViewModel.cs tiedoston AddCommandia.
        //Muunna PositionWithImageInfo tyyppisen listan itemi
        //Google Maps haluamaan muotoon.
        for (int i = 0; pwiil.Count > i; i++) {
            pinCount++;
            ((MapViewModel)BindingContext).AddCommand.Execute(new MapPinInfo()
            {
                PositionInfo = new PositionInfo {
                    Position = pwiil[i].Position,
                    IsMockLocation = pwiil[i].IsMockLocation
                },
                Label = Constants.googleMapsPinName + " " + pinCount,
                HasBeenManuallyAdded = false
            });
        }
    } catch (Exception ex) {
```

Kuva 16. ProcessToLocations PositionWithImageInfo-tyyppisellä listalla

Samaiseen MapPage-sivuun kuuluu vielä keskeneräiset SendData-, BuildData- ja BuildReport-metodit, jotka korvasivat verkkosivuston ohjelmointirajapinnan mukaan ottamisen myötä vanhemman SendEmail-metodin, joka kokosi kerätyistä positioista, kuvista ja kuvauksesta käyttäjälle lähetystä vailla olevan sähköpostin.

5.1.1 Käyttöoikeudet

Käyttöoikeuksia joutuu Android-alustalle kehitetyssä applikaatiossa kysyä joka kerta, kun applikaatio tekee jotain haluttuun käyttöoikeuteen liittyvää. Käyttöoikeuksien pyyntöön käytettiin jälleen yhtä James Montemagon kehittämää lisäosaa, nimeltä Plugin.Permissions. Käyttöoikeuksia ei kuitenkaan tarvitse saman käynnistyskerran aikana kysyä uudestaan, jollei käyttäjä ole päättänyt muuttaa niitä applikaation ajon aikana.

Käyttäjän antaman nimen muistiin tallentamiseen ja lukemiseen tarvittiin käyttöoikeudet tiedostojärjestelmän käyttöön. Kuvassa 17 näkyy, miten koodin puolella lupaa tiedostojärjestelmän käyttöön kysytään. Kameran ja puhelimen position käyttölupien kysyntään käytetään lähes samankaltaisia metodeja.

```
public async Task<PermissionStatus> RequestStoragePermissionsAsync(string rationale) {
    try {
        var permission = await CrossPermissions.Current.CheckPermissionStatusAsync<StoragePermission>().ConfigureAwait(false);
        if (permission != PermissionStatus.Granted) {
            if (await CrossPermissions.Current.ShouldShowRequestPermissionRationaleAsync(Permission.Storage).ConfigureAwait(true)) {
                await DisplayAlert(Constants.storagePermissionRequestTitle, rationale, Constants.alertConfirm).ConfigureAwait(false);
            }
            permission = await CrossPermissions.Current.RequestPermissionAsync<StoragePermission>().ConfigureAwait(false);
        }
        return permission;
    } catch (Exception ex) {
```

Kuva 17. Tiedostojärjestelmän käyttöluvan kysyvä async Task, Plugin.Permissions-lisäosan avulla toteutettu

5.1.2 Käyttöliittymä

Applikaation käyttöliittymän elementit tehdään XAML-tiedostoon. Elementtien sijoittelu toimii melkein samalla tavalla kuin verkkosivustojen kehityksen puolella, vaikka eroavaisuuksia onkin ja XAML-merkintäkieli perustuu XML-merkintäkieleen. Käyttöliittymän pystyi myös tehdä Visual Studio -kehitysympäristön tarjoamalla visuaalisella työkalulla. Se ei kuitenkaan toiminut kehityksen aikana kertaakaan, joten kaikki käyttöliittymään liittyvä lähdekoodi kirjoitettiin itse.

Applikaation käyttöliittymässä näkyvät tiedot ja lähdekoodissa olevat muuttujat eivät päivity samanaikaisesti, jollei XAML-tiedostossa näitä kahta ole liitetty toisiinsa. Ensin asetetaan näkymälle näkymämallin konteksti (Kuva 18), jonka jälkeen käyttöliittymässä näkyvään osaan voi liittää näkymämallista löytyvän muuttujan (Kuva 19, Kuva 20), jota voi sen jälkeen muuttaa mallissa eli applikaation lähdekoodissa.

```
<ContentPage.BindingContext>
    <!--Asetetaan BindingContext ListImageViewModel luokkaan-->
    <local:ListImageViewModel />
</ContentPage.BindingContext>
```

Kuva 18. CameraPage-sivun BindingContext, joka osoittaa ListImageViewModel-näkymämalliin. Rivit 11-14, liite 2

```
<ListView ItemsSource="{Binding ListImages}"
          HasUnevenRows="True"
          SelectionMode="None"
          Grid.Row="0">
```

Kuva 19. CameraPage-sivun ListView-elementin jäsenet liitetty ListImageViewModel-näkymämallin ListImages-listaan. Rivit 26–29, liite 2

```
public ObservableCollection<ListImage> ListImages { get; }
```

Kuva 20. ListImage tyyppinen ObservableCollection ListImageViewModel tiedostossa. Rivi 7, liite 3

Xamarin.Forms-käyttöliittymäkehys ei aina tehnyt haluttujen käyttöliittymämuutosten tekoa helpoksi. CameraPage-sivulla (Kuva 21) käytetty EditorRenderer-käyttöliittymäelementti ei antanut normaalisti vaihtoehtoa läpinäkyvälle taustalle. Elementille tehtiin CustomRenderer_Droid-niminen luokka, johon asetettiin Control.SetBackground-komennolla läpinäkyvä tausta ja Control.SetRawInputType()-komennon Android.Text.InputTypes.TextFlagNoSuggestions-taustakomennolla poistettiin kirjoitetun tekstin alleviivaus.

Kuva 21. Mobiiliapplikaation CameraPage-sivu

5.1.3 Virheiden raportointi

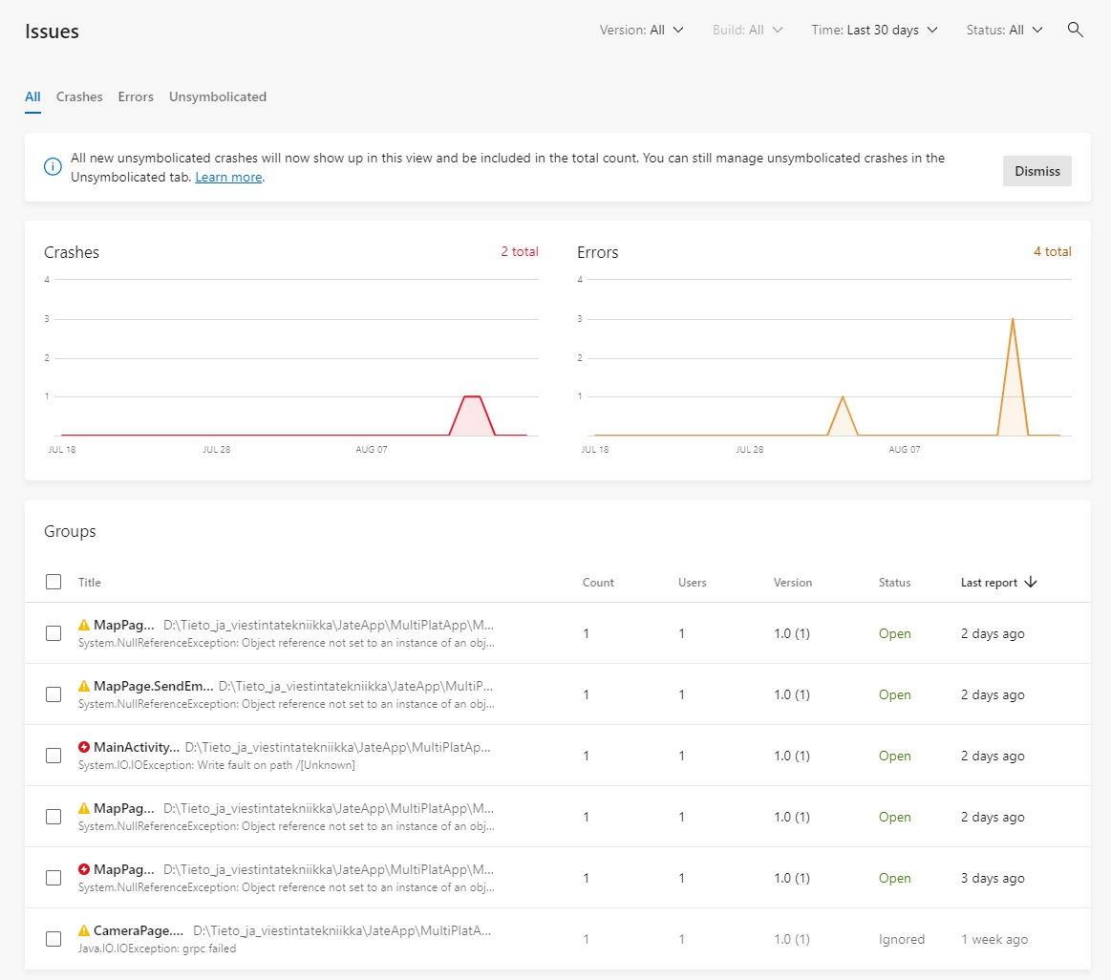
Applikaation virheiden ilmoittamiseen tarvittiin jokin helpohko ratkaisu, tällaisen tarjosi Visual Studio App Center -palvelu, jonka sisällyttäminen ei vaatinut

muuta kuin lisäosan asentamisen, sanakirja muuttujan tekemisen try/catch-blokin catch-osioon ja sen lähettämisen Visual Studio App Center -palvelun palvelimelle Crashes.TrackError()-funktion avulla. Palvelimelle lähetettiin itse kaapattu virhe ja lähdekoodissa asetetut lisätiedot (Kuva 22). Vaikka applikaatio ei ennen kaatumista ehtisikään lähettää virheilmoitusta, niin virheilmoitus säilyy tallennettuna muistissa niin kauan, kunnes applikaatio käynnistyy uudestaan ja yrittää samaisen virheraportin lähetystä.

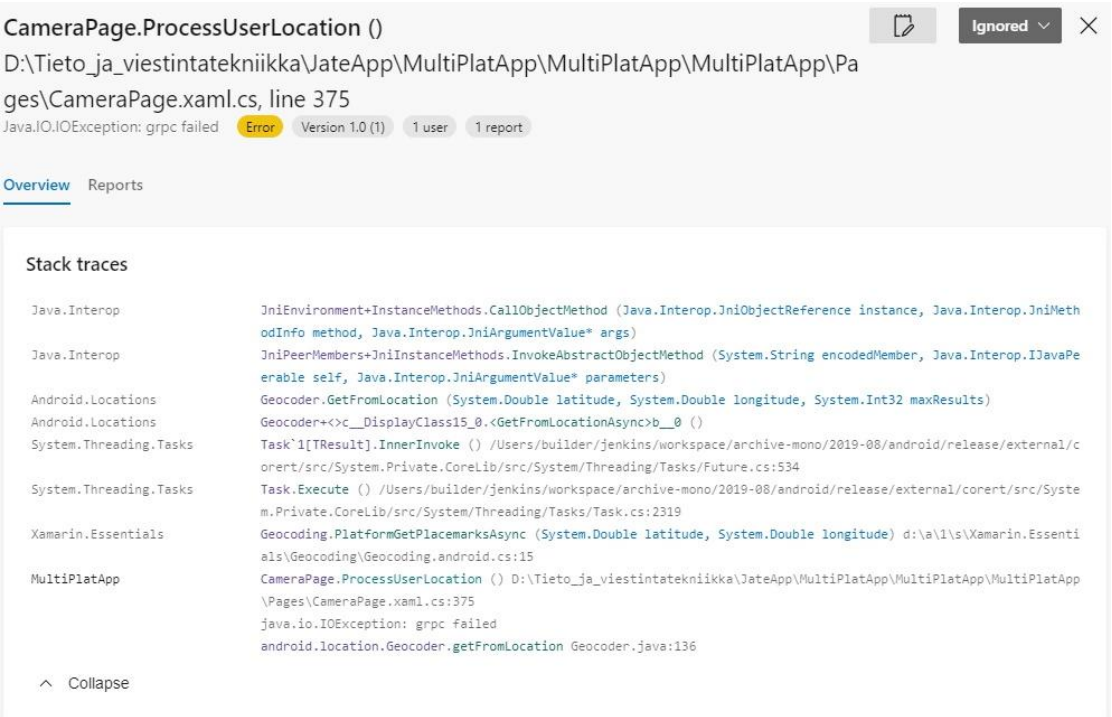
```
} catch (Exception ex) {
    DisplayAlert(Constants.errorDataTitle, Constants.errorData, Constants.alertConfirm);
    var properties = new Dictionary<string, string>
    {
        { "Category", Constants.errorCategoryData },
        { "Exception type", ex.GetType().ToString() }
    };
    Crashes.TrackError(ex, properties);
}
```

Kuva 22. Try/Catch-blokin catch-osio, DisplayAlert-ponnahdusikkuna, properties-sanakirja ja tiedon lähetys DescriptionPage-sivun konstruktorissa

Kaikki mahdolliset applikaation aiheuttamat virhetilanteet yritettiin kaapata. Ainoastaan ne virhetilanteet, joissa mahdollinen syypää on itse puhelimen rauta, jätettiin kaappaamatta. Kaapatut virheraportit näkyvät Visual Studio App Center -palvelussa (Kuva 23), jossa tallennetut ex- ja properties-muuttujien tiedot näytetään helposti ymmärrettävässä näkymässä verkkosivustolla (Kuva 24).



Kuva 23. Visual Studio App Center -palvelussa näkyvät virheraportit



Kuva 24. Virheraportti, jonka aiheutti epäonnistunut applikaation rakentaminen bugisella Visual Studio -kehitysympäristön versiolla

5.2 Verkkosivuston toteutus

Verkkosivuston toteutus aloitettiin seuraamalla Microsoftin dokumentaatiota. Sen avulla saatiin aikaiseksi ensimmäinen, jokseenkin toimiva verkkosivu. Ensin valmistui verkkosivuston indeksisivu, johon tahdottiin lisätä ilmoitusten suodattaminen päivämäärän, lähettäjän, kategorian ja ilmoituksen yksilöllisen tunnisteen mukaan. Samaisten kenttien mukaan haluttiin myös mahdollisuus järjestää ilmoitukset.

Indeksisivun rakenne koostuu sivun datakontekstin asettavasta luokasta, joka on tehty kuvastamaan malliluokkien yhteyksiä toisiinsa. Datakontekstiin kuuluvat malliluokat taas kuvastavat tietokannan jokaista taulukkoa. Näiden jälkeen IQueryable-tyyppiseen listaan voidaan LINQ-kyselyllä (Kuva 25) hakea kaikki ilmoitukseen liittyvä tieto.

```
IQueryable<Ilmoitus> ilmoituksetIQ = from s in _context.Ilmoitukset
select s;
```

Kuva 25. IlmoituksetIQ-listan LINQ-kysely

LINQ-kyselyn ajettua, voidaan jatkaa täyden listan järjestämiseen tai suodattaa listaa ennen seuraavaan vaiheeseen siirtymistä lambdalausekkeen (Kuva 26) avulla. Lista korvataan tämän avulla uudella listalla, johon on aikaisemmasta täydestä listasta lisätty vain ne ilmoitukset, joiden yksilöllinen tunniste vastaa tarkalleen käyttäjän hakemaa numeroa.

```
ilmoituksetIQ = ilmoituksetIQ.Where(s => s.Id.Equals(Id));
```

Kuva 26. IlmoituksetIQ-listan suodattaminen

IlmoituksetIQ-lista järjestetään tämän jälkeen uudella lambdalausekkeella, joka järjestää listan oletuksena laskevaan järjestykseen päivämäärän mukaan. Listan järjestyksen jälkeen haetaan tietokannasta halutut tiedot ilmoitus, kategoria, pinni ja pinnikuva taulukoista ja näytetään ne indeksisivulla. Indeksisivulla kerralla näkyvät tulokset rajoitetaan tällä hetkellä kymmeneen. Koska sivun vaihdoksen johdosta sivu ladataan uudestaan ja listassa mennään kymmenen jäsentä eteenpäin, pitää sivuston säilyttää suodatus ja listan järjestys. Listassa seuraavalle sivulle siirtyminen suoritetaan yleisesti käytössä olevalla

PaginatedList-luokalla. Listan järjestys ja suodatus ei kuitenkaan sivunvaihdoksen yli ilman apua säilynyt, joten sen säilyttämiseksi siirrettiin asp-route-parametrin avulla sortOrder, pageIndex ja muut tarvittavat muuttujat seuraavalle sivulle (Kuva 27).

```
<a asp-page="./Index"
   asp-route-sortOrder="@Model.CurrentSort"
   asp-route-pageIndex="@(Model.Ilmoitukset.PageIndex - 1)"
   asp-route-currentEmailFilter="@Model.CurrentEmailFilter"
   class="btn btn-primary @prevDisabled">
    Previous
</a>
```

Kuva 27. HTML-merkintäkielen <a>-tunnisteen ja asp-route-parametrin avulla lähetettävät muuttujat

Ilmoituksien jokaista pinnitaulukkoa kohden näytetään karttasivulla yksi Google Maps -merkki. Merkin InfoWindow-osiossa näkyy ylläpitäjälle ilmoitus, pinnitaulukoiden yksilölliset tunnisteet, kategoria, lähettäjän sähköposti, päivämäärä, koordinaatit, tarkkuus, kuvat ja kuvaus. Jos karttanäkymää kuitenkin tarkistelee normaali käyttäjä, ei yksilöllisiä tunnisteita ja lähettäjän sähköpostia siirretä ollenkaan Google Maps -näkymän taustapuolelta verkkosivun puolelle.

Käyttöliittymä

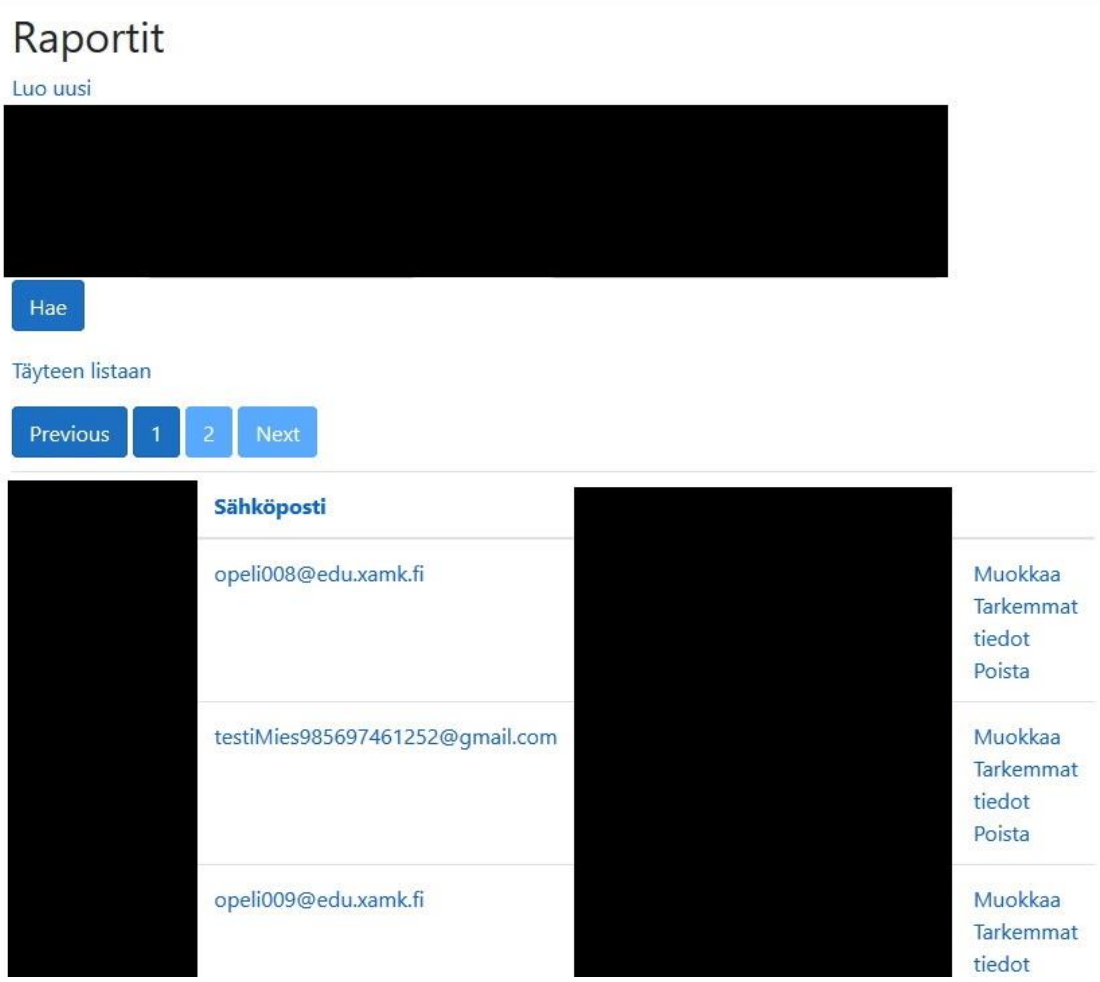
Käyttöliittymä perustui normaaliin Razor-sivun pohjaan. Razor-sivua ei ole yhtä helppo muokata kuin itse tehtyä verkkosivua. Mahdollisuuksia käyttöliittymän muokkaukseen oli joitakin, yksi niistä uuden site.css-tiedoston teko, tällä kertaa kuitenkin käytettiin Razor-sivun sisältämää styles-osiota. Styles-osion arvot asetetaan kaikkien muiden tyylitietojen jälkeen ja toimii parhaiten silloin, kun halutaan muokata käyttöliittymän pienempiä osia. Esimerkkinä kuva 28, jossa Google Maps -näkymän sisältävän sivun Styles-osiossa hiiren kursori asetetaan näyttämään normaali kursori osoittavan sormen sijasta.

```
.popup img {
    cursor: pointer
}
```

Kuva 28. Google Maps -näkymän infoikkunan kuvan kursorin tyylin asetus

Indeksisivulle haluttiin myös mahdollisuus siirtyä sivulta toiselle muillakin keinoin kuin vain edellisessä luvussa läpi mainitun previous-painikkeen ja samalla tavalla toimivan next-painikkeen avulla. Tähän tarvittiin PaginatedList-luokan TotalPages-muuttujaa ja tieto nykyisestä sivusta, nykyisen sivun painike asetetaan disabled-tilaan ja muitten sivujen painikkeet jäävät enabled-tilaan. Tämä toteutus tehtiin kokonaan Indeksisivun cshtml-tiedostossa Razor-sivun koodilohkon sisällä, joihin voi kirjoittaa suoraan C#-ohjelmointikieltä.

Verkkosivuston indeksisivulla (Kuva 29) ilmoituksista näytetään taulukon sisältämät tiedot. Listan sähköposti-linkkiä painamalla asp-route-parametri (Kuva 30) kutsuu taustapuolen lähdekoodia (Kuva 31) asettamaan email_asc-arvon CategorySort-muuttujaan ?:-operaattorilla. CategorySort-muuttujaan email_asc arvon asettaminen järjestää listan aakkosjärjestykseen sähköpostien mukaan. Linkin seuraava painallus vaihtaa arvon email_desc-arvoon ja kääntää järjestyksen.



Kuva 29. Ilmoituksien indeksisivu

```
<a asp-page="./Index" asp-route-sortOrder="@Model.EmailSort">
```

Kuva 30. HTML <a>-tagi ja asp-route-parametri

```
EmailSort = sortOrder == "email_asc" ? "email_desc" : "email_asc";
```

Kuva 31. EmailSort-muuttujan arvon asetus

5.3 Ohjelmointirajapinnan toteutus

Ohjelmointirajapinta toteutettiin Net Core 3.1 -sovelluskehyksen ja Visual Studio -sovelluksen tarjoamalla Scaffolding-ominaisuudella. Scaffolding-ominaisuus muodostaa ohjelmointirajapinnan CRUD-toiminnot kehittäjän toteuttaman Model-tiedoston ja tietokantakontekstin avulla. CRUD-toiminto on jaettu neljään osaan: Create, Read, Update ja Delete. CRUD-toimintojen Create- ja Update-toiminnot käyttävät hypertekstin siirtoprotokollan (HTTP) PUT-pyyntöä, kun taas Read-toiminto käyttää GET-pyyntöä ja Delete-toiminto DELETE-pyyntöä.

Scaffolding-ominaisuudella toteutettuja CRUD-toimintoja ei tarvinnut kovin paljoa automaattisen generoinnin jälkeen muuttaa. Tällaisenaan Delete-toiminto ei kuitenkaan osaa poistaa kahden taulukon yhdistävän taulukon yli ulommaista taulukkoa, vaikka se jäisikin orvoksi. Sivuston Delete-sivulle lisättiin tämän takia toiminto, joka poistaa orvoksi jäävät taulukot. Tämä ominaisuus toteutettiin tekemällä lista poistettavan tietokannan taulukon liitostaulukkojen taulukoista, jonka jälkeen poistetaan listan taulukot tietokannasta. Listan taulukoiden poiston jälkeen poistetaan itse taulukko, jolle poistopyyntö tehtiin.

6 ONGELMAT

Ongelmia kehityksen aikana tuotti tiettyjen sovelluksien epävakaus, ohjelmointikielien syntaksien osaamisen puute ja dokumentaation hajanaisuus. Xamarin.Forms-käyttöliittymäkehyksen päivitystahdin takia dokumentaatio oli vielä puutteellinen. Kaikkia ominaisuuksia ei dokumentaation puolelta löytynyt, vaan piti turvautua käyttöliittymäkehyksen kehittäjien yhteisöön. Microsoftin dokumentaation avulla sai toteutettua yksinkertaisen mobiiliapplikaation, suurin osa käyttöliittymäkehyksen hienommista ominaisuuksista jätettiin tutoriaalien ulkopuolelle. Microsoftin dokumentaatiosta ja tutoriaaleista ei myöskään löytynyt

mainintaa lisäosista, jotka nopeuttivat huomattavasti Xamarin.Forms-käyttöliittymäkehyksellä mobiiliapplikaation toteuttamista.

Asynkroninen ohjelmointi tuli projektin aikana uutena asiana. Sen opettelu vei aikaa ja jotkin mobiiliapplikaatiota piinanneet virheet johtuivat asynkronisen koodin osaamisen puutteellisuudesta. Asynkronisessa koodauksessa käytetyn ConfigureAwait-avainsanan oikeaoppisen käytön opettelu oli myös haaste. Mobiiliapplikaation toteutuksen alkupuolella ei ollut aina selvää, onko ConfigureAwait() asetettu väärin vai oikein. Opettelussa auttoi paljon Stephen Clearyn, Dan Lorenzin ja Stephen Toubin kirjoittamat blogikirjoitukset. Näiden opettelussa olisi mennyt vielä kauemmin ilman hyviä esimerkkejä ja omia testejä.

Jotkin käyttöliittymäkehyksen elementit eivät aina toimineet halutulla tavalla toistensa kanssa. Mobiiliapplikaation CameraPage-sivulla käytetyn ObservableCollection-listan muiden jäsenien kuvat hävisivät näkyvistä, kun yksi listan jäsen poistettiin. Listassa olevat kuvat pitää joko tallentaa välimuistiin tai puhelimen muistiin. Xamarin.FFImageLoading-lisäosan avulla päätettiin tallentaa ObservableCollection-listan kuvat välimuistiin applikaation ajon ajaksi.

Mobiiliapplikaation toteutuksen aikana Microsoft Visual Studio -sovellus päivitettiin silloin, kun tarvittiin uudemman version ominaisuuksia tai niitä haluttiin kokeilla. Sovelluksen päivittäminen 16.6.0-versioon aiheutti Java.Runtime-virheitä, eikä applikaatioita saatu sovellusversiolla ajettua. Ongelma ratkesi asentamalla sovelluksen vanhempi versio. Xamarin.Forms-käyttöliittymäkehyksen käytön aikana tulleet virheet korjautuivat yleensä poistamalla bin- ja obj-nimiset kansiot projektikansion juuresta.

7 YHTEENVETO

Alustavasti suunnitellun mobiiliapplikaation suunnittelu ja toteutus edistyi hyvää tahtia ja se saatiin ajoissa alustavan suunnitelman mukaiseksi. Mobiiliapplikaation suunnittelu sujui nopeasti, mutta toteutuksen aikana ongelmia oli Android-emulaattorin, asynkronisen koodin ja tiettyjen sovellusten ja niiden ominaisuuksien kanssa. Mobiiliapplikaation käyttöliittymään ei lisätty graafisia

ikoneja, vaan käytettiin tyyliteltyjä elementtejä. Graafisien ikonien poisjättäminen säästi toteutuksessa aikaa ja auttoi keskittymään sovelluksen toimivuuteen. Verkkosivuston alustava suunnittelu oli enemmän oikean sovelluskehikon etsimistä ja sen ominaisuuksien testaamista kuin käyttöliittymän ja ominaisuuksien suunnittelua. Sopivan sovelluskehyksen löydyttyä ja sen opeteltua alustava suunnittelu sujui kuitenkin hyvin ja yksinkertainen nettisivusto saatiin toteutettua nopeasti. Tietokannan, ohjelmointirajapinnan ja mobiiliapplikaation kanssa toimivan verkkosivuston toteutuksessa meni kuitenkin kauemmin.

Opinnäytetyön ulkopuoliset ongelmat hidastivat kokonaisuuden toteutusta. Jatkokehitykseen joutuvat mobiiliapplikaation ulkoasu ja datan lähetys, ohjelmointirajapinta ja verkkosivuston ulkoasu ja indeksisivun ja ilmoituksen yksityiskohdat näyttävän sivun käyttöliittymä. Mobiiliapplikaation ja verkkosivuston ulkoasun tulisi olla yhtenäisiä. Tällä hetkellä verkkosivusto käyttää Razor-sivun normaalia ulkoasua, joka ei seuraa mobiiliapplikaation teemaa. Verkkosivustoon tulee myös lisätä ylläpitäjien lisääminen ja kirjautuminen. Ilmoituksen tekeminen verkkosivulla pitäisi onnistua yhdeltä sivulta, tällä hetkellä ilmoitus pitää tehdä monessa eri osassa eri sivuilla. Tietokantaan tulisi lisätä ainakin yksi tietue, johon lisätä ilmoituksen tila. Karttanäkymäsivua tulee muokata mobiili- ja tablettilaitteilla sopivaksi. Mobiiliapplikaation datan kerääminen ja lähetys ovat vielä keskeneräisiä ohjelmointirajapinnan hankaluuden ja myöhäisen lisäämisen johdosta. Mobiiliapplikaation tulee myös lisätä vieraslajien ilmoitukseen näkymä, jossa käyttäjä voi valita vieraslajin nimen tai kuvan perusteella. Jollei ilmoituksen lähetyksen aikana löydy yhteyttä, ilmoitus tulisi lisätä jonoon, ja sen lähetystä yrittää uudestaan mobiiliapplikaation seuraavan käynnistyskerran jälkeen.

Työ antoi kokemusta mobiiliapplikaation, verkkosivuston, tietokannan ja ohjelmointirajapinnan toteutuksen eri osista. ASP.NET Core ja Xamarin.Forms tulivat työn aikana tutuiksi. ASP.NET Core ja Xamarin.Forms eivät olleet aiemmin tuttuja, joten suunnittelussa ja toteutuksessa dokumentaatio ja muiden kehittäjien tekemät blogikirjoitukset auttoivat projektin osien oikeaoppisessa suunnittelussa ja toteutuksessa. Tekniikkoja jo hyvin osaavien kehittäjien neuvojen seuraaminen auttoi vähentämään virhetilanteita ja ongelmia projektin myöhemmässä vaiheessa, joka johti vakaamman ja modulaarisemman kokonaisuuden toteutukseen.

Verrattuna samankaltaisiin mobiiliapplikaatioihin, työssä tuotetun mobiiliapplikaation avulla ilmoituksen tekeminen on suoraviivaisempaa ja ilmoituksen paikannustietoja voi tarkastella ja vaihtaa ennen ilmoituksen lähetystä. Verkkosivuston lisäys tuo kokonaisuuteen helposti hallinnoitavan tietokannan ja mahdollisuuden tarkastaa toisten lähettämiä ilmoituksia.

Kokonaisuutta viedään eteenpäin ja sitä tullaan markkinoimaan alustavasti Kotkan ympäristövastaavalle. Jatkossa mahdollinen markkinointi tulisi tapahtumaan sosiaalisessa mediassa.

LÄHTEET

Agile Alliance. 2001. The Agile Manifesto. WWW-dokumentti. Saatavissa: https://www.agilealliance.org/agile101/the-agile-manifesto/ [viitattu 20.11.2020].

Bernard, A. 2020. Stop Wild Dumps, to stop seeing them. WWW-dokumentti. Päivitetty 19.06.2020. Saatavissa: https://play.google.com/store/apps/details?id=fr.stopdechargessauvages.android&hl=en_US&gl=US [viitattu 19.11.2020].

Cherry Red. 2017. Clean the Creek. WWW-dokumentti. Päivitetty 09.01.2017. Saatavissa: https://play.google.com/store/apps/details?id=cleanthecrreek.redcherry.com.ctc [viitattu 18.11.2020].

East Texas Council of Governments. 2020. Report Illegal Dumping. WWW-dokumentti. Päivitetty 16.04.2020. Saatavissa: https://play.google.com/store/apps/details?id=com.app.reportillegaldumping [viitattu 18.11.2020].

Gamelab. 2019. Tervetuloa XAMK Gamelabiin! WWW-dokumentti. Päivitetty 2019. Saatavissa: https://www.gamelab.fi/ [viitattu 24.11.2020].

Google Play s.a. bbits kehittäjä. WWW-dokumentti. Saatavissa: https://play.google.com/store/apps/developer?id=bbits [viitattu 19.11.2020].

Hong, P. 2018. Practical Web Design: Learn the fundamentals of web design with HTML5, CSS3, Bootstrap, jQuery, and Vue.js. E-kirja. Birmingham: Packt Publishing, Limited. Saatavissa: https://kaakkuri.finna.fi/Record/nelli29_mamk.4100000004384996 [viitattu 20.11.2020].

Kooders.fi. 2018. #MitäVittua – Ketterä ohjelmistokehitys. Blogi 28.08.2018. Saatavissa: https://www.kooders.fi/blogi/mitavattua-kettera-ohjelmistokehitys [viitattu 28.08.2020].

Lano, K. 2009. UML 2 Semantics and Applications. E-Kirja. Hoboken: John Wiley & Sons, Inc. Saatavissa: https://kaakkuri.finna.fi/Record/nelli29_mamk.1000000000807353 [viitattu 1.12.2020].

Learnrazorpages.com. 2019. Welcome To Learn Razor Pages. WWW-dokumentti. Päivitetty 08.05.2019. Saatavissa: https://www.learnrazorpages.com/ [viitattu 31.08.2020].

MDN Contributors. 2020. JavaScript. MDN Web Docs. WWW-dokumentti. Muokattu päivämäärä 16.08.2020. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript [viitattu 28.08.2020].

Measy, P., Berridge, C., Gray, A., Oliver, L., Wilmhurst, D. & Wolf, L. 2015. Agile Foundations - Principles, practices and frameworks. E-Kirja. Swindon: BCS Learning & Development Limited. Saatavissa: https://kaakkuri.finna.fi/Record/nelli29_mamk.3710000000359262 [viitattu 1.12.2020].

Microsoft. 2015. Introduction to the C# language and the .NET Framework. WWW-dokumentti. Päivitetty 07.07.2020. Saatavissa: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework [viitattu 28.08.2020].

Microsoft. 2019. Relationships. WWW-dokumentti. Päivitetty 09.09.2020. Saatavissa: https://docs.microsoft.com/en-us/ef/core/modeling/relationships [viitattu 14.10.2020].

Microsoft. 2020a. What is Xamarin.Forms? WWW-dokumentti. Päivitetty 28.05.2020. Saatavissa: https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin-forms [viitattu 28.08.2020].

Microsoft. 2020b. Razor syntax reference for ASP.NET Core. WWW-dokumentti. Päivitetty 12.02.2020. Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-3.1 [viitattu 03.09.2020].

Microsoft. 2020c. Choose between the vCore and DTU purchasing models - Azure SQL Database and SQL Managed Instance. WWW-dokumentti. Päivi-

tetty 28.05.2020. Saatavissa: https://docs.microsoft.com/en-us/azure/azure-sql/database/purchasing-models [viitattu 02.09.2020].

Microsoft. 2020d. Introduction to the core Azure Storage services. WWW-dokumentti. Päivitetty 08.04.2020. Saatavissa: https://docs.microsoft.com/en-us/azure/storage/common/storage-introduction [viitattu 04.09.2020].

Microsoft. 2020e. Introduction to Azure Blob storage. WWW-dokumentti. Päivitetty 24.06.2020. Saatavissa: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction [viitattu 11.12.2020].

Postman. 2020. The Collaboration Platform for API Development. WWW-dokumentti. Päivitetty 09.09.2020. Saatavissa: https://www.postman.com/ [viitattu 09.09.2020].

Sarkkinen, M. 2019. Viransijainen ympäristötarkastaja. Sähköpostikeskustelu 23.09.2019. Kotkan ympäristökeskus.

StatCounter. 2020. Browser Market Share Finland. WWW-dokumentti. Päivitetty 29.11.2020. Saatavissa: https://gs.statcounter.com/browser-market-share/all/finland/#monthly-202010-202010-bar [viitattu 29.11.2020].

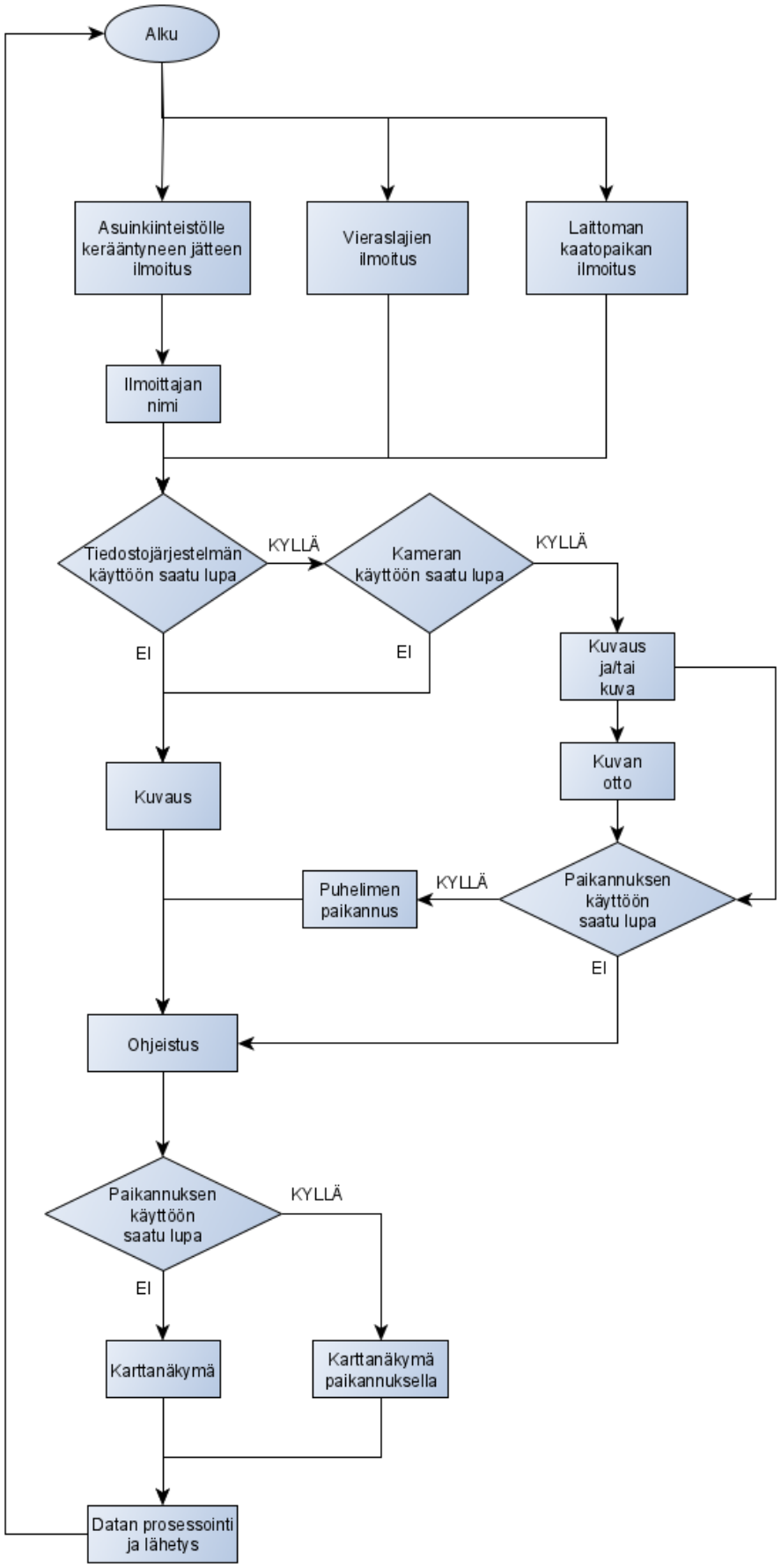
Sumo Logic 2019. What is Microsoft Azure, and Why Use It? PDF-dokumentti. Päivitetty 15.04.2019. Saatavissa: https://assets-www.sumologic.com/resources/brief/microsoft-azure.pdf?mtime=20190415105234&focal=none [viitattu 01.09.2020].

Thornsby, J. 2016. Android UI Design. E-kirja. Birmingham: Packt Publishing, Limited. Saatavissa: https://kaakkuri.finna.fi/Record/nelli29_mamk.3710000000726206 [viitattu 19.11.2020].

TrashOut. 2020. TrashOut - World Cleanup Day partner. WWW-dokumentti. Päivitetty 15.09.2020. Saatavissa: https://play.google.com/store/apps/details?id=me.trashout [viitattu 18.11.2020].

Kuvaluettelo

Liite 1



Kuva 1. Ilmoituksen eteneminen mobiiliapplikaatiossa.

Liite 2

```
10
11  <ContentPage.BindingContext>
12      <!--Asetetaan BindingContext ListImageViewModel luokkaan-->
13      <local:ListImageViewModel />
14  </ContentPage.BindingContext>
15
16  <!--Bindataan ItemsSource ListImages ObervableCollectioniin-->
17  <AbsoluteLayout>
18      <Grid Padding="0"
19            Margin="0"
20            AbsoluteLayout.LayoutFlags="All"
21            AbsoluteLayout.LayoutBounds="1,0,1,1">
22          <Grid.RowDefinitions>
23              <RowDefinition Height="*" />
24              <RowDefinition Height="50" />
25          </Grid.RowDefinitions>
26          <ListView ItemsSource="{Binding ListImages}"
27                    HasUnevenRows="True"
28                    SelectionMode="None"
29                    Grid.Row="0">
30              <ListView.ItemTemplate>
31                  <DataTemplate>
32                      <ViewCell>
33                          <FlexLayout Direction="Row"
34                                      AlignItems="Center"
35                                      HeightRequest="300">
36                              <forms:CachedImage Source="{Binding ImageSource}"
37                                                 HeightRequest="300"
38                                                 FlexLayout.Grow="1"
39                                                 FlexLayout.Order="1" />
```

Kuva 2. Mobiiliapplikaation CameraPage-sivun käyttöliittymän lähdekoodia.

Liite 3

```
1   using System.Collections.ObjectModel;
2   using System.ComponentModel;
3   using Xamarin.Forms;
4
5   namespace MultiPlatApp {
        17 references
6       public class ListImageViewModel : INotifyPropertyChanged {
            6 references
7           public ObservableCollection<ListImage> ListImages { get; }
8
9           //Uusi Command luokka
            1 reference
10          public Command<ListImage> RemoveCommand {
11              get {
12                  return new Command<ListImage>((ListImage) => {
13                      //Poistetaan listImage ListImages ObservableCollectionista
14                      ListImages.Remove(ListImage);
15                  });
16              }
17          }
18
            1 reference
19          public Command<ListImage> AddCommand {
20              get {
21                  return new Command<ListImage>((ListImage) => {
22                      ListImages.Add(ListImage);
23                  });
24              }
25          }
26
            0 references
27          public ListImageViewModel() {
28              //Alustetaan tyhjä ListImage tyyppinen ObservableCollection
29              ListImages = new ObservableCollection<ListImage> { };
30          }
31
32          public event PropertyChangedEventHandler PropertyChanged;
33
34          private string isBusy;
35
            10 references
36          public string IsBusy {
37              get => isBusy;
38              set {
39                  if (isBusy == value) {
40                      return;
41                  }
42                  isBusy = value;
43                  OnPropertyChanged(nameof(IsBusy));
44              }
45          }
46
            1 reference
47          private void OnPropertyChanged(string propertyName) {
48              PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
49          }
50      }
51  }
```

Kuva 3. ListViewImageModel-luokan lähdekoodi.