OULUN AMMATTIKORKEAKOULU

Tuomas Kinnunen

RIISTAKOLMIOLASKENTA-SOVELLUS

RIISTAKOLMIOLASKENTA-SOVELLUS

Tuomas Kinnunen
Opinnäytetyö
Syksy 2014
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, Ohjelmistosuunnittelu

Tekijä: Tuomas Kinnunen
Opinnäytetyön nimi: Riistakolmiolaskenta-sovellus
Työn ohjaaja: Veijo Väisänen
Työn valmistumislukukausi ja -vuosi: Syksy 2014
Sivumäärä: 36 + 3 liitettä

Opinnäytetyön aiheena oli suunnitella riistakolmiolaskenta-sovellus mobiililaitteelle. Sovelluksella kerätään riistakolmiolaskennan aikana tehdyt riistaeläinhavainnot ja lähetetään ne tietokantaan.

Suunnitelmana sovelluksen toteutukselle oli käyttää apuna yrityksen toteuttamaa Scatman-tiedonkeruujärjestelmää, jonka rinnalla riistakolmiolaskenta tulee toimimaan. Sovellukselle tuli toteuttaa tietokanta, palvelin sekä mobiilikäyttöliittymä.

Opinnäytetyö alkoi perehtymisellä riistakolmiolaskentaan, jonka jälkeen tehtiin määrittely sovelluskokonaisuudelle. Sovellukselle suunniteltiin tietokanta sekä palvelimen rajapinta, jonka avulla käyttöliittymän ja tietokannan välille saatiin tietoyhteys. Lopuksi toteutettiin sovelluksen tietokanta, rajapintafunktioita sekä osa käyttöliittymästä.

Asiasanat: riistakolmiolaskenta, HTML, JavaScript, C#, CSS, tietokannat, mobiilisovellukset

ALKULAUSE

Haluan kiittää Scatman Oy:ta työntekijöineen opinnäytetyöpaikan tarjoamisesta. Erityiskiitokset haluan antaa Jani Laineelle, joka kärsivällisesti opasti minua opinnäytetyön aikana uusien asioiden opiskelussa sekä ohjelmoinnillisissa haasteissa.

Oulussa 13.1.2015

Tuomas Kinnunen

SISÄLLYS

SANASTO

AJAX	Lyhenne sanoista Asynchronous JavaScript And XML. Joukko web-sovelluskehityksen tekniikoita, joiden avulla web-sovelluksista voi tehdä vuorovaikutteisempia
Back End	Taustalla toimiva osa ohjelmasta, jonka kanssa käyttäjä ei ole suoraan tekemisissä
CSS	Lyhenne sanoista Cascading Style Sheets. On erityisesti WWW-dokumenteille kehitetty tyyliohjeiden laji
C#	Ohjemointikieli
Deserialisointi	Ohjelmassa olevan datan purkaminen merkkijonosta
HTML	Lyhenne sanoista Hypertext Markup Language. Web-sivujen kehityksessä käytettävä kuvauskieli
HTTP	Lyhenne sanoista Hypertext Transfer Protocol. Selaimien ja WWW-palvelimien tiedonsiirrossa käyttämä protokolla
JavaScript	Pääasiassa web-ympäristössä käytettävä dynaaminen komentosarjakieli

JSON	Lyhenne sanoista JavaScript Object Notation. Yksinkertainen avoimen standardin tiedostomuoto tiedonvälitykseen
Karttatiili	Englanniksi maptile. Esimerkiksi Maanmittauslaitoksen internetissä julkaisemia karttasivuja, joita käytetään karttasovelluksissa. Karttatiili sisältää yhden karttasivun
Serialisointi	Ohjelmassa olevan datan muuttaminen merkkijonoksi

1 JOHDANTO

Opinnäytetyöni lähtökohtana oli lähteä kehittämään mobiilisovellusta helpottamaan riistakolmiolaskennassa suoritettavaa tiedonkeruuta. Projektin tilaajana toimi Scatman Oy, jonka tarkoituksena on tulevaisuudessa jatkokehittää ja pilotoida kokonaisuutta maastoympäristössä.

Scatman Oy:n kehittämällä Scatman-järjestelmällä kartoitetaan öljyvahingon seurauksia rannoilla tai muualla maastossa. Suunnitelmana oli toteuttaa riistakolmiolaskenta-sovellus, joka tulisi toimimaan osakokonaisuutena Scatman-järjestelmässä. Riistakolmiolaskenta-sovellukseen käytettäisiin Scatman-järjestelmää kehittäessä hyväksi todettuja menetelmiä. Tällä mobiilisovelluksella riistakolmiolaskennan tiedonkeruu pystyttäisiin suorittamaan maastossa laskentaa suorittaessa. Lisäksi tiedot voidaan siirtää ja tallentaa palvelimelle automaattisesti. Sähköinen tiedonkeruu ja tallentaminen mahdollistavat nopeamman tilastoinnin ja raportoinnin riistatilanteesta.

Projektiin kokonaisuuteen sisältyy tämän opinnäytetyövaiheen aikana riistakolmiolaskenta-mobiilisovellus, web service -rajapinta sekä tietokanta. Järjestelmään on mahdollista lisätä myöhemmin ominaisuutena muita riistaeläinten kartoituksia, kuten esimerkiksi petohavainto- ja hirvilaskennat. Tulevaisuudessa sovellukseen voidaan toteuttaa web-sivusto tallennettujen tietojen tarkastelua varten.

2 RIISTAKOLMIOLASKENTA

2.1 Riistakolmiolaskennan perusteet

Riistakolmiolaskenta on 1980-luvulla kehitetty riistan laskentamuoto, jolla pidetään lukua erityisesti maamme metsäkanalintujen kannoista, sekä niiden muutoksista. Tämä on tärkeää riistakantojen tutkimisen, sekä sopivien saaliskiintiöiden suunnittelun takia. Laskennan järjestää Riista- ja kalatalouden tutkimuslaitos (RKTL) yhdessä Metsästäjäin Keskusjärjestön kanssa. Riistakolmiolaskennoilla on kolme ensiarvoisen tärkeää päämäärää:

- metsästyksen suunnittelun parantaminen määrittämällä saaliskiintiöitä laskennan tulosten perusteella
- tiedon hankkiminen riistaeläinten elinympäristövaatimuksista sekä elinympäristömuutosten vaikutuksista
- riistaeläinseurantojen kokoaminen saman runsausseurannan piiriin. Olennaista on kerätä tietoja useista lajeista samalla alueella, että ymmärretään niiden vaikutus toisiinsa. (1, linkit Kolmiolaskenta -> Mitä on kolmiolaskenta.)

Riistakantojen tunteminen on välttämätön edellytys kestävän metsästysverotuksen suunnittelussa. Pitkältä aikaväliltä tehdyt laskelmat antavat ymmärryksen eläinkantojen luontaisesta vuosivaihtelusta, ja mahdollistavat pitkäaikaismuutosten havaitsemisen. (1, linkit Kolmiolaskenta -> Mitä on kolmiolaskenta.)

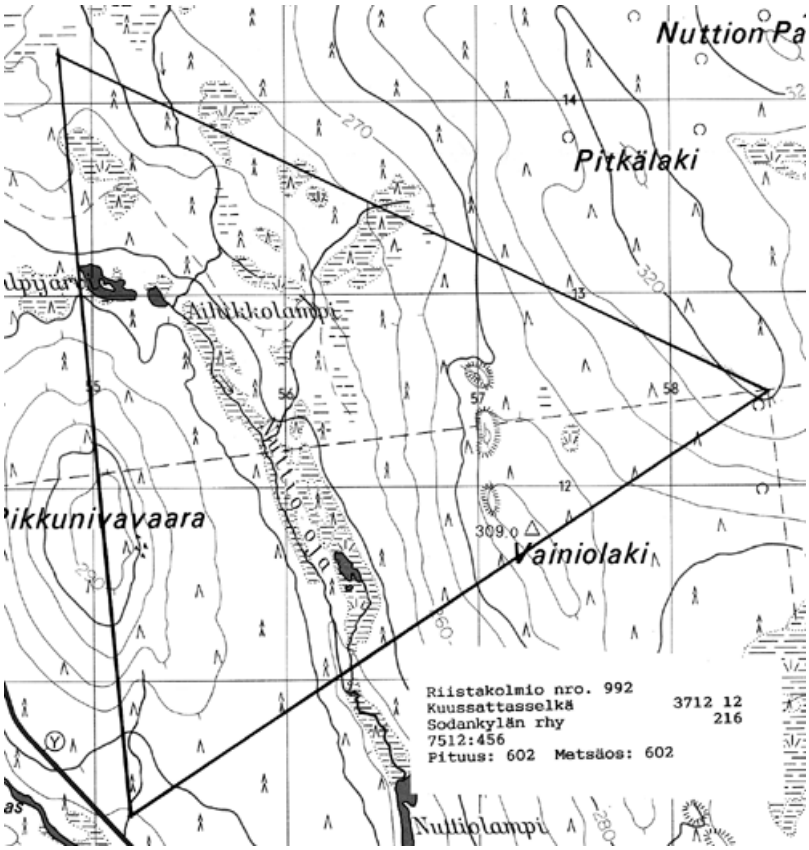
2.1.1 Historiaa

Riistantutkimus on tehnyt vuodesta 1945 lähtien ns. riistatiedustelun, jossa noin 500 havainnoitsijaa kerää havaintoja maamme riistaeläinten kannoista, sekä kantojen muutoksista edellisvuoteen verrattuna. Vielä 1980-luvun lopulla oltiin huonosti perillä aivan tavallistenkin riistalajien runsaudesta ja runsauden vaihtelusta. (1, linkit Kolmiolaskenta -> Mitä on kolmiolaskenta.)

1980-luvulla pyrittiin löytämään menetelmä, jolla saataisiin mahdollisimman monesta riistalajista kattava laskentatulos yhdellä laskentakerralla. Päädyttiin laskentamuotoon, jossa yhdelle karttasivulle piirretyllä kolmioalueella suoritettiin kolmion linjoja pitkin kuljettava riistalaskenta. (1, linkit Kolmiolaskenta -> Mitä on kolmiolaskenta.).

2.1.2 Riistakolmio

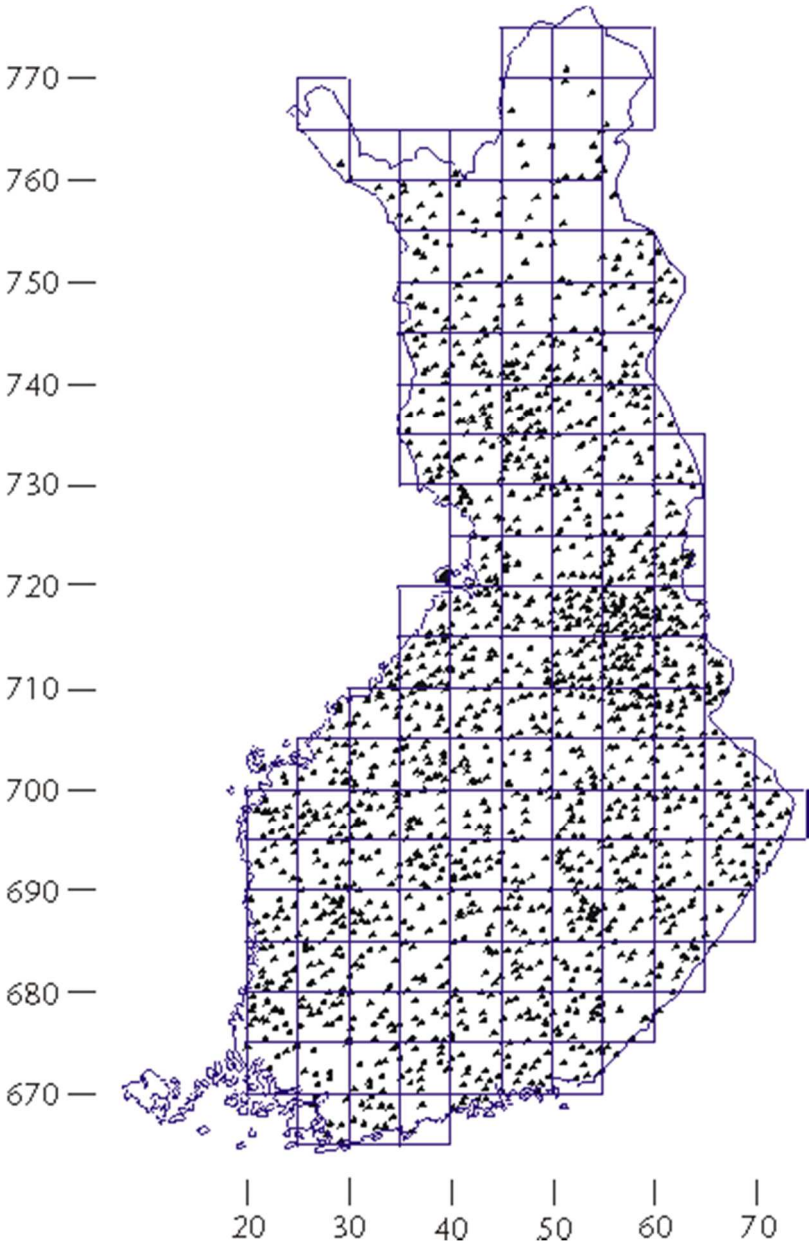
Riistakolmio on yhdelle karttasivulle piirretty tasasivuinen kolmio. Yhden kolmion sivun pituus on 4 km, ja siten yhdessä laskennassa kuljettava etäisyys on 12 km (kuva 1). Kolmioiden ideana on ollut alusta pitäen, että niiden paikat pysyvät muuttumattomina vuodesta toiseen. Vaikka alue muuttuisi sille tehtyjen metsähakkuiden tai rakentamisen takia, tulisi silti kulkea alkuperäistä riistakolmion reittiä. (9, s. 2.)



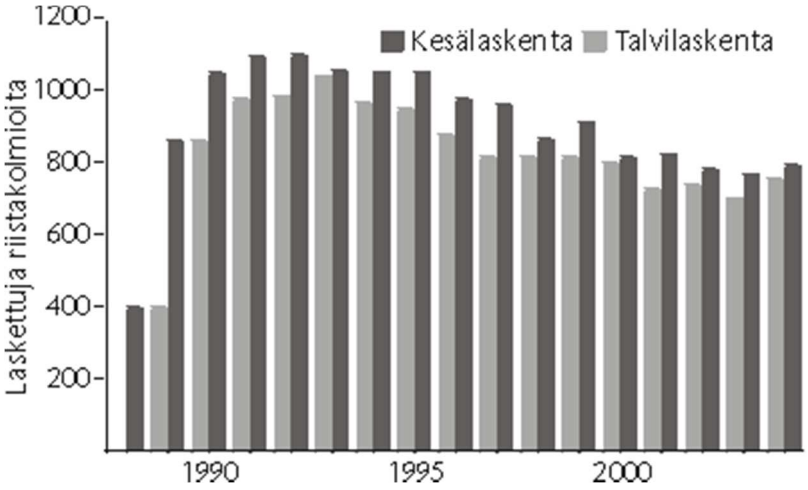
KUVA 1. Riistakolmio karttapohjalla (9, s. 3.)

2.1.3 Riistakolmioverkostot

Kolmiolaskennan suunnitteluvaiheessa arvioitiin noin 1200 riistakolmion riittävän koko maamme alueella takaamaan riittävän tietouden riistalajien kannanvaihtelusta. Alkuvuosina tavoite täyttyi, mutta nykypäivään mennessä riistakolmioita on autioitunut jonkin verran. Edelleenkin painotetaan riistakolmioiden pysyvyyttä. Vaikka riistakolmion maasto muuttuisi, on pyrittävä silti tekemään laskenta juuri samalla alueella vuodesta toiseen. Tällä saadaan tietoutta metsähakkuiden sekä muiden alueellisten muutosten vaikutuksesta riistaeläinkantoihimme. Kaikkiaan riistakolmioita maahamme on perustettu noin 1650 kappaletta (kuva 2), joista viimevuosina on laskettu noin 800, mikä on jäänyt jonkin verran alkuperäisestä tavoitteesta (kuva 3). (9, s. 4.)



KUVA 2. Riistakolmiot maassamme (9, s. 5.)



KUVA 3. Riistakolmiolaskennat maassamme (9, s. 6.)

2.2 Laskentamuodot

Riistakolmiolaskenta jaetaan kesä- sekä talvilaskentoihin, joita tässä luvussa käsitellään.

2.2.1 Kesälaskenta

Kesälaskenta suoritetaan aikavälillä 26.7–10.8, mutta mieluiten 4.8. mennessä. Laskenta tehdään yhden päivän aikana päiväsaikaan, mutta pyritään välttämään ääreviä säätiloja kuten kylmää, hellettä, voimakasta sadetta tai voimakasta tuulta. Laskennassa lasketaan näköhavainnot seuraavista metsäkanalinnuista: metso, teeri, riekko ja pyy. Lisäksi lasketaan havainnot metsäjäniksestä ja lehtokurpasta. (1, linkit Laskentaohjeet -> Kesälaskentaohje.)

Laskennassa edetään kolmen hengen rintamassa siten, että keskimmäinen henkilöistä suunnistaa ja pyrkii kulkemaan tarkasti kartalle merkittyä riistakolmion sivua pitkin. Laskijoiden rintamaketju laskee linnut yhteensä 60 metriä leveältä pääkaistalta. Tämä onnistuu parhaiten siten, että molemmat henkilöt suunnistajan lisäksi pitävät 20 metrin etäisyyden tähän hänen molemmin puolin, ja laskevat lentoon lähtevät linnut 10 metrin etäisyydeltä itsestään. Myös suunnistaja laskee 10 metrin etäisyydeltä molemmin puolin havaitut linnut. Tiheän metsän sekä maastonmuotojen takia etäisyys keskimieheen on usein vaikeaa pitää tarkalleen 20 metrissä. Tällöin on oleellista tarkistaa, lähtivätkö linnut lentoon 60 metrin pääkaistalta. Jos linnut lähtivät lentoon pääkaistan ulkopuolelta,

merkitään niiden havainto omalle sarakkeelleen lomakkeella (liite 1). Jos kolmiolinjalle osuu järvi, lampi tai pelto, ne kierretään, eikä kierron aikana tehtyjä havaintoja kirjata ylös. (1, linkit Laskentaohjeet -> Kesälaskentaohje.)

Kirjattavat havainnot jaetaan kahteen ryhmään: 60 metrin pääkaistalta havaitut linnut ja pääkaistan ulkopuolelta havaitut linnut, jotka kirjataan ylös omille sarakkeilleen. Tarkka havaintopaikka merkitään kartalle juoksevalla numerosarjalla, joka tulee olla sama kuin havaintolomakkeen rivinumero. Aikuisista linnuista käytetään havaintoa kirjatessa tunnuksia: *N* = naaras, *K* = koiras ja *O* = sukupuoli tuntematon. Poikasista käytetään tunnusta *p*. Täten esimerkiksi naaraspuolinen teeri ja kolme poikasta merkitään: *teeri N + 3p*. Jos poikasten lukumäärästä ollaan epävarmoja, kirjataan poikasten havainto sulkeisiin. Havainto kirjataan tällöin: *teeri N + (3p).* Jos osa poikasista on pääkaistan ulkopuolella ja osa sisäpuolella, merkitään havainto emolinnun havaintopaikan mukaisesti. Metso- ja teerikukot eivät osallistu poikueen hoitamiseen, mutta pyyn ja riekon molemmat emolinnut voivat olla poikueen läheisyydessä. Tässä tilanteessa merkintä kirjataan: *riekko NK + 5p.* Jos emolintua ei havaita ollenkaan, merkintä kirjataan: *riekko 3p.* (1, linkit Laskentaohjeet -> Kesälaskentaohje.).

2.2.2 Talvilaskenta

Talvilaskennassa keskitytään enimmäkseen lumijälkien laskemiseen. Kolmiolinjan ylittävät jäljet lasketaan seuraavista lajeista: metsäjänis, rusakko, orava, liito-orava, majava, piisami, susi, kettu, naali, supikoira, karhu, kärppä, lumikko, minkki, hilleri, näätä, ahma, mäyrä, saukko, ilves, villisika, valkohäntäpeura, hirvi, metsäpeura ja metsäkauris. Lisäksi lasketaan näköhavainnot seuraavista lajeista: metso, teeri, riekko, pyy, peltopyy, fasaani, kanahaukka ja korppi. (1, linkit Laskentaohjeet -> Talvilaskentaohje.)

Kolmion laskenta suoritetaan yhtenä päivänä, johon voi osallistua useita laskijoita. Laskennan tuloksiin vaikuttavat merkittävästi sen ajan säätilanne ja lumitilanne, jolloin jäljet ovat syntyneet. Varsinaista laskentaa ennen tehdään ns. kolmion esikierto, jossa lumeen merkataan näkyvästi jokainen kolmiolinjan ylittävä jälkihavainto. Varsinainen laskenta suoritetaan vuorokauden jälkeen esikier-

rosta, jonka aikana kirjataan ylös kaikki esikierron jälkeen syntyneet jälkihavainnot. Jos esikiertoa ei tehdä, laskenta tulee suorittaa 1–2 vrk:n kuluttua sellaisesta lumisateesta, joka on peittänyt siihen asti syntyneet jäljet. Talvilaskennassa kolmiolinjalla olevat järvet, lammet ja pellot pyritään ylittämään ja laskemaan jäljet niiltäkin osuuksilta. Mikäli alue joudutaan kiertämään, kierron aikana havaittuja jälkiä tai näköhavaintoja ei kirjata. (1, linkit Laskentaohjeet -> Talvilaskentaohje.)

Jälki -sekä näköhavainnot merkitään kartan kolmiolinjalle lyhyillä poikkiviivoilla, numeroidaan ja selitetään havaintolomakkeella (liite 2). Jos samassa kohdassa havaitaan useita jälkiä, merkitään ne ainoastaan yhdellä poikkiviivalla sekä tarvittavalla määrällä numeroita. Jokainen numero kirjataan havaintolomakkeelle silti omalle rivilleen. Laskennassa kirjataan ainoastaan jäljet jotka risteävät kulku-uran kanssa. Poikkeuksena susi ja kettu, joiden jäljet voidaan kirjata ylös, mikäli nämä ovat selvästi varoneet ylittämästä esikiertouraa. (1, linkit Laskentaohjeet -> Talvilaskentaohje.).

2.3 Riistakolmiolaskennan tulokset

Ainutlaatuiset riistakolmioaineistot ovat nostaneet suomen riistatutkimuksen kansainväliseen kärkeen. Tietoisuus on kasvanut riistavarojen ja metsästyksen välisestä suhteesta. Tällä tarkoitetaan metsästyksen ekologista kestävyyttä, eli riistalajien monimuotoisuuden säilyvyyttä. (1, linkit Kolmiolaskenta -> Mitä on kolmiolaskenta.)

Riistakolmiolaskentojen tulokset ovat antaneet hyvät edellytyksen riistalajien kannanvaihteluiden tutkimiseen. Esimerkiksi metsäkanalintujen kannanvaihtelut ovat osoittautuneet jaksottaisiksi. Kantojen huiput ja aallonpohjat ovat seuranneet toisiaan keskimäärin 6–7 vuoden välein. (1, linkit Kolmiolaskenta -> Mitä on kolmiolaskenta.)

Saadusta aineistosta on myös pystytty tutkimaan petojen ja saaliseläinten välistä vuorovaikutusta eripuolella maata. Esimerkiksi on pystytty tutkimaan petoeläinten käyttäytymistä alueilla, joilla saaliseläimiä on niukemmin saatavilla. Metsästäminen lasketaan myös saalistukseksi, ja tähän riistakolmiolaskentojen

tulokset ovat vaikuttaneet saaliskiintiömuutoksilla, metsästysrajoituksilla ja metsästyspoliittisilla muutoksilla. (1, linkit Kolmiolaskenta -> Mitä on kolmiolaskenta.)

Kolmiolaskentojen tuloksista ovat hyötyneet sekä EU, maa- ja metsätalousministeriö että metsästysseurat. Tietoja käytetään myös ympäristöhallinnossa sekä maa- ja metsävarojen käytön suunnittelussa. (1, linkit Kolmiolaskenta -> Mitä on kolmiolaskenta.).

3 SOVELLUKSEN SUUNNITTELU

Tässä pääluvussa esitetään vaatimusmäärittely sovellukselle sekä kuvataan sovelluksen pääosilta vaadittavat toiminnot.

3.1 Vaatimusmäärittely

3.1.1 Käyttäjävaatimukset

Käyttäjävaatimuksilla tarkoitetaan toimia, jotka käyttäjän tulee pystyä toteuttamaan sovellusta käyttäessä. Mobiilikäyttöliittymän käyttäjävaatimuksia ovat

- sovellukseen kirjautuminen
- riistakolmion lataaminen
- pääsy kaikkiin käyttäjän organisaation riistakolmioihin
- oman sijainnin näkeminen kartalla
- havainnon lisääminen
- havaintojen lähettäminen palvelimelle.

3.1.2 Toiminnalliset vaatimukset

Toiminnalliset vaatimukset ovat sovelluksen toimimiseen edellytettäviä vaatimuksia järjestelmältä. Toiminnallisia vaatimuksia ovat

- tiedon tallentaminen tietokantaan
- tiedon lukeminen tietokannasta
- kaksisuuntainen ja tietosuojattu tiedonsiirto mobiililaitteen ja palvelimen välillä
- mobiililaitteen paikkatiedon (GPS) käyttäminen
- mobiililaitteen paikallisen tallennustilan käyttäminen.

3.1.3 Ei-toiminnalliset vaatimukset

Ei-toiminnalliset vaatimukset liittyvät sovelluksen käytettävyyteen. Ei-toiminnallisia vaatimuksia ovat

- sovelluksen helppo käytettävyys maasto-olosuhteissa

- sovelluksen tulee toimia mahdollisimman monella mobiililaitteella
- sovellus tulee suojata sisäänkirjautumisella käyttäjätunnuksen ja salasanan avulla.

3.2 Tietokannan määrittely

Sovelluksessa käytettävä data sijaitsee tietokannassa (liite 3) olevissa tauluissa. Tietokannasta tulee löytyä tallennusmahdollisuus seuraaville tiedoille:

- käyttäjätiedot
- havaitun riistaeläimen tiedot
- riistakolmion paikkatiedot
- organisaation tiedot

Sovelluksen tietokanta koostuu seuraavista tauluista:

- User
- Organization
- Sighting
- SightingType
- OrganizationTriangle
- Triangle

User-taulu sisältää käyttäjän perustiedot, sekä tämän kirjautumistiedot sovellukseen. Organization-taulu sisältää tiedot eri organisaatioista. Näitä voivat olla esimerkiksi eri yritykset tai riistanhoitoyhdistykset. Tämä taulu mahdollistaa organisaatiolle määrättyjen riistakolmioiden tietojen linkittymisen käyttäjälle, joka kuuluu tähän organisaatioon. Sighting-taulu sisältää tiedot riistaeläimistä tehdyistä havainnoista. SightingType-taulu sisältää eri havaintotyypit, esimerkiksi kesä- ja talvihavainnot. Tämä taulu antaa mahdollisuuden lisätä sovellukseen uusiä havaintotyyppejä ilman muutosten tekemistä käyttöliittymäkoodiin. OrganizationTriangle-taulu määrittää mitkä riistakolmiot kuuluvat millekkin organisaatiolle. Triangle-taulu sisältää tiedon riistakolmion paikkatiedoista. Opinnäytetyön liitteet osioista löytyy tarkka kuvaus tietokannan rakenteesta (liite 3).

3.3 Palvelimen rajapinnan määrittely

Palvelimen rajapinta toimii tiedonkäsittelijänä mobiilikäyttöliittymän ja tietokannan välillä. Tässä sovelluksessa palvelimen rajapinnalta vaaditaan vähintään seuraavat kolme toimintoa:

- sisäänkirjautumisessa rajapinnan on pystyttävä tunnistamaan käyttäjä ja palauttamaan tälle tietoa.
- havainnon lisäämisessä rajapinnan on pystyttävä tallentamaan käyttäjän havaitsemat tiedot
- havainnon päivittämisessä rajapinnan on pystyttävä hakemaan tietokannassa jo oleva havaintotieto ja päivittää se

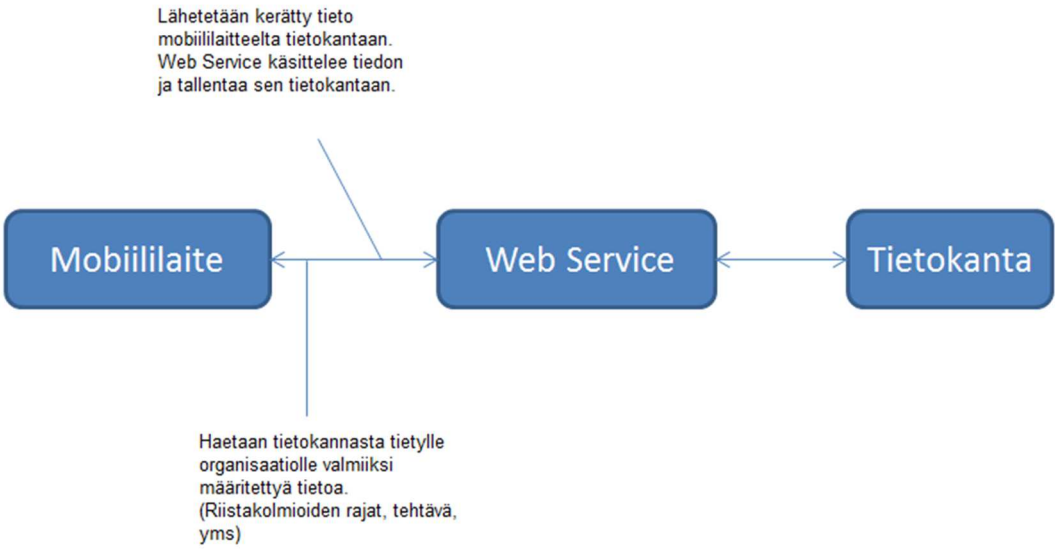
3.4 Mobiilikäyttöliittymän määrittely

Mobiilikäyttöliittymältä vaaditaan seuraavat toiminnot:

- sisäänkirjautuminen
- riistakolmion valinta
- kartan näyttäminen
- paikkatiedon määrittäminen
- riistakolmion piirtäminen kartalle
- käyttäjän paikan piirtäminen kartalle
- havaintojen merkkaus kartalle
- tiedon tallennus paikallisesti
- tiedonsiirto palvelimelle

4 SOVELLUKSEN TOIMINNAN KUVAUS

Kuva 4 antaa peruskäsityksen sovelluksen rakenteesta.



KUVA 4. Sovelluksen rakenne

4.1 Sovelluksen toiminta yleisesti

Alussa sovellus kysyy käyttäjätunnuksen ja salasanan. Käyttäjän syöttämät tiedot välitetään palvelimelle, jossa tunnuksen ja salasanan oikeellisuus varmistetaan. Käyttäjän organisaation riistakolmiotiedot ja niiden karttatiilet välitetään palvelimelta mobiililaitteelle, joka tallentaa tiedot paikalliseen muistiin. Kirjautumisvaihe vaatii toimivan tietoliikenneyhteyden palvelimeen, mutta kun tiedot on kertaalleen haettu mobiililaitteeseen, voidaan riistakolmiolaskenta suorittaa offline-tilassa. Palvelimelta haetut riistakolmion tiedot mahdollistavat nyt riistakolmion piirtämisen karttatiilistä muodustuneen kartan päälle. Mobiililaitteen gps-toiminto määrittää tämän jälkeen laskennan suorittajan oman paikan kartalle. Kun laskennan suorittaja liikkuu, sovellus piirtää tämän kulkeman reitin mobiililaitteen kartalle. Tällä tavalla käyttäjän on helppo suunnistaa pitkin riistakolmion sivua. Sovellus tallentaa käyttäjän tekemien havaintojen paikkatiedot, ajan sekä

käyttäjän tekemät valinnat mobiililaitteen paikalliseen muistiin. Tallennettu havainto jää näkymään omana merkkinä kartalle. Laskenta suoritetaan tällä tavalla loppuun, jonka jälkeen käyttäjä voi lähettää paikalliseen muistiin tallennetut havaintotiedot palvelimen tietokantaan. Myös tähän vaiheeseen vaaditaan tietoliikenneyhteys.

4.2 Palvelimen rajapinta

Palvelimen rajapinnan tehtävänä sovelluksessa on käsitellä sekä välittää tietoa mobiililaitteen ja tietokannan välillä. Tässä järjestelmässä rajapintatoimintoja tarvitaan vähintään kolme, jotta sovelluksen toiminta on mahdollista.

Kirjautumisvaiheessa verrataan mobiililaitteen lähettämiä kirjautumistietoja tietokannassa oleviin tietoihin. Mikäli kirjautumistiedot täsmäävät, palautetaan mobiililaitteelle käyttäjän organisaation tiedot. Kirjautuminen epäonnistuu mikäli tiedot poikkeavat.

Havainnon lisäyksessä tarkistetaan vielä käyttäjätiedot, eli tunnistetaan mobiililaitteen käyttäjä. Seuraavaksi tarkistetaan havainnon tietojen oikeellisuus. Lopuksi rajapintatoiminto tallentaa havaintotiedon tietokantaan.

Havaintojen päivittäminen toimii hyvin samankaltaisesti kuin havainnon lisäys. Päivitystoiminto hakee tietokannasta havainnon avaintiedolla olemassa olevan havainnon. Sitten se korvaa olemassa olevan havainnon tiedot uusilla ja tallentaa muutokset tietokantaan.

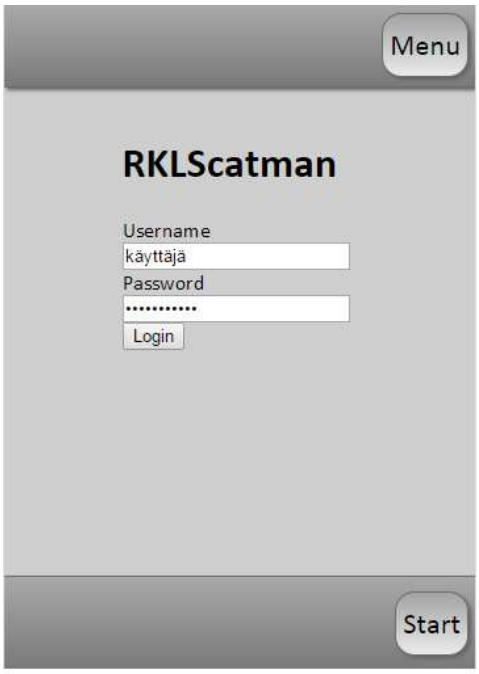
4.3 Käyttöliittymä

Tässä osiossa kuvataan mobiilikäyttöliittymän toiminta askel askeleelta.

4.3.1 Sisäänkirjautuminen

Sovelluksen käynnistyessä ensimmäisen kerran käyttäjä ohjataan sisäänkirjautumiseen (kuva 5). Käyttäjä syöttää käyttäjätunnuksensa sekä salasanansa. Onnistuneen sisäänkirjautumisen jälkeen käyttäjä ohjataan riistakolmion valintanäytölle. Tässä vaiheessa sovellus on hakenut käyttäjän organisaation tiedot tietokannasta ja tallentanut ne laitteen paikalliseen muistiin. Tämä mahdollistaa

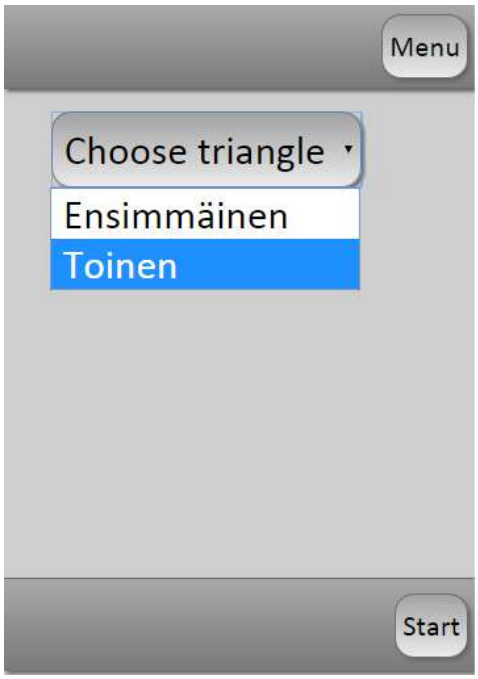
laitteen käytön offline-tilassa. Mikäli käyttäjä ei kirjaudu ulos palvelusta, niin sovelluksen käynnistyessä seuraavan kerran käyttäjä ohjataan suoraan riistakolmion valintanäytölle.



KUVA 5. Sisäänkirjautuminen mobiilikäyttöliittymässä

4.3.2 Riistakolmion valinta

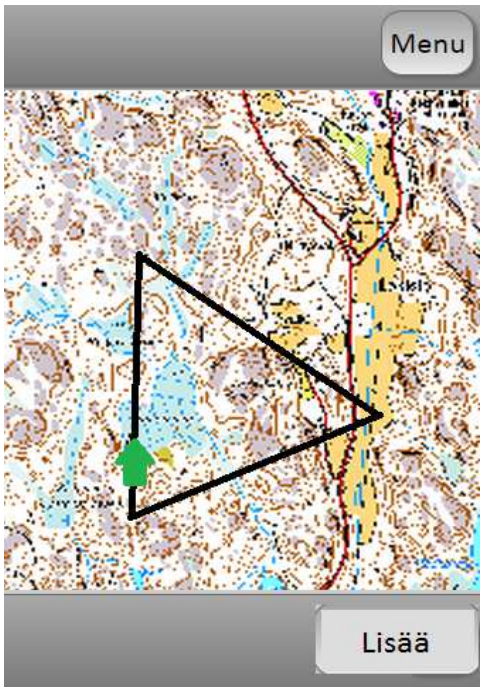
Seuraavana näkymänä aukeaa riistakolmion valintasivu (kuva 6). Tältä sivulta löytyy alasvetovalikko, jonka vaihtoehtoina on tietokannasta haetut riistakolmiotiedot. Mikäli käyttäjän organisaation tiedoista löytyy useampi kuin yksi riistakolmio, tulee käyttäjän tässä vaiheessa valita, minkä niistä haluaa laskea. Valinnan tehtyä riistakolmiolaskenta alkaa ja käyttäjä ohjataan karttanäkymään.



KUVA 6. Riistakolmion valinta mobiilikäyttöliittymässä

4.3.3 Karttanäkymä

Karttanäkymässä (kuva 7) näkyy kartalle piirrettynä riistakolmion sivut sekä käyttäjän sijainti. Käyttäjä lähtee suunnistamaan pitkin riistakolmion sivua, ja mikäli hän poikkeaa linjalta, on hänen helppo palata takaisin kävelemään oikealle kohdalle. Kun käyttäjä havaitsee riistaeläimen hän painaa Lisää-painiketta ja hänet ohjataan havainnon lisäyssivulle. Karttanäkymästä selviää myös käyttäjän tekemät riistaeläinhavainnot, jotka näkyvät kartalla omina merkkeinä. Merkkiä painamalla käyttäjä pääsee takaisin kyseisen havainnon lisäykseen, jossa hän voi tarvittaessa muokata virheelliset tiedot oikeiksi.



KUVA 7. Karttanäkymä

4.3.4 Havainnon lisääminen

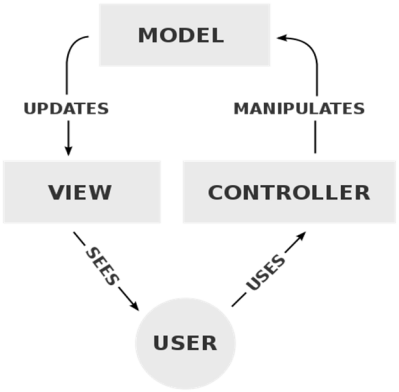
Havainnon lisäyssivulla (kuva 8) käyttäjä valitsee sivulla näkyvistä alasvetovalikoista oikeat riistaeläimen tiedot sekä mahdollisten poikasten lukumäärän. Valinnat tehtyä hän tallentaa tiedot mobiililaitteen paikalliseen muistiin. Tässä vaiheessa käyttäjä ohjataan takaisin karttanäkymään, ja juuri tehty havainto näkyy karttanäkymässä omana merkkinä.

KUVA 8. Havainnon lisääminen mobiilikäyttöliittymässä

5 TOTEUTUS

5.1 ASP.NET MVC

Sovelluksen toteutuksessa työkaluna käytettiin Microsoft Visual Studio-ohjelmistokehitysympäristöä. Sovellus tehtiin web-sovelluksena käyttäen ASP.NET MVC -ohjelmistokehystä. Sovelluksen graafisen ulkoasun rakenteen ja tyylittelyn toteutukseen käytettiin HTML-kuvauskieltä (Hyper Text Markup Language) ja CSS-tyylisääntöjä (Cascading Style Sheets). Käyttöliittymän dynaaminen toiminta toteutettiin JavaScript-ohjelmointikielellä ja palvelimen rajapinta ohjelmoitiin C#-kielellä.



KUVA 9. MVC-toiminta (3.)

ASP.NET MVC -ohjelmistokehys on Microsoftin tekemä ja sitä käytetään erityisesti web-sovellusten käyttöliittymien toteutuksessa. MVC tulee sanoista Model, View, Controller eli malli, näkymä ja ohjain (kuva 9). Mallit hallitsevat sovelluksen dataa ja ilmoittavat näkymille ja ohjaimille, mikäli sen tilassa havaitaan muutosta. Näkymät pyytävät tietoa malleilta ja toteuttavat käyttäjälle näkyvän osan sovelluksesta. Ohjaimet ottavat vastaan käyttäjän komentoja ja muuntavat ne komennoiksi näkymille tai malleille.

5.2 Tietokanta

Sovelluksen tietokannan hallintaan käytettiin Microsoftin SQL Server -sovellusta, joka on RDBMS-tyyppinen (relational database management system).

Sitä käytettiin tämän projektin aikana, kun haluttiin nähdä mitä muutoksia tietokantaan oli todellisuudessa syntynyt.

Tässä sovelluksessa tietokanta on luotu code first -menetelmällä, missä tietokanta määritellään model-luokkina ja varsinainen tietokanta luodaan näiden pohjalta automaattisesti (2). Käytännössä koodin model-luokka vastaa siis tietokannan taulua ja kuvaa sen avaimet ja relaatiot muihin tauluihin. Seuraavaksi esitetään prosessi tietokannan User-taulun toteutusvaiheista. Kaikki sovelluksen tietokannan taulut on luotu käyttäen samaa periaatetta.

Ensimmäisenä vaiheena luodaan käyttäjätiedoille model-luokka, johon tulee kaikki käyttäjälle tarvittavat tiedot. Kuva 10 esittää sovelluksen UserModel-luokan.

```
namespace ScatmanHunter.Models
{
    [Table("User")]
    public class UserModel
    {
        [Key]
        [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
        public int UserId { get; set; }

        [Required(ErrorMessageResourceName = "Required", ErrorMessageResourceType = typeof(ScatmanHunter.Resources.Errors.Strings))]
        [ForeignKey("Organization")]
        public int OrganizationId { get; set; }

        [Required(ErrorMessageResourceName = "Required", ErrorMessageResourceType = typeof(ScatmanHunter.Resources.Errors.Strings))]
        public string Username { get; set; }

        [Required(ErrorMessageResourceName = "Required", ErrorMessageResourceType = typeof(ScatmanHunter.Resources.Errors.Strings))]
        [MinLength(8, ErrorMessageResourceName = "MinLength", ErrorMessageResourceType = typeof(ScatmanHunter.Resources.Errors.Strings))]
        public string Password { get; set; }

        [Required(ErrorMessageResourceName = "Required", ErrorMessageResourceType = typeof(ScatmanHunter.Resources.Errors.Strings))]
        public string FirstName { get; set; }

        [Required(ErrorMessageResourceName = "Required", ErrorMessageResourceType = typeof(ScatmanHunter.Resources.Errors.Strings))]
        public string LastName { get; set; }

        public string Email { get; set; }

        public string Phonenumber { get; set; }

        //Navigation properties
        public virtual OrganizationModel Organization { get; set; }
    }
}
```

KUVA 10. Käyttäjätietojen model-luokka

Toisena vaiheena luodaan DataContext-luokka, joka määrittelee sovelluksen muistiin model-luokkien avulla tietokantaa vastaavan rakenteen. Se myös generoi valmiiksi yhteyskoodin tietokantaan, mikä sallii sovelluksesta tehdyt tietokantakyselyt. DataContext seuraa automaattisesti sovelluksen tekemiä muutoksia muistissa oleviin model-ilmentymiin ja välittää muutokset transaktioina varsinai-

seen tietokantaan. Luokka vaatii toimiakseen Entity Framework -luokkakirjaston. Entity Framework vähentää kirjoitettavan koodin määrää merkittävästi tietokantakyselyiden toteutusta tehdessä.

Koodirivi, jossa käsitellään UserModel-luokkaa, löytyy kuvasta 11 tummennettuna kohtana.

```
namespace ScatmanHunter.DAL
{
    public class DataContext : DbContext
    {
        public DataContext()
            : base("ScatmanHunterConnStr")
        {
        }

        public DbSet<OrganizationModel> Organizations { get; set; }
        public DbSet<UserModel> Users { get; set; }
        public DbSet<OrganizationTriangleModel> OrganizationTriangles { get; set; }
        public DbSet<TriangleModel> Triangles { get; set; }
        public DbSet<SightingModel> Sightings { get; set; }
        public DbSet<SightingTypeModel> SightingTypes { get; set; }
    }
}
```

KUVA 11. DataContext-luokka

Kun tietokanta halutaan luoda tai mallin tietoja muutetaan, ajetaan migraatio. Migraatio on kokoelma muutoksia verrattuna jo olemassa olevaan tietokantaan. Migraatiot antavat mahdollisuuden muokata tietokantaa vaiheittain muutos muutokselta. Seuraavassa kuvassa (kuva 12) oleva koodi on otettu migraatiotiedostosta, ja se luo tietokantaan User-taulun tarvittavine tietoineen.

```
CreateTable(
    "dbo.User",
    c => new
        {
            UserId = c.Int(nullable: false, identity: true),
            OrganizationId = c.Int(nullable: false),
            Username = c.String(nullable: false),
            Password = c.String(nullable: false),
            FirstName = c.String(nullable: false),
            LastName = c.String(nullable: false),
            Email = c.String(),
            Phonenumber = c.String(),
        })
    .PrimaryKey(t => t.UserId)
    .ForeignKey("dbo.Organization", t => t.OrganizationId, cascadeDelete: true)
    .Index(t => t.OrganizationId);
```

KUVA 12. Migraatio User-taululle

5.3 Palvelimen rajapinta

Palvelimen rajapinta (web service) muodostuu ohjaimista (controller), jotka käsittelevät mobiililaitteelta tulevia http-kyselyitä ja palauttavat http-vastauksia. Seuraavaksi esitetään rajapinnan toteutusta vaihe vaiheelta, koodia avuksi käyttäen.

Sisäänkirjautumisessa kutsutaan ohjaimesta löytyvää login-funktiota (kuva 13), joka ottaa vastaan mobiililaitteen lähettämän JSON (JavaScript Object Notation) -muotoisen merkkijonon. Merkkijono eli data-muuttuja sisältää käyttäjän syöttämät kirjautumistiedot. JSON-muotoinen merkkijono täytyy deserialisoida rajapinnan funktiossa koodiluokkien ilmentymiksi, jotta sen sisältämiä tietoja voidaan käyttää. Login-funktion tapauksessa merkkijono deserialisoidaan UserModel-luokan ilmentymäksi. GetUserprofileByUsernameAndPassword-funktio ottaa parametrina data-muuttujasta deserialisoidun UserModel-ilmentymän ja palauttaa uuden ilmentymän, joka asetetaan user-muuttujaan. Funktio hakee tietokannasta käyttäjän syöttämiä tietoja vastaavat käyttäjätiedot ja palauttaa ne. Palautusarvo on null, mikäli kirjautumistiedoilla ei löydy käyttäjätietoja. Onnistuneen käyttäjätietojen haun jälkeen haetaan käyttäjän organisaatiotiedot GetOrganizationDataByUser-funktiolla (kuva 14). JSON-apufunktio serialisoi tiedot JSON-muotoiseksi merkkijonoksi ja ne palautetaan takaisin mobiililaitteelle. Virhetilanteessa palautetaan JSON-muotoinen virheviesti.

```
public ActionResult login(string data)
{
    //validate credentials and get user data
    UserModel user = BLL.GetUserprofileByUsernameAndPassword(deserializeJSON<UserModel>(data));
    if (user == null)
    {
        return Json(new { IsError = true, ErrorMsg = "Invalid username or password" });
    }

    //get organization data
    OrganizationDataViewModel organizationData = BLL.GetOrganizationDataByUser(user);
    if (organizationData == null)
    {
        return Json(new { IsError = true, ErrorMsg = "Failed to fetch user's organization data" });
    }

    return Json(new { IsError = false, Data = organizationData });
}
```

KUVA 13. Login-rajapintafunktio

GetOrganizationDataByUser-funktio (kuva 14) tekee varsinaiset tietokantakyselyt. Funktio ottaa parametrina vastaan UserModel-tyyppisen ilmentymän user. Tässä vaiheessa luodaan dataContext-muuttuja, jolla voidaan ottaa yhteys tietokantaan. Ensimmäinen tietokantakysely hakee käyttäjän käyttäjätiedot sekä organisaation tiedot, ja ne tallennetaan organizationData-ilmentymään. Toinen tietokantakysely hakee käyttäjän organisaatiolle kuuluvat riistakolmiotiedot tallentaen ne listana organizationData-ilmentymään. Lopuksi funktio palauttaa organizationData-ilmentymän login-funktiolle.

```
public static ViewModels.OrganizationDataViewModel GetOrganizationDataByUser(Models.UserModel user)
{
    try
    {
        using(DAL.DataContext dataContext = new DAL.DataContext)
        {
            //Get organization and user
            ViewModels.OrganizationDataViewModel organizationData =
                (from u in dataContext.Users
                join o in dataContext.Organizations on u.OrganizationId equals o.OrganizationId
                where u.UserId == user.UserId
                select new ViewModels.OrganizationDataViewModel()
                {
                    User = u,
                    Organization = o
                }).Single();

            //Get organization triangles
            organizationData.Triangles =
                (from ot in dataContext.OrganizationTriangles
                join t in dataContext.Triangles on ot.TriangleId equals t.TriangleId
                where ot.OrganizationId == organizationData.OrganizationId
                select t
                ).ToList();

            return organizationData;
        }
    }
    catch
    {
        return null
    }
}
```

KUVA 14. GetOrganizationDataByUser-funktio

Havainnon lisäämisen tietokantaan suorittava insertSighting-funktio (kuva 15) toimii pääpiirteittäin samalla tavalla login-funktion kanssa. Kuvan 15 koodi kuvaa funktion toiminnan.

```
public ActionResult insertSighting(string data)
{
    //deserialize data
    SightingViewModel viewModel = deserializeJSON<SightingViewModel>(data);
    if (viewModel == null)
    {
        return Json(new { IsError = true, ErrorMsg = "Invalid data" });
    }

    //validate user credentials
    if (!BLL.CheckUsernameAndPassword(viewModel.Username, viewModel.Password))
    {
        return Json(new { IsError = true, ErrorMsg = "Invalid username or password" });
    }

    //save sighting to database
    if (!BLL.insertSighting(viewModel.Sighting))
    {
        return Json(new { IsError = true, ErrorMsg = "Saving data failed" });
    }

    return Json(new { IsError = false });
}
```

KUVA 15. insertSighting-funktio

5.4 Käyttöliittymä

5.4.1 Single-page application

Sovelluksen käyttöliittymä toimii SPA (Single-page application) -tyyppisenä web-sovelluksena. SPA:n ideana on sovelluksen toimiminen käyttäen ainoastaan yhtä web-sivua. Tavoitteena on saada sulavampi käyttäjäkokemus verrattuna perinteiseen monisivuiseen applikaatioon. SPA-tyyppisessä sovelluksessa kaikki tarvittava koodi ladataan joko yhdellä sivunlatauksella, tai sovelluksen vaatimat resurssit ladataan dynaamisesti ja käytetään myöhemmin sivulla. (4.)

5.4.2 Paikallinen muisti sivunvaihdossa

Tässä sovelluksessa käyttöliittymä lataa näytettävien sivujen HTML-koodit mobiililaitteen paikalliseen muistiin (Local storage). Paikallinen muisti pitää sisällään tiedon (value) sekä viiteavaimen (key), jolla tietoon pystytään viittaamaan. Paikallisen muistin toimintoihin tarvitaan tässä sovelluksessa tiedonhaku- sekä tiedontallennusfunktio (kuva 16). Funktiot on toteutettu käyttäen JavaScript-koodikieltä. Tiedonhaku tapahtuu käyttäen getFromLocalStorage-funktiota. Funktio ottaa vastaan viiteavaimen, jonka perusteella se hakee paikallisesta muistista viiteavainta vastaavan tiedon. Tiedon tallentaminen taas tapahtuu saveToLocalStorage-funktion avulla. Tämä ottaa vastaan sekä viiteavaimen että tiedon, ja tallentaa molemmat paikalliseen muistiin. Olemassa olevat tiedot ylikirjoitetaan mikäli, viiteavaimella löytyy aiemmin tallennettua tietoa.

```
//Local storage

function getFromLocalStorage(key) {

    var value = localStorage.getItem(key);

    if (value !== null || value !== "null") {
       // alert("something in local storage");
        return value;
    }
    else {
        return null;
    }

}

function saveToLocalStorage(key, value) {

    if (key !== null || key !== null) {
        localStorage.setItem(key, value);
    }
    else {
        return null;
    }

}
```

KUVA 16. Local storage-funktiot

MainPage-sivu (kuva 17) toimii tässä sovelluksessa SPA-tyylisesti ainoana näytettävänä sivuna. Sivu on toteutettu käyttäen HTML-kuvauskieltä sekä CSS-tyylisääntöjä. Sivun content-area-säiliöön ladataan paikallisesta muistista kulloinkin näytettävän toiminnon HTML-koodit. Alla olevan koodin head-osiossa määritetty ClientScripts.js-tiedosto sisältää käyttöliittymään toteutetut JavaScript-koodit. ClientStyles.css-tiedosto sisältää toteutetut CSS-tyylisäännöt. Tiedostot määriteltyä MainPage-sivu ymmärtää käyttää ClientScripts.js-tiedostoa sekä ClientStyles.css-tiedostoa.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <link rel="stylesheet" type="text/css" href="/Data/ClientStyles.css">
    <script type="text/javascript" src="http://code.jquery.com/jquery-1.9.1.js"></script>
    <script type="text/javascript" src="/Data/ClientScripts.js" ></script>
    <script type="text/javascript">

        $(document).ready(function () {
            changePage('LoginPage');
        });
    </script>
</head>
<body>

        <div class="menu-area">
            <button class="menu-button">Menu</button>
        </div>
        <div class="content-area" id="content-area">
        <!-- Tämän sisällöksi vastaanotetaan html-koodit local storagesta -->
        </div>
        <div class="bottom-menu-area">
            <button onclick="changePage('SightingPage')" class="main-button">Start</button>
        </div>
</body>
</html>
```

KUVA 17. MainPage-sivu

Kun mobiilikäyttöliittymän sivua vaihdetaan, kutsutaan JavaScript-funktiota changePage (kuva 18). Funktio ottaa vastaan viiteavaimen (key), jonka perusteella mobiililaitteen paikallisesta muistista haetaan sitä vastaava tieto. Haettu tieto sisältää sivun HTML-koodin ja se asetetaan MainPage-sivun content-area-säiliöön.

```
function changePage(key) {
    var page = getFromLocalStorage(key)
    $('#content-area').html(page);
}
```

KUVA 18. changePage-funktio

5.4.3 Käyttöliittymän yhteys palvelimen rajapintaan

Käyttöliittymä tarvitsee tietokannasta löytyvää tietoa sisäänkirjautumisessa sekä havainnon lähettämisessä. Käyttöliittymä kommunikoi palvelimen rajapinnan kanssa käyttäen JavaScript-kielellä toteutettuja funktioita (kuva 19).

Sisäänkirjautumisessa käyttöliittymän login-funktion AJAX-käsky lähettää palvelimen rajapinnan login-funktiolle (kuva 13) kirjautumistiedot JSON-muodossa. Onnistuneen kirjautumistietojen tarkistuksen jälkeen funktio ottaa vastaan käyttäjän organisaatiolle kuuluvat tiedot ja tallentaa ne mobiililaitteen paikalliseen

muistiin. Virhetilanteissa käyttäjälle näytetään joko kirjautumisen epäonnistumisesta tai yhteysvirheestä johtuvat virheviestit.

Havainnon lähettämisessä käyttöliittymä suorittaa sendSighting-funktion. Funktion AJAX-käsky lähettää rajapinnan insertSighting-funktiolle (kuva 15) käyttäjätiedot sekä havainnon tiedot JSON-muodossa. Virhetilanteissa käyttäjälle näytetään virheviestit.

```
function Scatman() {

    this.login = function(username, password) {
        $.ajax({
            url: 'http://localhost/api/login',
            type: 'POST',
            data: { username: username, password: password }
        }).success(function (msg) {
            if (msg.IsError == true) {
                //login failed
                //show error message
                showMessage(msg.ErrorMsg);
            }
            else {
                //successfull login
                //save organization data to local storage
                saveToLocalStorage('organization_data', msg.Data);
            }
        }).error(function () {
            //show error message
            showMessage('Cannot connect to server');
        });
    }

    //send sighting to server (sigthing data is json object)
    this.sendSighting = function(username, password, sighting) {
        $.ajax({
            url: 'http://localhost/api/insertSighting',
            type: 'POST',
            data: { Username: username, Password: password, Sighting: sighting }
        }).success(function (msg) {
            if (msg.IsError == true) {
                //show error message
                showMessage(msg.ErrorMsg);
            }
            else {
                //successfull insert
                showMessage('Sighting saved');
            }
        }).error(function () {
            //show error message
            showMessage('Cannot connect to server');
        });
    }
```

KUVA 19. Käyttöliittymän rajapintafunktiot

5.4.4 PhoneGap

Erilaisten mobiilikäyttöjärjestelmien vuoksi on kehitetty useita ohjelmistokehyksiä ja kirjastoja, jotka mahdollistavat saman sovelluskoodin kääntämisen usealle eri käyttöjärjestelmälle. Käyttämällä näitä ratkaisuja vältytään koodin uudelleenkirjoittamiselta, kun halutaan sovelluksen toimivan uudella alustalla. Tosin tässä

menetetään jonkin verran suorituskykyä verrattuna täysin natiiviin sovellukseen. Tällaisten sovellusten kehityksessä yksi ratkaisuista on PhoneGap. (6.)

PhoneGap on kehitysalusta mobiililaitteille, jonka ideana on voida kehittää sovelluksia riippumatta laitteen käyttöjärjestelmästä. PhoneGap antaa kehittäjälle mahdollisuuden kehittää sovelluksia käyttäen JavaScriptia, HTML5:tä ja CSS:ää sen sijaan, että tämä kehittäisi sovellusta suoraan tietyn käyttöjärjestelmän natiivikoodikielellä. (5.)

PhoneGap-kehitysalustaa ei vielä sovellettu riistakolmiolaskenta-sovellukseen. Scatman Oy on käyttänyt sitä aiemmissa sovelluksissaan, ja se on yksi vaihtoehdoista riistakolmiolaskennan jatkokehityksessä.

6 YHTEENVETO

Mobiililaitteet yleistyvät kaikenlaisessa toiminnassa. Tehtäviä sekä toimia, jotka on aikaisemmin hoidettu käyttäen paperia ja kynää, pyritään nykyään toteuttamaan erilaisilla sovellusratkaisuilla. Tällaisilla ratkaisuilla säästetään ensinnäkin valtavat määrät luonnonvaroja sekä nopeutetaan tiedon liikkumista. Scatman Oy:n toteuttama Scatman-järjestelmä kehitettiin ensisijaisesti öljyvahinkojen kartoitukseen ja puhdistustöiden organisointiin. Se nopeuttaa merkittävästi tiedon liikkumista tapahtumapaikalta komentokeskukseen.

Yrityksen johto oli suunnitellut toteuttaa tiedonkeruujärjestelmän riistaeläinten havainnointiin Scatman-järjestelmän pohjalta. Käytyämme läpi opinnäytetöiden vaihtoehtoja riistakolmiolaskenta-sovellus vaikutti itselleni henkilökohtaisesti mielenkiintoisimmalta. Olen pienestä asti harrastanut aktiivisesti metsästystä sekä luonnossa liikkumista. Käytössäni on ollut muutamia erilaisia karttasovelluksia metsästyksessä, joiden toimintojen tietämisestä oli paljon apua tätä sovellusta suunnitellessa.

Alkuperäisenä tavoitteena oli saada toteutettua toimiva versio järjestelmästä. Käytännön työn edetessä tuli todettua asetettu tavoite lähes mahdottomaksi, sillä uusien menetelmien opiskelussa meni ajateltua kauemmin aikaa. Aikaa kului myös paljon sivujen ulkonäön suunnittelussa ja toteutuksessa, sillä HTML sekä CSS olivat minulle ennestään tuntemattomia. Raportin kirjoittamisessa lähdinkin keskittymään kokonaisuuden esittämiseen lukijalle. Mikäli mahdollinen jatkokehittäjä yrityksessä tai sen ulkopuolella lukee raportin, on hänen helppo lähteä kehittämään sovellusta eteenpäin valmiin suunnitelman perusteella. Esimerkkinä keskeneräisyydestä näkyy sovelluksessa esitettävien riistakolmiolaskennan tietojen ja kenttien virheellisyys verrattuna paperiseen riistakolmiolaskenta-lomakkeeseen.

Jatkokehitysmahdollisuuksina sovellukselle olisi ensimmäisenä sovelluksen toteutuksen loppuun vienti. Myös mahdollisuus peto- ja hirvihavaintojen laskemiselle mainittakoon jatkokehitysmahdollisuutena. Scatman Oy on toteuttanut

aiemmin opinnäytetyönä karttatoiminnon, jonka käyttämistä tässä työssä voidaan myös harkita jatkokehityksessä (10). Riistakolmiolaskenta-sovellusta suunnitellessa tarkoituksena on koko ajan ollut, että se tullaan kääntämään tulevaisuudessa PhoneGap-sovellukseksi. Käytännössä kääntäminen olisi ollut sen verran suuri ja aikaa vievä työ, että se jätettiin pois tästä opinnäytetyöstä. Sen sijaan keskityttiin sovelluksen suunnitteluun ja back-end-puolen toteutuksiin.

Opinnäytetyö kokonaisuudessaan antoi hyvän kuvan ASP.NET MVC-tyyppisen sovelluksen suunnittelusta ja toteutuksesta. Koulussa oppimani tiedot ohjelmoinnista auttoivat paljon kokonaisuuden ymmärtämisessä, vaikka suurin osa käytetyistä menetelmistä olivat minulle täysin entuudestaan vieraita.

LÄHTEET

1. Riistakolmiot. 2014. Luonnonvarakeskus. Saatavissa: https://riistakolmiot.fi. Hakupäivä 25.9.2014.
2. Code first database. 2014. Microsoft. Saatavissa: http://msdn.microsoft.com/en-us/data/jj193542.aspx. Hakupäivä 8.12.2014.
3. MVC-toimintamalli. 2015. Wikipedia. Saatavissa: http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller#mediaviewer/File:MVC-Process.svg. Hakupäivä 9.12.2014.
4. Single-page application. 2015. Wikipedia. Saatavissa: http://en.wikipedia.org/wiki/Single-page_application. Hakupäivä 10.12.2014.
5. PhoneGap. 2014. Wikipedia. Saatavissa: http://en.wikipedia.org/wiki/PhoneGap. Hakupäivä 16.12.2014.
6. Toivonen, Jesse 2013. PhoneGap-työvälineohjelmistojen kehitys älypuhelimiin. Turku: Turun ammattikorkeakoulu, ohjelmistotuotannon koulutusohjelma. Opinnäytetyö. Saatavissa: https://www.theseus.fi/bitstream/handle/10024/62810/Toivonen_Jesse.pdf?sequence=1. Hakupäivä 16.12.2014.
7. Kesälaskentalomake. 2014. Luonnonvarakeskus. Saatavissa: http://riistakolmiot.fi/wp-content/uploads/2014/07/kesalaskenta_lomake_2014.doc. Hakupäivä 26.9.2014.
8. Talvilaskentalomake. 2014. Luonnonvarakeskus. Saatavissa: http://riistakolmiot.fi/wp-content/uploads/2014/07/talvilaskenta_lomake_2014.doc. Hakupäivä 26.9.2014.
9. Helle, Pekka - Wikman, Markus 2005. Riistakolmiot. Helsinki: Riista- ja kalatalouden tutkimuslaitos. Saatavissa http://www.rktl.fi/www/uploads/pdf/riistakolmiot. Hakupäivä 16.12.2014.
10. Kortet, Tarmo 2014. Kartta-aineistot Scatman web service -palveluun. Oulu: Oulun seudun ammattikorkeakoulu, tietotekniikan koulutusohjelma. Opinnäytetyö. Saatavissa: http://www.theseus.fi/bitstream/handle/10024/76174/Kortet_Tarmo.pdf?sequence=1. Hakupäivä 17.12.2014.

Liite 1 Kesälaskentalomake

RIISTAN- JA KALANTUTKIMUS

RIISTAKOLMION KESÄLASKENTA 2014

Palauta heti laskennan jälkeen

Palautettu riistakolmiot.fi sivustolla kyllä / ei

Riistakolmio nro

Sähköposti
Puhelin
GPS-paikannin? ☐ *Kyllä* ☐ *Ei*

Riistanhoitoyhdistys
Metsästysseura

Kolmion nimi

Muut laskijat _____

Laskentapäivä _____ / *2014 klo* _____ – _____ *lämpötila* _____ *°C*

Tuuli ***tyyni tai heikko / kohtalainen / kova***

Pilvisyys ***kirkas / puolipilvinen / pilvipouta / sadetta***

	HAVAINNOT	
	60 m:n pääkaistalta	pääkaistan ulkopuolelta
1.		
2.		
3.		
4.		
5.		
6.		
7.		
8.		
9.		
10.		
11.		
12.		
13.		
14.		
15.		
16.		
17.		
18.		

(7.)

Liite 2 Talvilaskentalomake

RIISTAN- JA KALANTUTKIMUS

RIISTAKOLMION TALVILASKENTA

Palauta heti laskennan jälkeen
Palautettu riistakolmiot.fi sivustolla kyllä/ei

Riistakolmio nro

Riistanhoitoyhdistys

Metsästysseura

Muut laskijat (lkm) _____

Edellinen lumisade _____ / _____

Esikierto _____ / _____ klo _____ - _____

Laskenta-aika _____ / _____ klo _____ - _____

Lumipeitteen paksuus kolmion kärjissä _____ ; _____ ; _____ cm

Lämpötila aamulla _____ °C klo _____ ja laskennan aikana _____ °C Tuuli tyyni / heikko / kohtalainen

Laskettavat lajit, **jäljet:** metsäjänis **ei** jänis, rusakko, orava, liito-orava, majava, piisami, susi, kettu, naali, supikoira, karhu, kärppä, lumikko, minkki, hilleri, näätä, ahma, mäyrä, saukko, ilves, villisika, valkohäntäpeura, hirvi, metsäpeura, metsäkauris

nähdyt: metso, teeri, pyy, riekko, peltopyy, fasaani, kanahaukka, korppi

	Laji ja lukumäärä		Laji ja lukumäärä		Laji ja lukumäärä
1.		21.		42	
2.		22.		43.	
3.		23.		44.	
4.		24.		45.	
5.		25		46.	
6.		26.		47.	
7.		27.		48.	
8.		28.		49.	
9.		29.		50.	
10.		30.		51	
11.		31.		52.	
12.		32.		53.	
13.		33.		54.	
14.		34.		55.	
15.		35.		56.	
16.		36.		57.	
17.		37.		58.	
18.		38.		59.	
19.		39.		60.	
20.		40.		61.	

(8.)

Liite 3 Tietokannan rakenne