TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutitehnika instituut

IAG40LT

Anett Kann 120903

PILVANDMETÖÖTLUSE RAKENDUSED

Bakalaureusetöö

Juhendaja: Vladimir Viies
PhD
Dotsent

Tallinn 2015

Autorideklaratsioon

Olen koostanud antud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud. Käesolevat tööd ei ole varem esitatud kaitsmisele kusagil mujal.

Autor: Anett Kann

24.05.2015

BAKALAUREUSETÖÖ ÜLESANNE

Üliõpilane: **Anett Kann** Matrikkel: **120903**

Lõputöö teema eesti keeles:
Pilvandmetöötluse rakendused

Lõputöö teema inglise keeles:
Cloud computing applications

Juhendaja (nimi, töökoht, teaduslik kraad, allkiri):
Vladimir Viies

Konsultandid: -

Lahendatavad küsimused ning lähtetingimused:
Anda ülevaade rakenduse koostamise vahenditest pilvandmetöötluses, lähtudes Eestist. Koostada juhis rakenduse loomiseks ja realiseerida sellest lähtuvalt vabalt valitud näidisrakendus.

Eritingimused: -

Nõuded vormistamisele: Vastavalt Arvutitehnika instituudis kehtivatele nõuetele

Lõputöö estamise tähtaeg: 08.06.2015

Ülesande vastu võtnud: _____ kuupäev: 24.05.2015
(lõpetaja allkiri)

Annotatsioon

Käesolev töö uurib pilvandmetöötluse mõistet Eesti näitel, tüüplahenduste raamistikke ning lihtsa avalikel teenustel põhineva rakenduse loomise ülesehitust ning protsessi, milles realiseeritakse päikesekalkulaator. Iseloodud projekt põhineb Google Maps API'l, mis võimaldab rakenduses kasutada hõlpsasti maailmakaarti ja OpenWeatherMap API'l, mis kuvab kaardile hetke ilmastikutingimused.

Töö tulemusena loodud päikesekalkulaatori põhjal analüüsitakse täpsemalt koodi ülesehitust, andmete hankimist, nende sidumist omavahel koodiga ning vastavust väljatoodud üldistele standarditele.

Töö esimene osa kirjeldab pilvandmetöötluse mõistet üldiselt, selle standardeid ning arengujärku Eestis. Teises osas on kirjeldatud täpsemalt .NET raamistikku ja selle tunnusjooni ning lühidamalt on kokku võetud teiste tüüpraamistike olemus. Kolmas osa hõlmab reaalset rakenduse loomise kirjeldust ning etappe ja teooriale toetudes loodud päikesekalkulaatori näidet.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 60 leheküljel, 3 peatükki, 12 joonist, 1 tabelit.

Abstract

Cloud computing applications

The goal is to analyze the process of developing a simple working cloud computing application based on the theory gathered in this thesis. The application was chosen so it would fill the conditions of being usable in Estonia and using public data, resulting in a sun calculator that displays the sunrise and sunset times on given day anywhere on the world map.

The theory used for the application consists of general information about cloud computing and it's standards set by National Institute of Standards and Technology, as well as an example of its development and uses in Estonia, and a short analysis of the chosen software framework .NET and some others that are suitable for developing a cloud computing application. The project is based on and gets its information and functional qualities from two separate API's. Google Maps API allows the application to use the world map with some necessary functionalities for the sun calculator, for example the coordinates of any place on Earth. OpenWeatherMap API adds the functionality of displaying weather of some larger cities on the map.

The thesis will also include the general steps of creating a cloud computing application, which will include setting the task the application will have to fulfill, choosing suitable tools for developing the application, using API's for simple access to public data and linking them with the program, writing code for any other wanted functionalities, and finally uploading the project to a cloud.

The resulting sun calculator application will be based on the same steps. The structure, data acquiring methods, linking data with the code and meeting the general cloud computing and framework standards of the sun calculator will be analyzed to give a better review of how to create a cloud computing application in the most simple and beneficial way.

The thesis is in Estonian and contains 60 pages of text, 3 chapters, 12 figures, 1 table.

Lühendite ja mõistete sõnastik

API	***Application Programming Interface*** Andmevahetuse rakendusliides programmeerimise hõlbustamiseks
CLR	***Common Language Runtime*** .NET platvormi käituskeskkond, täidab Microsofti vahekeelde MSIL tõlgitud programme
CLS	***Common Language Specification*** Valik keeli defineeritud selliselt, et ühes CLR keeles kirjutatud programm suhtleks teistes CLR keeltes kirjutatud programmidega
CTS	***Common Type System*** Defineerib selle, kuidas CLR'is tüüpe deklareerida, kasutada ja hallatada
D5	***Digital 5*** 5 juhtivat e-valitsemisega riiki koondav ühendus
GUI	***Graphic User Interface*** Graafiline kasutajaliides, mis kasutab arvuti graafikakuvamise võimalusi ja teeb programmide kasutamise lihtsamaks
HTTP	***The Hypertext Transfer Protocol*** Protokoll teabe edastamiseks arvutivõrkudes
IaaS	***Infrastructure as a Service*** Pilvandmetöötluse pakkumine infrastruktuurina
MVC	***Model-View-Controller*** Muster kasutajaliidese implementeerimiseks, mis jagab rakenduse kolme ossa - mudel, vaade ja kontroller

PaaS	***Platform as a Service*** Pilvandmetöötluse pakkumine andmetöötluse platvormina
SaaS	***Software as a Service*** Pilvandmetöötluse pakkumine tarkvara rentimisena
URL	***Uniform Resource Locator*** Internetiaadress mingi veebilehe poole pöördumiseks
WPF	***Windows Presentation Foundation*** Graafiline alamsüsteem kasutajaliideste formuleerimiseks Windows'i rakendustes
XML	***Extensible Markup Language*** Laiendatav märgistuskeel, mis on mõeldud andmete struktureerimiseks

Sisukord

Jooniste nimekiri

Tabelite nimekiri

Sissejuhatus

Pilvandmetöötlus on üpriski modernne, kuid kiiresti adapteeruv ja arenev suund infotehnoloogias, mis lihtsustab tunduvalt seniseid lahendusi. Üks suurimaid eeliseid pilvandmetöötluse rakenduste puhul on infotehnoloogia sektori töö lihtsustamine avaandmete pakkumisega.

Töö eesmärk on uurida pilvandmetöötluse rakenduste loomist ja nende ülesehitust, luues ise avalike teenuste põhjal rakendus. Loodud rakendust analüüsitakse ja võrreldakse juba olemasolevatega ning jälgitakse vastavust standarditele. Tulemus seatakse üles pilvlahendusena.

Töö on vajalik, kuna pidevalt areneva pilvandmetöötluse rakenduste suuna lahtimõtestamine on päevakohane ning praktika käigus loodud rakendus annab võimaluse selgeks teha, kuidas võimalikult efektiivselt arendada pilvepõhist tarkvara pilvandmeteenuste põhjal. Avalik teenus tähendab seda, et sellele pääsevad ligi kõik huvilised. Samal põhjusel on huvitundvatel arendajatel võimalus ise pilvandmetöötluse rakendus luua ja hiljem soovi korral avalikult üles riputada. Pilvandmetöötluse arenedes on uued abimaterjalid pidevalt vajalikud.

Antud töö on tarvilik eelkõige arendajatele, kes soovivad luua avaliku teenuse rakendusi, kuid vajavad selleks täiendavat informatsiooni. Töös on kokku kogutud vajalik teave iseseisvaks tüüplahenduse loomiseks.

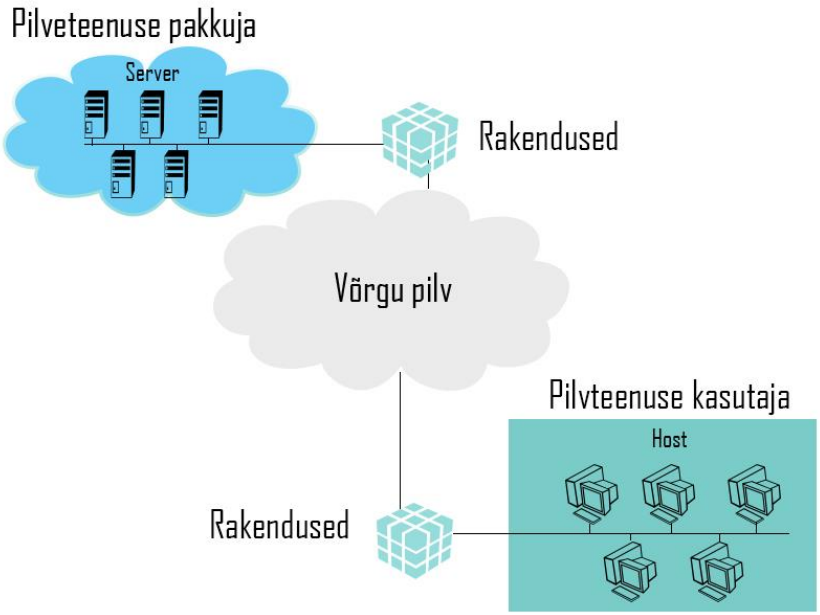
1. Pilvandmetöötlus Eestis

1.1. Pilvandmetöötlus

Pilvandmetöötlus põhineb pilves asuva andmestiku vahetamisel pilve andmekihi, rakenduskihi ja teenuste kihi vahel ning selle töötlemisel vastavates kihtides. Pilvandmetöötluse rakendus kui teenus on võrguplatvormi vahendusel lõppkasutajani jõudev rakendusprogramm.

Pilvteenuse pakkuja osutab teenust kasutajale, hoiustades võrgu pilves avalikke andmeid, millele vajadusel lihtsasti ligi saab. Sellised pilves paiknevad ja avalikult kasutatavad rakendused on näiteks aina enam populaarsust koguv tavakasutaja andmete hoiustamissüsteem Dropbox ja laialdaselt kasutatav maailmakaart Google Maps. Pilvandmetöötlusel põhinevad ka näiteks igapäevaselt kasutatavad e-maili teenused. Paljud inimesed puutuvad pilvega kokku sellest mitteteadlikuna.

Pilvandmetöötluse mudel on võrdlemisi lihtne. (Joonis 1) Pilvteenuse pakkuja võimaldab pilve üles seatud rakendustele ligi pääseda tavakasutajatel, aga ka näiteks arendajatel, kes saavad tänu pilve kaudu kättesaadavatele API'dele lihtsasti oma loodud pilvlahendustesse funktsionaalsusi lisada. Lõpptulemusena valminud rakendus sõltub otseselt võrgu pilvest ning selles paiknevatest teenustest ning jõuab lõppkasutajani taaskord pilve kaudu.



Joonis 1. Pilvandmetöötluse üldine mudel.

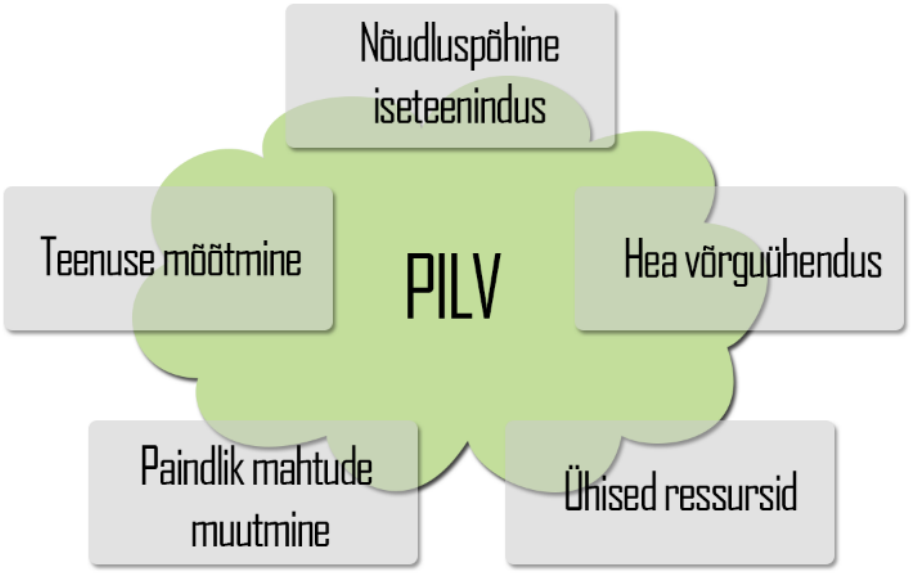
Pilvandmetöötlus võimaldab teenuse kasutajal tarvitada ressursse, mis ei paikne tema isiklikus arvutis; see on ühtlasi üks suuremaid pilvandmetöötluse eeliseid. Selline eriarvutite vahel jaotatud andmete paiknemine parandab suuresti jõudlust ning säästlikkust ning tarbija ei pea isegi teadma, kus üks või teine tema kasutatud teenus tegelikkuses paikneb. Suurimaks nõrkuseks võib pidada turvalisuse küsimust, sest avalike teenuste kaudu on lihtsam laialdaselt näiteks pahavara levitada või serverite massirünnakuga suuremat kahju tekitada, samuti võib tekkida risk andmete kaole.

1.2. Pilvandmetöötluse standardmudel

Pilvandmetöötlus on alles noor ning pidevas arengujärgus, mille tõttu selle karakteristikud arenevad ja muutuvad aja vältel. Standardid on välja töötatud ühtlustamiseks ja ühtluse säilitamiseks erinevate lahenduste puhul üle maailma, et vältida potentsiaalseid ebakõlasid. Sama meetodit on rakendatud ka pilvandmetöötlusele ning seda viimaste aastatel üha rohkem.

National Institute of Standards and Technology (NIST) 2011.aastal väljastatud publikatsioonis on standardne pilve mudel võetud kokku järgnevalt:

1) Vajalikud karakteristikud



Joonis 2. Pilve karakteristikud.

- Nõudluspõhine iseteenindus. Tarbijal on võimalik pilvandmetöötluse võimalusi ära kasutada iseseisvalt, teenusepakkuja abita.
- Hea võrguühendus. Võimalused on saadaval üle terve võrgu ning neile pääseb ligi standardsete mehhanismidega, et edendada kliendiplatvormide kasutamist.
- Ühised ressursid. Teenusepakkuja töödeldavad ressursid on üles seatud nii, et need oleksid võimelised teenindama mitut klienti korraga erinevate füüsiliste ja virtuaalsete vahenditega, mis jaotatakse vastavalt klientide nõuetele. Klient ei tea ega kontrolli seda, kus talle pakutud ressursid (näiteks mälu, virtuaalmasinad ja andmete säilitamise võimalused) paiknevad.
- Paindlik mahtude muutmine. Töövõimet võib üles skaleerimise eesmärgil kiirelt ja elastselt (upscaling) jaotada või vabastada, et kiiresti alla skaleerida (downscaling). Tarbijale on töövõime sageli piiramatu ning mahte saab juurde osta mistahes koguses igal ajal.
- Teenuse mõõtmine. Pilvesisene süsteemihaldus juhib ja optimeerib ressursside kasutamist. Ressursikasutust saab jälgida, selle õigsust kontrollida ning ette kanda, pakkudes sellega läbipaistvust süsteemi töö kohta nii teenuse osutajale kui ka tarbijale. [1] (Joonis 2)

2) Teenusemudelid (Tabel 1)
   - Tarkvara teenusena (SaaS). Tarbijale pakutakse võimalust kasutada teenusepakkuja rakendusi, mis jooksevad pilve infrastruktuuril. Rakendused on kättesaadavad erinevatest kliendi seadmetest kasutajaliidese kaudu, näiteks veebibrauseri vahendusel. Klient ei halda ega kontrolli pilvetaristut, sealhulgas ei võrku, servereid, operatsioonisüsteeme ega isegi individuaalseid rakenduse võimalusi, välja arvatud limiteeritud konfiguratsiooni sätted. [1] Pakutav teenus ei asu kliendi juures, vaid seda hoiustatakse pilves ning teenuse ning selle kättesaadavuse eest vastutajaks on teenuse pakkuja. SaaS võimaldab kasutajal mingit olemasolevat lõplikku rakendust kasutada, näiteks e-maili teenused.
   - Platvorm teenusena (PaaS). See on lahendus, mida tarbijale pakutakse. PaaS lubab pilvetaristusse paigaldada tarbija poolt loodud või omastatud

rakendusi, kasutades programmeerimiskeeli ja tark- ning riistvara, mida toetab pakkuja. Tarbija ei halda ega kontrolli pilve infrastruktuuri, kaasa arvatud võrku, servereid, operatsioonisüsteeme või andmete säilitamist, kuid saab rakendusi ja limiteeritult nende raamistikke muuta. [1] Kliendile jääb ülesanne kas leida või luua rakendus, mida mingis sobivas raamistikus jooksutada. PaaS tüüpi teenus on näiteks Microsoft Azure, mis toetab paljusid eriraamistikke ning võimaldab kasutajatel hõlpsasti veebilehti ja rakendusi hostida.

- Infrastruktuur teenusena (IaaS). Tarbijale pakutav lahendus sätestab andmete töötlemist, säilitamist, võrke ja teisi fundamentaalseid pilvtöötluse ressursse, kus tarbijal on võimalik kasutada ja käivitada suvalist tarkvara, kaasa arvatud operatsioonisüsteeme ja rakendusi. Tarbija ei halda ega kontrolli pilve infrastruktuuri, kuid omab kontrolli operatsioonisüsteemide, andmete säilitamise, rakenduste ja mõnede limiteeritud võrgukomponentide (näiteks tulemüür) üle. [1] Klient valib serverite poolt jooksutatud virtuaalsetest masinatest sobiliku, milles saab luua oma töökeskkonna. Näiteks võib samuti tuua Microsoft Azure'i, mis pakub näiteks riistvara tellimise tuge.

Mudel	Kasutajad	Pakutavad teenused	Eesmärk
SaaS	Lõppkasutajad	Valmisrakendused, näiteks Emaili teenused, Google Maps, CRM…	Kasutada teenuseid
PaaS	Arendajad ja paigaldajad	Teenuste ja rakenduste testimine, arendus, integratsioon ja paigaldamine. DB, Web Server…	Pakkuda ja paigaldada lõppkasutajale teenuseid
IaaS	Süsteemihaldurid	Virtuaalmasinad, operatsioonisüsteemid, CPU, mälu, tagavarateenused, andmete hoiustamine.	Pakkuda testimiseks, arendamiseks, integratsiooniks ja paigaldamiseks sobilikku keskkonda

Tabel 1. SaaS, PaaS, IaaS mudelid. [7]

3) Kasutuselevõtumudelid

- Privaatpilv. Pilve infrastruktuur on ettevõttesisene või kliendipõhine. Seda saab hallata vastav organisatsioon või kolmas osapool ning see võib eksisteerida nii kohapeal kui eemal. [1] Privaatpilv on disainitud sarnaselt avalikule pilvele, kuid on tunduvalt turvalisem, kuna on üles seatud kindlas ettevõttes ning võimaldab seega paremat kontrolli ettevõtte andmete üle. Samas on privaatpilv kulukam, kuna ettevõte peab kulud iseseisvalt katma.
- Kommuunipilv. Pilv on jagatud mitme organisatsiooni vahel ning toetab kindlat kogukonda ühiste vaatenurkadega. Seda võivad hallata kõik ühinenud organisatsioonid või kolmas osapool ning see võib eksisteerida nii kohapeal kui eemal. [1] Seetõttu on paremini jaotatud ka kulud ning tegu on tunduvalt säästlikuma mudeliga kui privaatpilv.
- Avalik pilv. Pilve infrastruktuur on kättesaadav avalikkusele või mingile suurele tööstusharu grupile ning seda omab organisatsioon, mis müüb pilvteenuseid. [1] Põhimõtteliselt on tegu pilvega, mis on suunatud kõigile kasutajatele piiranguteta. Kasutajad toetuvad avalikult pakutavatele teenustele, mis muudab selle mudeli kasutamise väga säästlikuks ning kiireks, kuna suuremad kulud kaetakse teenusepakkuja poolt.
- Hübriidpilv. Pilve infrastruktuur on eelnevalt mainitud pilvetüüpide kombinatsioon, mis jäävad eraldi seisma, kuid on ühendatud tehnoloogiaga, mis võimaldab andmete ja rakenduste teisaldatavust. [1] Võib öelda, et hübriidpilv ühendab teiste kasutuselevõtumudelite parimaid omadusi ning kuna avalikud ja privaatsed andmed on kombineeritud, kuid säilitavad sõltumatuse, on kasutajal võimalik oma projektidesse rakendada avalike teenuste andmeid, samal ajal, kui tähtsamad või isiklikud andmed saab paigutada oma privaatpilve. (Joonis 3)

Joonis 3. Kasutuselevõtumudelid.

Sellisele NIST'i poolt määratud üldisele globaalstandardile võiks vastata iga pilv. Siiski, kuna tegu on globaalse standardiga, siis olenevalt riigist ja piirkonnast, kus pilv asub, võivad juurde tulla mitmesugused piirangud või nõudmised, mida jälgima peab.

1.3. Pilvandmetöötluse areng Eestis

Nagu Euroopa pilvandmetöötluse arendamise juht ja Eesti president Toomas Hendrik Ilves möödunud aastal väitis, siis pilvandmetöötluse potentsiaali märgatakse enamasti väljaspool Euroopat, enamasti Ameerika Ühendriikides. Kuna firmad võistlevad omavahel pidevalt, tuleks Euroopal rohkem sammu pidada. [2] Eesti on seda kohustust täitnud, võimaldades Eesti kodanikele selliseid teenuseid nagu näiteks elektrooniline ID kaart, millega on võimalik dokumente allkirjastada, rääkimata mobiil-ID võimalustest. Taoliste teenuste võimaldamine rahvale sillutab tee pilvraalinduse edasiarenguks.

Lisaks paljudele riigiga seotud teenustele, nagu näiteks e-valimised, tuludeklaratsiooni esitamine, isikliku firma loomine ja palju muud, on hiljuti plaani võetud valitsus turvalisuse huvides pilvele üle viia, ehk luua virtuaalsed „andmesaatkonnad". Eelkõige on see vajalik riigi andmete hoiustamise turvalisuse tõstmiseks, kuna riik on kaitsetu välisrünnakute vastu, nagu näitas ka 2007. aasta massirünnak veebilehtede vastu. [2] Samuti on selliste andmete hoiustamine hea viis vältida andmekadu looduslike katastroofide korral, nagu 2011. aastal Fukushima katastroof Jaapanis kaasa tõi. [3]

Valitsuse pilvele üleviimisel tuleb jälgida, et ei minda vastuollu mõningate Eesti seaduste ja sätetega. Näiteks on Eestis seadus, mis nõuab, et valitsus ei tohi küsida kodanikult andmeid, mis juba olemas on ja kodanik on iseenda andmete omanik ning kontrollib seda, kes ja miks tema andmetele ligi pääseb. [2] Otseselt Eesti seadustes taolise riiki mõjutava muutuse läbi viimiseks reforme tegema ei pea, kuid sarnaseid reegleid vastava seadusega tuleb jälgida ning plaan tuleb mahutada globaalsetesse standarditesse nii, et lõpptulemus oleks kooskõlas kõigi seotud riikide seadustega.

Suurim küsimus plaani läbiviimisel on turvalisus ja usaldus. Tuleb olla kindel, et see, kes süsteemile ligipääsu taotleb, on ka tegelikult see, kelleks ta väidab end olevat. Muidu on kõik süsteemid lihtsasti haavatavad. Sama oluline on läbipaistvus. Näiteks Eestis on see lahendatud nii, et ID kaarti on võimalik kasutada eesmärgil, et näha, kes on sooritanud kodaniku isiklike andmete päringuid ja mis põhjusel. Samal ajal on süsteemid tugevalt krüpteeritud ning on peaaegu võimatu neist läbi murda. [2] Selline turvalisuse, usalduse ja läbipaistvuse kombinatsioon on vajalik, et ka kodanike ja tavakasutajate usaldus pilve toimimisse kasvaks.

Eesti on ühtlasi viit kõige enam arenenud e-valitsemisega riiki koondava ühenduse D5 asutajaliige. Peale Eesti kuuluvad sinna veel Suurbritannia, Lõuna-Korea, Iisrael ja Uus-Meremaa. [4] Ühenduse esimene kokkusaamine toimus 2014.aasta detsembris ning üheks arutatavaks teemaks oli avalik turg, mille raames arutleti ka Suurbritannias asuva G-Cloud - raamistiku üle, mis on tekitanud Suurbritanniasse avatud, läbipaistva ja konkurentsivõimelise turu. [5] G-Cloud käitub kui pilvevahendaja, kus pilvteenuse pakkujad saavad oma teenuseid pakkuda ja andmeid hoiustada. Tarbijad saavad lihtsasti neid teenuseid otsida ja võrrelda nii, et protsess vastaks täielikult Suurbritannia ja Euroopa standarditele. [6] Tänu D5 olemasolule on võimalik usaldust üles ehitada teiste liikmesriikidega, kes on potentsiaalsed andmete hoiustajad, kui lähebki riigi andmete pilve üleviimiseks.

Üldiselt on Eestis palju avalikke teenuseid, mis on kodanikele pilve kaudu kättesaadavad ning tulevikuplaanid vaid edendavad seda. Palju funktsionaalsusi on juba võrgus teenustena üleval, alates televisioonist lõpetades hariduse ja riigiteenustega. Riigi pilvele üle viimine oleks vaid järjekordne samm Eesti pilve arendamises.

2. Tüüplahenduste raamistike analüüs

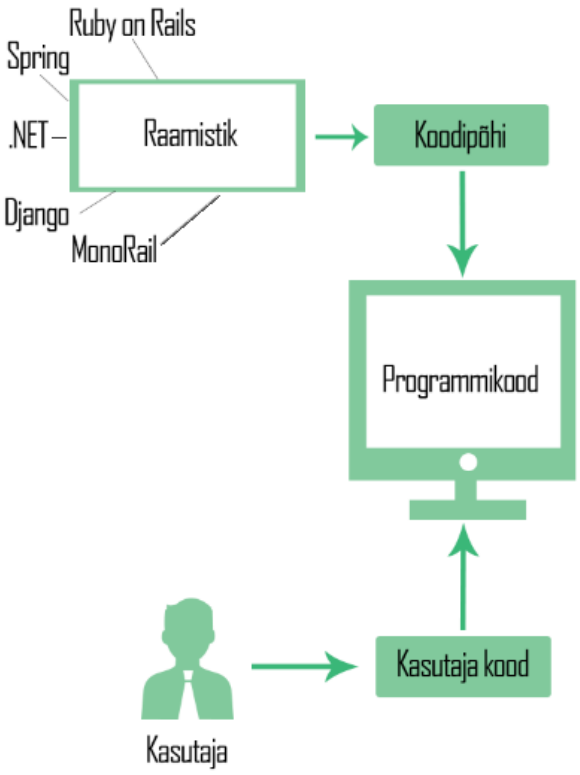
2.1. Tüüplahenduse raamistik

Tüüplahendus on mistahes rakendus, mis saab oma andmeid pilvlahendusest, töötleb neid seal või lükkab töödeldud andmeid pilvlahendusse. Tarkvara raamistik on platvorm tarkvararakenduste hõlpsamaks arendamiseks. Tüüplahenduse raamistik annab koheselt kasutatavad (out of the box) võimalused, mis teevad võimalikuks importida vajalikud töötavad teegid pilvlahenduse loomiseks, jättes vaid funktsionaalsuse lisamise mure.

Koodi kirjutamist lihtsustab raamistikuga loodud vundament, mille vahenditele toetudes saab paindlikult ja vähem veaohtlikult programmeerida kvaliteetsemaid, usaldusväärsemaid ja korrektsema ülesehitusega rakendusi spetsiifilise platvormi jaoks. Näiteks võivad raamistikus olemas olla klassid ja funktsioonid, millega saab töödelda sisendit, hallata kasutatavat riistvara ja suhelda süsteemi tarkvaraga. Tänu sellele ei pea programmeerijad iga uue rakenduse loomisel juba varem läbi tehtud ja olemasolevat lahendust uuesti välja nuputama.

Sarnasusi saab välja tuua rakendusliidesega API, kuid neil on oluline erinevus. Nagu nimigi ütleb, on raamistik programmeerimise vundament, samal ajal kui API võimaldab ligipääsu raamistiku poolt toetatud elementidele. [8] Põhimõtteliselt on raamistik programmikoodi põhi, mille funktsionaalsust rakendab kasutaja oma vajaduste järgi. (Joonis 4)

Raamistikke kasutades implementeerib kasutaja programmile omased objektid ja meetodid ning need algväärtustatakse ja kutsutakse välja raamistiku poolt. Raamistik defineerib programmivoo kontrolli. Kasutaja võib valikuliselt raamistiku poolt pakutud vaikimisi koodi asendada oma kirjutatud koodiga. Samas raamistiku enda koodi modifitseerida ei saa [9]. Vastasel juhul kaotaks raamistik oma mõtte.



Joonis 4. Tarkvara raamistiku üldine mudel

Põhilised eelised raamistike kasutamise juures on esiteks eelnevalt kompileeritud ja testitud koodi taaskasutamine, mis tõstab uue rakenduse usaldusväärsust ning vähendab ajakulu; teiseks võimaldavad raamistikud efektiivsemat programmeerimise praktiseerimist ja asjakohast disaini, mis on pidevalt ajakohane; kolmandaks, raamistik võimaldab kasutajal vajadusel funktsionaalsust laiendada. [9] Paljud raamistikud on paindlikud programmeerimiskeelte suhtes, võimaldades mitmel erineva programmeerimiskeele eelistusega inimesel korraga ühe projekti kallal töötada.

Miinusteks võib lugeda seda, et raamistike loomine on küllaltki keeruline ning aeganõudev, seega ka kulukas. Samuti on uue raamistiku ära õppimine pühendumust nõudev. Puuduseks võib lugeda ka selle, et raamistikud lisavad koodile pikkust. Kui raamistikke valesti kasutada, võib tekkida jõudluse langus, kuna osa kasutatud koodist on üleliigne. [10] Näiteks võib juhtuda, et API funktsioone kutsutakse välja vajaduseta.

Raamistikke on tänu kasvavale nõudlusele palju ja erinevaid, kuid pilvandmetöötluse tüüplahenduste loomisel üks populaarsemaid on .NET raamistik. See selgub ka Eesti infotehnoloogia üliõpilaste seas läbi viidud küsitluse tulemustest (Lisa 1). Järgnevalt

lahatakse pikemalt .NET raamistikku, mille alusel luuakse hiljem pilvandmetöötluse rakendus ning antakse lühike ülevaade ka teistest populaarsetest platvormidest mõnede tuntumate keelte kaupa (C#, Java, PHP, Ruby, Python).

2.2. .NET

Tüüplahendustele sobilikest raamistikest üks tuntumaid ja enam levinuid on Windows'i põhine .NET raamistik. See on mitmekeelne rakenduskäivituskeskkond, mis haldab läbipaistvalt taristuteenuste tuumikut. [13] .NET tehnoloogia toetab uue generatsiooni rakenduste ning veebiteenuste kompileerimist ja jooksutamist.

Visual Studio .NET raamistiku üldine ülesehitus ei ole keeruline, kuid sisaldab palju funktsionaalsusi. (Joonis 5) Raamistik on disainitud täitmaks järgmisi eesmärke:

- Objekt-orienteeritud programmeerimise keskkonna pakkumine olenemata sellest, kus objektikood talletatud on ja kuidas programmikoodi täidetakse
- Programmikoodi täitmise keskkonna pakkumine, mis minimaliseerib tarkvara ja selle uuenduste installeerimise vajadust ja võimalikke eriversioonide vahel tekkivaid konflikte
- Programmikoodi täitmise keskkonna pakkumine, mis võimaldab koodi ohutult täita; kaasa arvatud sellist koodi, mis on loodud mõne tundmatu või vähem usaldusväärse osapoole poolt
- Programmikoodi täitmise keskkonna pakkumine, mis elimineerib skriptitud või interpreteeritud keskkondade jõudlusprobleemid
- Arendaja kogemuse muutmine järjepidevaks ja mugavaks laialdaselt varieeruvate rakenduste puhul, näiteks Windows'i ning veebipõhised rakendused
- Saavutada eesmärk, et .NET raamistikul põhinev kood võib integreeruda mistahes muu koodiga [11]

.NET raamistik sisaldab endas CLR'i, mis on ühtlasi selle tähtsaim osa, ning .NET raamistiku klassiteeki. CLR on antud raamistiku aluspõhi. CLR haldab koodi selle täitmise ajal, pakkudes selliseid teenuseid nagu mäluhaldus, programmiosade haldus ja kaugligipääs. See raamistiku osa pakub palju eeliseid, edendades näiteks suurel osal turvalisust. Erinevatel hallatavatel osadel on erinevad usaldustasemed, mis sõltuvad erinevatest asjadest, eeskätt komponentide päritolust. Usaldustase määrab ära, milliseid

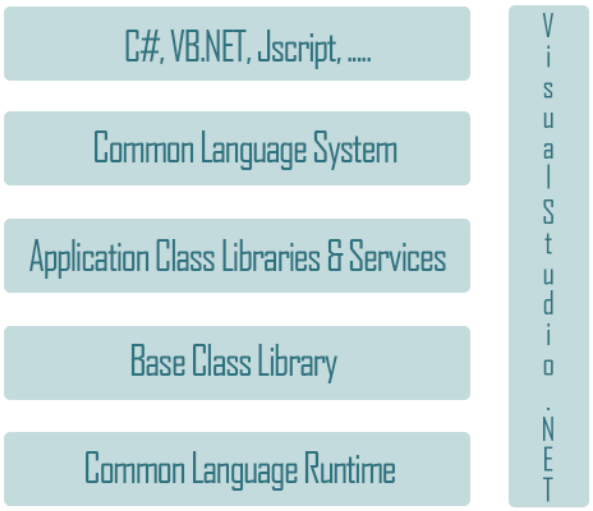
tegevusi on komponendil lubatud teha ja mis juurdepääsud neil olemas on. [13] Tänu turvalisuse suurendamisele saavad kasutajad kindlad olla, et mingil veebilehel käitatav programm võib ekraanil näidata mingit animatsiooni, mängida muusikat või rakendada muud funktsionaalsust, kuid ei pääse samal ajal ligi nende isiklikele andmetele.

CLR edendab ka arendajate produktiivsust, andes arendajatele võimaluse kirjutada rakendusi enda valitud keeles nii, et saadakse kasu kõigist CLR pakutud funktsionaalsustest. Lisaks rakendab CLR töökindluse tõstmiseks CTS'i, tehes kindlaks, et kõik keeled jagavad samu andmetüüpe. [13] CTS defineerib iga andmetüübi eraldi klassina ning iga .NET'iga ühilduv keel peab selle definitsiooni juurde jääma. [14] Kokkuvõtvalt ongi CLR disainitud selleks, et jõudlust parendada.

Klassiteek on terviklik, objekt-orienteeritud kogum korduvkasutatavatest tüüpidest, mida kasutades saab arendada rakendusi alates tavapärastest GUI rakendustest kuni veebivormideni ja XML veebiteenusteni. Klassiteek on tihedalt integreerunud CLR'iga. [11] Microsoft .NET'is on kõik klassitüübid grupeeritud nimeruumidesse. [13] Klassiteek muudab .NET raamistiku liigendatuks ning lihtsasti kasutatavaks vähendades niiviisi aega, mis kulub raamistiku ära õppimiseks.

Üks ilmne teema .NET'i puhul on erinevate programmeerimiskeelte ühendatus ja koostalitlus. Selle saavutamiseks tuleb luua teatud normid. CLS on kogum reegleid ja piiranguid, mida iga .NET'i ühilduvusega keel peab jälgima. Microsoft vaatleb CLS'iga määratud .NET ühilduvust kolmel tasemel:

1) Ühilduv produtseerija - komponenti, mis on arendatud seda tüüpi keeles, võib kasutada ka iga teine keel
2) Tarbija - selle kategooria keel võib kasutada klasse, mis on loodud ükskõik, mis muus keeles
3) Laiendaja - selle kategooria keeled võivad kasutada 'tarbija' taseme klasse ning võivad ka klasse laiendada, kasutades pärilust (ehk uute klasside tuletamine juba olemasolevatest). [14]



Joonis 5. Visual Studio .NET ülesehitus

.NET raamistikku saab kasutada, et arendada näiteks konsoolirakendusi, GUI rakendusi, WPF rakendusi, ASP.NET rakendusi, Windows'i teenuseid ja mitmesuguseid muud liiki projekte. Igal eritüüpi rakendusel on omad eelised, vastavalt sellele, mida lõpptulemusena saavutada tahetakse.

Edaspidi luuakse töös ASP.NET rakendus. ASP.NET on veebivormide- ja veebiteenuste loomise tehnoloogia .NET raamistikus ning võimaldab arendajal täiendada veebilehe andmeid lehe täieliku taaslaadimiseta. [12] Veebivormid võimaldavad kiiresti luua vormidel põhinevaid veebilehti. Veebiteenused lubavad kasutada andmevahetust, kasutades selliseid standardeid nagu HTTP ja XML sõnumite saatmine, et liigutada andmeid tulemüüridest läbi. [13] ASP.NET töötab HTTP protokollil ja kasutab HTTP käsklusi. ASP.NET rakendused on kompileeritud koodid, mis on kirjutatud konponentidest või objektidest, mis on olemas .NET raamistikus ning saavad kasutada klassihierarhiat. [12] ASP.NET rakendusi võib kirjutada näiteks C#, Visual Basic.NET, Javascript keeltes, olenevalt arendaja eelistustest. ASP.NET'i rakendustel on lihtsustamiseks olemas erinevad kontrollielemendid nagu näiteks nupud, sildid ja tekstikastid.

Selleks, et .NET raamistikus valminud programm oleks lihtsasti mõistetav ja loetav, on Microsoft loonud juhendi „Framework Design Guidelines" raamistiku disaini standarditest. [15] Eeldatavalt võiks iga .NET programmeerija koodi kirjutades neid juhtnööre jälgida, et programm oleks üheselt mõistetav ning lihtsasti loetav kõigile arendajatele. Antud standardeid jälgitakse ka edaspidi loodavas koodis.

.NET'i tähtsaimad eelised on keeltest sõltumatus - ettevõttes võivad töötada erinevate programmeerimiskeelte eelistustega inimesed sama projekti raames nii, et erikeelte eelistus ette ei jää; objekt-orienteeritus; standartiseeritus; paindlikkus; suur funktsionaalsus; laialdane kasutamine ning seega hea dokumenteerimine (tüüpvead on teada) ning toetav kogukond; ning kindlasti lihtsus raamistiku kasutamisel (Lisa 1). Kindlasti on võimalik saada pidevat uut tehnoloogiat Microsoftilt rakenduste loomiseks ning tähtis on laialdane programmeerimiskeelte valik, mille hulgas on nii dünaamilisi kui staatilisi, objekt-orienteeritud kui funktsionaalseid keeli.

Suurimaks puuduseks on platvormist sõltumine, sest raamistik on vaid Windows'i põhine. Raamistik on juba lõplik ja seetõttu muutumatu. Üliõpilaste poolt on veel mainitud, et koodi on lihtne pöördprojekteerida ning keeruline ära õppida (Lisa 1).

2.3. Veebiarenduskeelte raamistikud

Tüüplahendustele on lisaks populaarsele .NET raamistikule loodud mitmeid teisi tarkvararaamistikke. Järgnevad raamistikud on valitud nii, et oleks esindatud arenduskeskkonnad potentsiaalsetele veebiarenduskeeltele C#, Java, Python, PHP ja Ruby.

MonoRail on täielikult MVC'ga ühilduv veebiraamistik, mis võimaldab arendust C# keeles. Eelisteks on see, et projekti struktuur on sama kõikide MonoRail'i lahenduste puhul; kontrollereid ja nende sõltuvusi ning parameetreid saab konteinerist vajadusel juurde lisada; kontrollereid saab hõlpsasti tänu raamistikus sisalduvatele vahenditele testida. MonoRail ei ole hea valik, kui projekt sõltub liiga palju kolmandast osapoolest või kui programmeerimisoskused piirduvad vaid veebivormidega. [16] MonoRail on mõnes mõttes kui lihtsustatud versioon .NET raamistikust, eriti on tagasi tõmmatud koodi massiivsuse poole pealt.

Orleans on raamistik, mis võimaldab lihtsasti programmeerida suure ulatusega andmetöötluse rakendusi. See on loodud Microsoft Research'i poolt ja disainitud pilvandmetöötluses kasutamiseks. [17] Antud raamistik pole väga laialdaselt kasutatav, kuid seda on ulatuslikult kasutatud Microsoft Azure'is. Programmeeritavaks keeleks on C#.

CakePHP on vaba lähtekoodiga veebirakenduste loomise raamistik ning ühildub MVC'ga. Raamistik sisaldab endas Ruby on Rails raamistikust inspireeritult teeke, klasse ja käitusaegset infrastruktuuri. CakePHP tugevateks eelisteks on aktiivne ja sõbralik kogukond, paindlikkus, MVC arhitektuur, ühilduvus PHP4 ja PHP5'ga, sisse ehitatud valideerimine ja palju muud. [18] Nagu nimigi ütleb, on tegu rangelt PHP keelele suunitletud raamistikuga.

CodeIgniter on veebirakenduste loomise vahend arendajatele. Raamistik sisaldab endas mahukalt teeke programmeerimise põhitegevuste jaoks, samuti lihtsat liidest ja loogilist struktuuri teekidele ligipääsemiseks. Kasutab MVC platvormi ning omab aktiivset kogukonda ning põhjalikku dokumentatsiooni. Mahult väike, lihtsasti õpitav ja jõudluselt väga kiire ning kindlasti tugev konkurent teistele raamistikele. [19] Antud raamistik on samuti orienteeritud PHP keeles arendamisele.

Spring raamistik on platvorm, mis pakub lihtsat keskkonda Java rakenduste arendamiseks. Spring tegeleb infrastruktuuriga, samal ajal kui arendaja saab keskenduda oma koodile. See raamistik on arendajale kasulik, kuna ei pea tegelema erinevate API'dega, seda teeb platvorm arendaja eest. Raamistik on väga mahukas ning seega võib tekkida probleeme dokumentatsiooniga. [20] Lisaks on raamistikul olemas head lahendused ettevõtetele, seda on pigem lihtne ära õppida ning tal on suur funktsionaalsus (Lisa 1).

Athena raamistik on vaba lähtekoodiga raamistik Java rakenduste loomiseks. Raamistik lihtsustab Java veebirakenduste arendamist, eemaldades käsitsi kaardistamise ja andmebaasi uuendamise nõude. Athena toetab teenusepakkumist serveri poolt mitmele kliendile, tänu millele on arendajatel lihtne luua pilverakendusi. Raamistiku eeliseks on suur vastuvõtlikkus muutustele. [21]

Django on Pythoni rakenduste loomisele spetsialiseerunud raamistik, mille põhieesmärk on aidata arendajatel rakendusi viia ideest realisatsioonini nii kiiresti kui võimalik. Platvorm on vabavarana saadaval ning vaba lähtekoodiga. Raamistik on võimeline vastu pidama ka arvukale liikumisele veebilehtedel ning on väga paindlik, võimaldades programmeerida väga erineva nõudlusega rakendusi. [22] Eelisteks võib lugeda seda, et lihtne on luua lihtsaid rakendusi ning raamistikku on lihtne ära õppida, kuna olemas on

hea dokumentatsioon ja õpetused. Samuti on raamistik mugav kiireks prototüüpimiseks. (Lisa 1)

OpenStack on vaba lähtekoodiga raamistik, mis on loodud spetsiaalselt pilvandmetöötluse jaoks, kusjuures baaskoodid on pärit Rackspace'ilt ja NASA'lt. See on lihtne ning suuresti skaleeritav platvorm ning võimaldab luua oma pilve, milles rakendusi arendada. Tähtsal kohal on sõna „avatus", mis tähendab limiitideta tarkvara, üsnagi avatud disaini võimalusi, lähtekoodide avalikku kättesaadavust ning avatud kogukonda. [23]

Ruby on Rails, vahel ka lihtsalt Rails, on Ruby keelt laiendav raamistik veebilehtede ja rakenduste loomiseks. Rails raamistiku API on laialdaselt dokumenteeritud ning hõlpsasti kättesaadav. Raamistik kombineerib veebiserveril jooksva rakenduse arendamiseks Ruby keele HTML, CSS ja JavaScriptiga. [24] Tegu on küpse ja täielikult välja arendatud raamistikuga. Kuigi Ruby keelt on peetud keeruliselt õpitavaks ja vähepopulaarseks, on antud raamistiku nimetus tuttav ka nendele, kes otseselt sellega kokku pole puutunud. Raamistikus on mugav kirjutada. (Lisa 1)

Ramaze on väga lihtne ning otsekohese põhimõttega veebiraamistik, toetudes põhimõttele, et lihtsad asjad peavad jääma lihtsaks ning keerukad asjad peavad olema võimalikud. Tähtis on ka see, et iga osa oleks võimalikult modulaarne ja taaskasutatav, et tagada lihtsust. Projektid põhinevad MVC'l, kuid arendaja ei ole sunnitud seda kasutama. Tegu on üsna paindliku raamistikuga. [25]

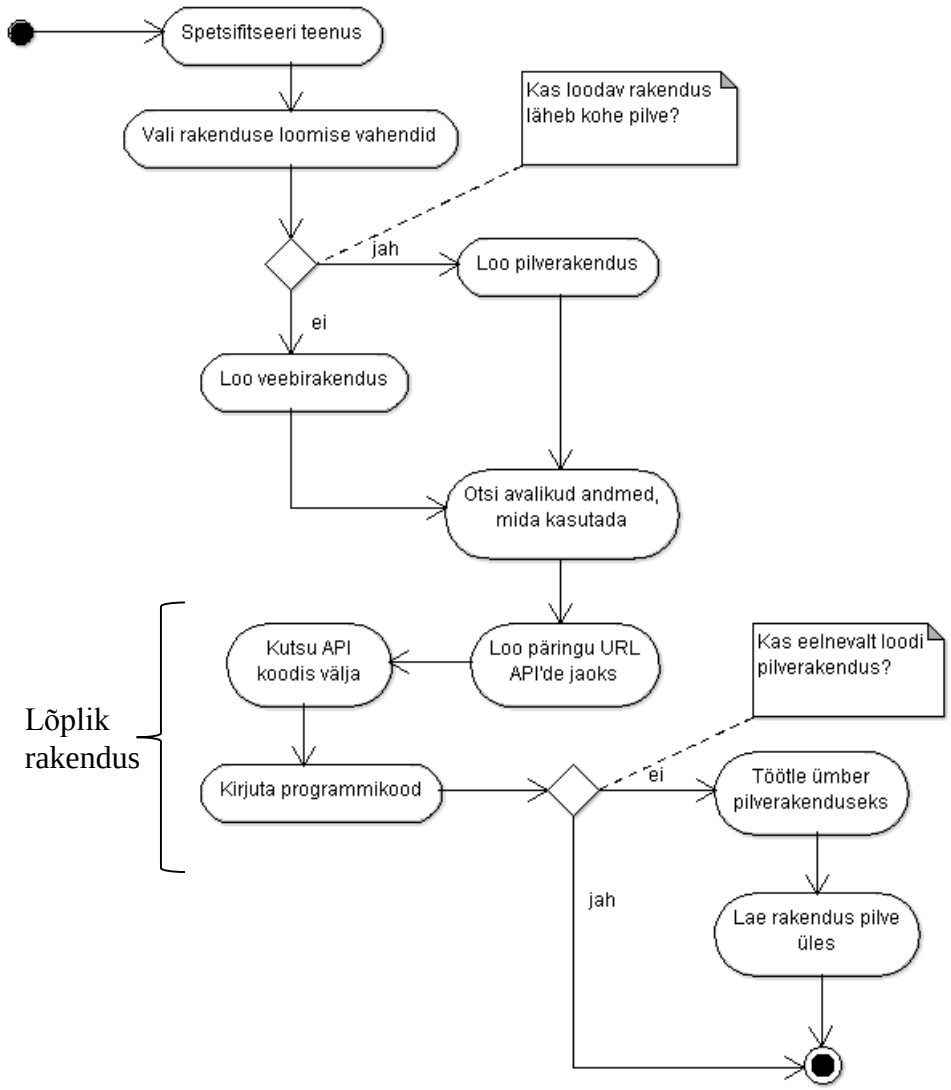
Nendest raamistikest tuntumaks osutusid Ruby on Rails, Django ja Spring ning need olid ühtlasi raamistikud, millega oli Eesti üliõpilastel isiklikke kogemusi. Kõige vähem tuntud olid Ramaze ja Orleans, millest ei olnud kuulnud ükski vastanutest ning vähe teati ka OpenStack'i, CodeIgniter'it ja Athenat. (Lisa 1)

Kokkuvõtvalt võib öelda, et kuigi on loodud üsna mitmeid raamistikke, on nende suurim põhimõtteline erinevus programmeerimiskeelte toetus ning see, mis operatsioonisüsteemidele need limiteeritud on. Üldiselt on siiski iga raamistiku põhimõte üks ja sama - lihtsuse, kiiruse ja mugavuse tagamine arendajale.

3. Pilvandmetöötluse rakenduse koostamine

3.1. Pilvandmetöötluse rakenduse loomise etapid

Rakenduse loomine pilvandmete põhjal on sama lihtne kui tavalise rakenduse loomine. Selle loomise protsessist annab üldise kirjelduse mudel joonisel 6.



Joonis 6. Pilverakenduse loomise protsess.

Esiteks tuleb püstitada eelkõige iseenda jaoks vajalik teenuse spetsifikatsioon, mis määrab ära rakenduse lõpliku funktsionaalsuse. Sellele toetudes on võimalik programmi kirjutama hakata. Seejärel tuleb valida vajalikud vahendid rakenduse loomiseks. Nendeks on sobilik töökeskkond (näiteks Visual Studio) ning raamistik (näiteks .NET), mille valik peaks eelkõige põhinema eelistatud programmeerimiskeelel. Programmeerimiskeele

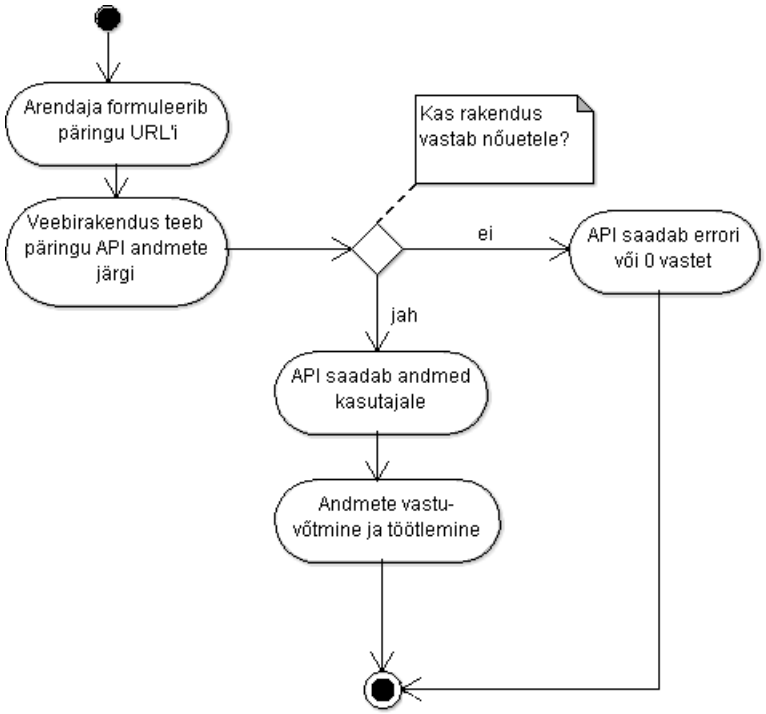
eelistus on ühtlasi ainuke asi, mis seab piirangud edaspidisel programmikirjutamisel, kuna arvestama peab keelele sobiva raamistiku funktsionaalsusi ning selle standardeid.

Seejärel tuleb luua soovitud rakendus, mille saab hiljem pilve üles seada. Võimalik on veebirakenduse asemel luua ka kohene pilverakendus. Näiteks Visual Studio's on võimalik luua pilverakendus, mida saab lihtsasti otse Azure'i laadida, kuid see pole tingimata vajalik. .NET raamistikule toetudes on võimalik luua ASP.NET veebirakendus. Staatilise koodi ossa, mis .NET raamistikus on .aspx laiendiga kood, kirjutatakse rakenduse visuaalne osa, mis on otseselt kasutajale nähtav ning ligipääsetav ning dünaamilises koodiosas, mis ASP.NET rakenduses on võib valikuna olla kirjutatud C# programmeerimiskeeles, on võimalik realiseerida vajalik loogika soovitava funktsionaalsuse lisamiseks. ASP.NET veebirakendus on täielikult ühilduv Azure'i tehnoloogiaga ning seega ei teki rakenduse üleslaadimisega pilve probleemi.

Tuleb jätta meelde, et mugav on kasutada pilvandmetel põhinevat informatsiooni. Viited pilvandmetöötluses kasutatavatele API'dele tehakse staatilises koodis ning see võimaldab ligipääsu kõigile API avalikele andmetele, mida saab lihtsasti mõne rea koodiga enda rakenduses realiseerida. Päringu sooritamisel saadab API andmed kasutajale ning neid töödeldakse vastu võttes vastavalt arendaja rakendatud nõuetele.

Mõnede API'de kasutamise puhul peab jälgima, et oleks olemas isiklik võti, mis identifitseerib andmete päringu sooritaja. See tagab suurema turvalisuse, andes võimaluse piirata võtme kasutust nii, et päringute tegemine on lubatud vaid kindlatele IP aadressidele.

API'de kasutamine ei ole keeruline. Täpsem kirjeldus API'de kasutamise kohta tuleb välja joonisest 7.



Joonis 7. API väljakutsumine ja kasutamine.

Kui API'dest saadud funktsionaalsus on realiseeritud, tuleb kirjutada valmis programmikood. Huvipakkuva API dokumentatsioon sisaldab tihti väga konkreetseid ja põhjalikke näiteid erinevate funktsionaalsuste tutvustamiseks, näiteks Google Maps API kaardil kuvatava markeri kasutamise õpetus [29], mille põhjal on äärmiselt lihtne oma rakendust täiendada. Peale HTML'i koodiosa täiendamise on võimalik programmikoodi kirjutada ka valitud programmeerimiskeeles dünaamilises koodis, näiteks selleks, et arendada välja mingisugune arvutusloogika, mille tulemused saab hiljem HTML'is välja kutsuda ja visuaalselt kasutajale kuvada. See pole vajalik ning soovi korral on võimalik kõik realiseerida ka HTML koodis. Programmi kirjutamisel tuleb lähtuda valitud raamistikus etteseatud reeglistikest ja standarditest.

Kui kood on valmis ning kirjutati pilverakendus, on kasutajal toimiv pilvlahendus, mis on kõigile kättesaadav pilve kaudu. Kui lõpptulemusena loodi veebirakendus, mida soovitakse pilve üles seada, tuleb see sõltuvalt raamistikust mõnedel juhtudel kõigepealt ümber töödelda pilverakenduseks, et oleks võimalik programm lõppkasutajani viimiseks pilve laadida.

Selleks, et võimalikult efektiivset rakendust luua, tuleb valida iseendale sobiv keel ja arenduseks asjakohane keskkond ning kindlasti juhinduda reeglistikest, mis raamistikule

on ettenähtud, et kood saaks üheselt mõistetav ja vastaks üldistele nõuetele. Lisaks on hea võimalusel kasutada varem realiseeritud pilvteenuseid, et juba olemasolevat asja uuesti mitte leiutama hakata.

3.2. Päikesekalkulaatori realiseerimine

3.2.1 Teenuse spetsifikatsioon

Rakenduse valikul mängis rolli see, et programm peab olema kasutatav Eestis ja vähemalt osad kasutatud andmed peavad pärinema avalikust infost. Realiseeritavaks ideeks sai päikese asukohtade kalkulaator maailmakaardi järgi.

Teenuse põhiline funktsionaalsus avaldub maailmakaardil mingile asukohale vajutades päikesetõusu ja -loojangu kellaaja kuvamisena asukoha koordinaatide järgi. Avaneb kirjakast, milles on kirjas asukoha geograafiline pikkus, laius ning nende järgi välja arvutatud päikesetõusu kellaajad antud kuupäeval. Samuti kuvatakse lisafunktsionaalsusena vastavale ikoonile vajutades antud asukoha ilmastiku tingimused (temperatuur ja üleüldine info). Ilmastikku on võimalik näha kõikide suurimate linnade kohta üle maailma.

Kaardil on võimalik liikuda, hoides all hiireklahvi ja tirides seda soovitud suunda. Eesti kaardilt eemale nihkudes või huvi korral teiste riikide vastu on olemas nupp „Eesti“, mis viib kasutaja tagasi Eesti koordinaatidele. Kaardi suurust on soovi korral võimalik suurendada ja vähendada vastavate nuppudega või hiirenuppu kerides. Juhul, kui kasutaja on huvitatud konkreetsest asukohast, on võimalik kasutada otsingut, mis sisaldab andmeid terve maailmakaardi asukohtade kohta. (Lisa 2)

Sarnaselt valitud teenusele on realiseeritud näiteks rakendus SunCalc, mis asub veebiaadressil suncalc.net ning kuvab kasutajale ette maailmakaardi, millele hiirega vajutades kuvatakse asukoht ja kuupäev ning päikese asukohad päeva vältel. Samuti on näha päikese positsioonid ja trajektoor kaardile joonistatud joontega. Valitud asukoha korral on võimalik kuvada ilmaennustus eraldi aknas. Mainitud rakendus on programmeeritud tervenisti Javascriptis ja pidevas arengujärgus.

Lõpptulemusena saan võrrelda enda rakenduse ning juba olemasolevate funktsionaalsust ning standarditele vastamist ning mõelda välja, kuidas on kõige hõlpsam ja kasulikum pilvandmete põhist rakendust luua.

3.2.2 Vahendi valik

Kuna populaarseim tüüplahenduste raamistik on .NET (Lisa 1), luuakse päikesekalkulaator antud keskkonnas. .NET raamistikul on paljud funktsionaalsusi, mis võimaldavad lihtsasti kasutada pilvandmeid ja erinevaid vajalikke API'sid. Tänu laialdasele kasutamisele ning suurele kogukonnale on .NET raamistikul piisavalt põhjalik dokumentatsioon ning palju õpetusi, mille põhjal uusi rakendusi arendada. Kuigi .NET'i suurim puudus on Windows'i põhisus, ei takista see antud juhul töö arengut ja rakenduse loomist.

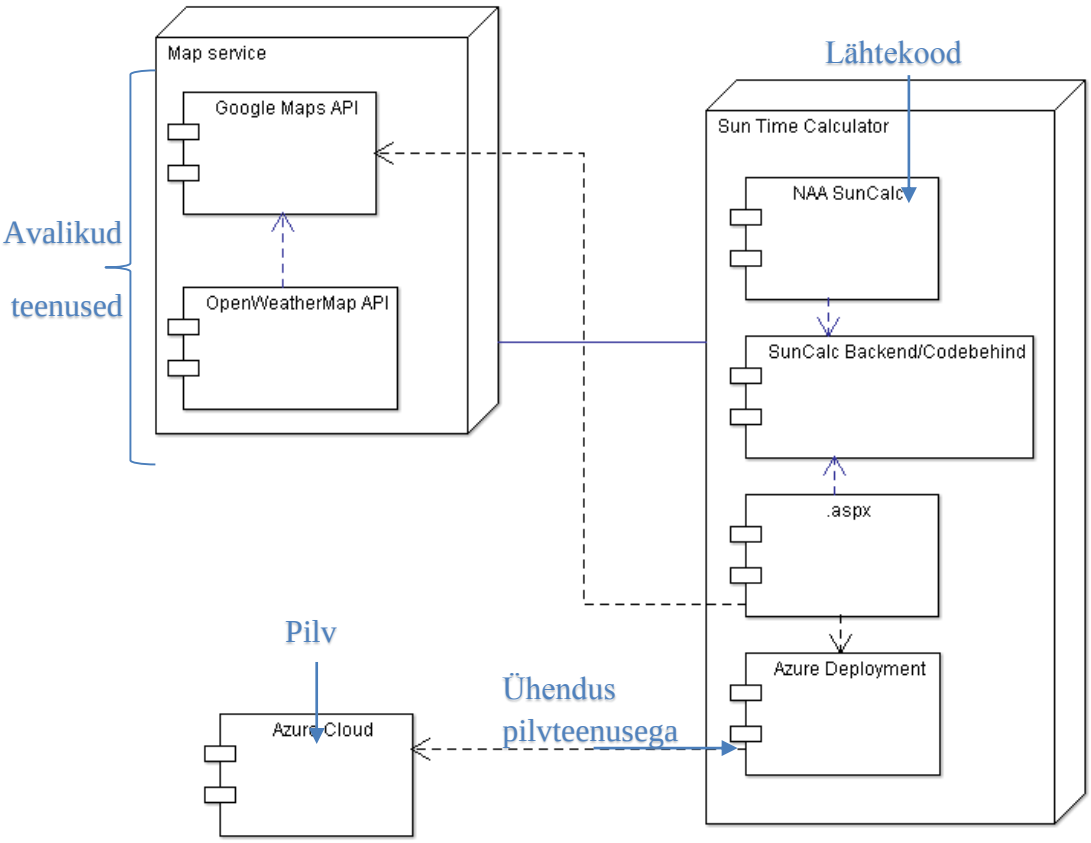
Päikesekalkulaator arendatakse ASP.NET veebirakendusena Visual Studio 2013 kasutades ning programmeeritavaks keeleks valitakse C# eelkõige isiklike eelistuste ning varasemate kogemuste tõttu. Lisaks on C# spetsiaalselt .NET raamistiku jaoks loodud keel ning antud hetkel loogiline valik. Rakendus seatakse katsetamiseks üles Microsoft Azure'i pilve, kust sellele on tingimusteta ligipääs sõltumata asukohast ja kasutajast. Kuna Azure'i kasutamine pärast kuu pikkust katseaja lõppu muutub tasuliseks, ei jää rakendus esialgu sinna püsivalt kättesaadavaks. Soovi korral on võimalik liikmelisust tasu eest pikendada.

Kaardipõhi, informatsioon koordinaatide ja asukohtade nimede ning muu kaardi otsene funktsionaalsus saadakse Google Maps API andmete kaudu, samuti vajalikud koordinaadid päikesetõusu- ja loojangu kellaaegade arvutamiseks. Suuremate linnade ilmastikutingimuste kuvamiseks on kasutatud OpenWeatherMap API't.

Kuna Google Maps API ning OpenWeatherMap API on JavaScripti põhised, on kaardi funktsionaalne osa loodud JavaScriptiga. Päikesekalkulaatori loogika, kus asuvad päikesetõusu ja -loojangu arvutused koordinaatide järgi, seisab eraldi koodina ning on realiseeritud C# programmeerimiskeeles.

3.2.3 Päikesekalkulaator

Rakendus koosneb päikesekalkulaatori loogikast ning avalikest teenustest, mille kaudu on kättesaadav kaart ja sellel kuvatav informatsioon. Lahendus on üles laetud pilve. Joonis 8 annab ülevaate üldisest programmi arhitektuurist.



Joonis 8. Programmi arhitektuur.

Päikesekalkulaator koosneb muuhulgas vabast lähtekoodist [30], mis realiseerib päikesetõusu- ja loojangu arvutamise loogika, kuid mida on modifitseeritud ja parandatud päikesekalkulaatori arendaja poolt. Avalikud andmed kutsutakse välja .aspx staatilises koodis, samal ajal kui ühendus päikesekalkulaatori loogika ning staatilise koodiga tehakse dünaamilises taustkoodis (code-behind). Lahendus on hetkel laetud üles Microsoft Azure'i poolt võimaldatud pilve.

Päikesekalkulaatori toimimise loogika on näidatud ära kasutuslugude diagrammina (joonis 9), kus arvuti on kasutatav töövahend, mis on vahelüliks teenuste ja lõppkasutaja vahel.

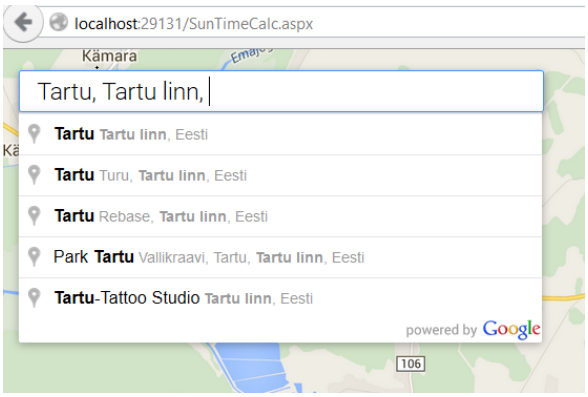
Joonis 9. Päikesekalkulaatori üldine mudel.

Selleks, et kuvada kasutajale nähtavat kaarti ja ilmastikku, tuleb teha päring Google Maps API ja OpenWeatherMap API pilvandmetest. Vajaliku informatsiooni API'dest saab kätte, luues nii-öelda päringu URL'i, mis kutsutakse välja rakenduse HTML osas. (Lisa 2) (Joonis 7) Google Maps API ja OpenWeatherMap API puhul tehakse seda Javascriptis. Tulemusena on kasutajal ees funktsionaalne Google Maps stiilis maailmakaart ning ilmastik suuremates linnades. (Lisa 2)

Päikesekalkulaatori puhul on näiteks kaart ja sellega seotud funktsionaalsused saadud teenusepakkujalt Google, mis pakub pilve kaudu rakendusena Google Maps API't. Lõppkasutajale on rakenduse pilve üles laadimise korral kättesaadav lõpliku teenusena päikesekalkulaator, mis sisaldab endas nii API kui arendaja rakendusi, mis omavahel

koostööd teevad (Joonis 1). Sama kehtib ilmastiku puhul kasutatud OpenWeatherMap API'ga.

Kasutajal on võimalik kaardil navigeerida tänu funktsionaalsusele, mille kasutatavad API'd kaasa annavad või mida arendaja on rakendanud. Näiteks soovitud asukoha otsimiseks on realiseeritud rakenduse koodi HTML osas koodilõik, mis kasutab Google Place Autocomplete vahendit. Autocomplete sisaldub API's ning seda on väga lihtne realiseerida Google Maps API dokumentatsiooni näidete baasil. Autocomplete paikneb Google Maps Javascript API asukohtade teegis ning sellega on võimalik anda rakendusele sarnaselt realiseeritud otsingukast Google Maps rakendusega. [27] Kui kasutaja hakkab sisestama aadressi, võimaldatakse tal teostada geograafiline otsing ning täidetakse edasine otsingusõna automaatselt rakenduse poolt (Joonis 10). Selle kasutamiseks tuleb rakendusse laadida Google Places teek, kasutades teegi parameetrit päritavas URL'is, mida on näha ka rakenduse koodis. (Lisa 2)



Joonis 10. Päikesekalkulaatori otsing.

Soovitud asukoha valimisel tuleb vajutada klahvi „enter", seejärel asukoht valideeritakse ning kaart liigutatakse vastavasse asukohta.

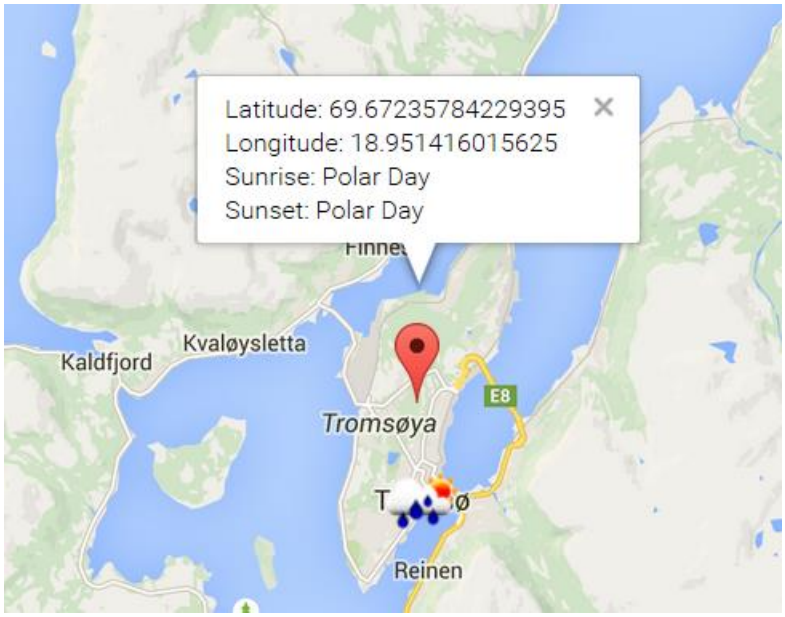
Navigeerimisel lisab mugavust ka nupp „Eesti", mille vajutamisel liigutatakse kaart tagasi Eesti keskpunkti koordinaatidele. Ka selle nupu funktsionaalsus tuleneb Google Maps API võimalustest. Kuna API sisaldab endas informatsiooni geograafiliste pikkuste ja laiuste kohta, on võimalik nende järgi seada kaardile mingi kindel keskpunkt ja siduda see nupuga, mis kasutaja tagasi valitud koordinaatidele viib (Lisa 2).

Kui kaart on navigeeritud õigesse kohta, on kasutajal võimalik märkida ära asukoht, kus soovitakse teada päikese tõusmise ja loojumise kellaaegu käesoleval kuupäeval. Asukoha ära märkimisel saadakse Google Maps API kaudu geograafiline pikkus ja laius, mille

järgi toimub päikesetõusu- ja loojangu kellaaegade arvutamine eraldiseisvas SunCalcLogic klassis. See on ühtlasi kõige pikem ja keerulisem osa koodist. Loogika on kirjutatud C# ning põhinetud on päikesepositsiooni arvutamise valemitel [26][28] ning samadel valemitel põhineval vabal lähtekoodil päikesetõusu- ja loojangu leidmiseks [30].

Päikesepositsiooni leidmiseks on vaja 4 muutujat: aeg; planeedi liikumine orbiidil ümber päikese, mille puhul tuleb arvestada, et tegu pole konstantse kiirusega, kuna planeedi orbiidil on iseärasused; pöörlemistelje ja planeedi orbiidi pinna vaheline nurk, mis ei ole võrdne 90 kraadiga ning planeedi vaatleja asukoht, mis määrab ära, kui kõrgele päike tõuseb [26]. Programmikood arvestab neid kõiki.

Liikudes piisavalt kõrgele või madalale, suudab programm välja arvutada ka polaarpäeva- ja öö võimalused (Joonis 11).



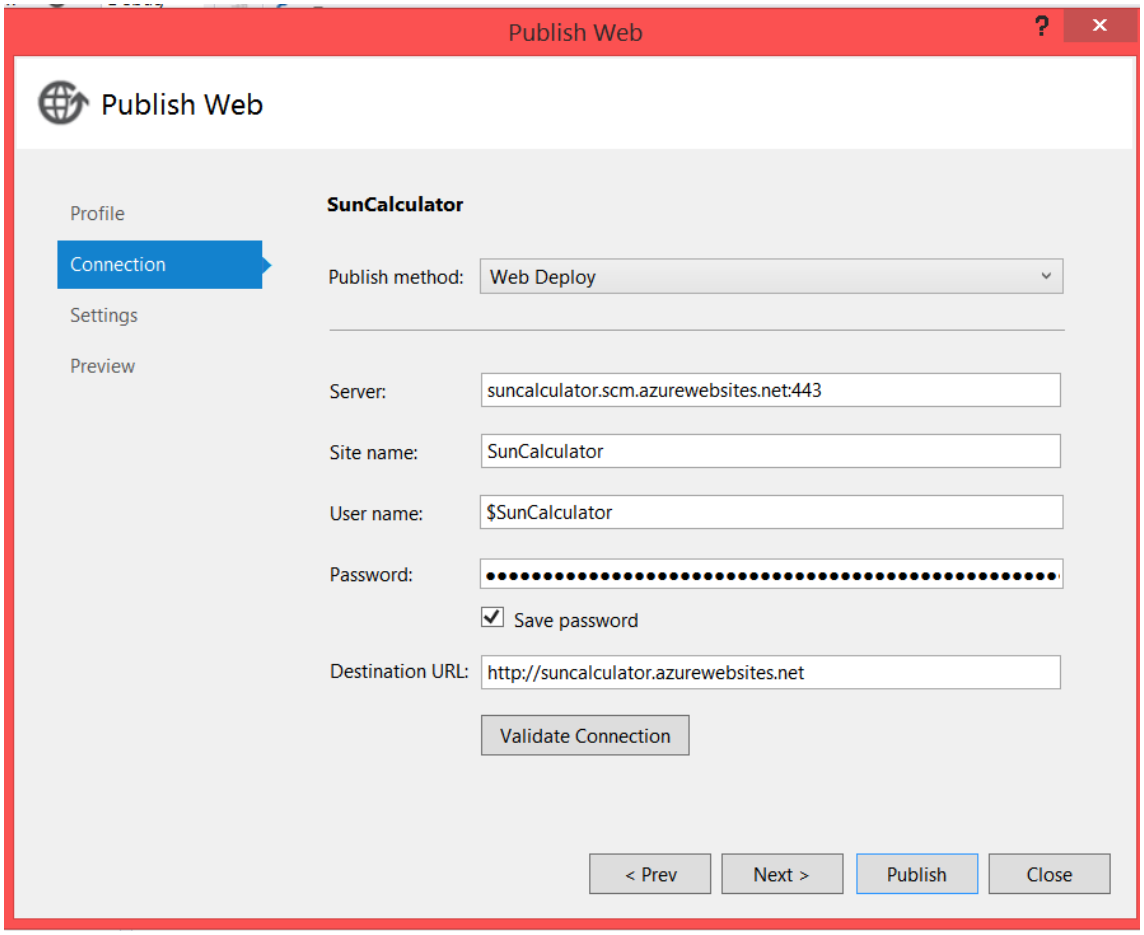
Joonis 11. Polaarpäeva kuvamine päikesekalkulaatoril.

Pärast arvutusi antakse tulemus edasi kaardile kuvamiseks ning seejärel kuvatakse andmed vastava kasutaja märgitud markeri asukoha kõrvale (Lisa 2).

Päikesekalkulaatori loomisel on põhinetud Microsoft'i .NET raamistiku disaini juhendile. Näiteks on oluline korrektsete nimetuste andmine erinevatele identifikaatoritele. Parameetrite nimetamisel kasutatakse camelCase'i, mille puhul esimene sõna on väikese algustähega ning edaspidised sõnad suurega, näiteks floatHour (Lisa 2). Muud identifikaatorid, sealhulgas nimeruumid ja klassid, nimetatakse niinimetatud

PascalCase'i kasutades, ehk iga sõna alustatakse suure algustähega. Loodud rakenduse puhul on näiteks nimeruum SunLogic, mis sisaldab klassi SunCalcLogic (Lisa 2) Rolli mängib ka sõnade liik ja eesliited. Reegleid on palju ning efektiivsema arenduse jaoks tuleb neid reegleid kindlasti jälgida vastavast juhendist [15].

Lõpptulemus seati prooviks üles Microsoft Azure'i pilvlahendusena. Selleks tuli Visual Studio avada ning ülesseatava rakenduse projekti peal parema hiireklahviga avada menüü, kus oli valik „Publish", mis ei nõudnud muud kui veebilehe andmete sisestamist ja soovi korral üldiste sätete muutmist. (Joonis 12) Kuna ASP.NET rakendus on täielikult ühilduv Azure'i pilvlahenduse võimalustega, võtab see vaid mõne hetke. Edaspidi tuleb koodi andmete täiendamisel rakendus uuesti pilve laadida, et muutused ka seal rakenduksid.



Joonis 12. Rakenduse laadimine Microsoft Azure'i

Võrreldes eelnevalt mainitud SunCalc.net rakendusega on kaart lihtsam ning vähesemate funktsionaalsustega, kuid täidab sama põhimõtet. Ilmastikutingimused on kuvatud otse kaardil ning ei avane eraldi aknas nagu SunCalc.net puhul. Päikesetõusu- ja loojangu

kellaajad ühtisid enamasti SunCalc.net rakenduse andmetega, kuid tekkisid mõned minutiliste erinevustega tulemused, kuna päikesekalkulaator kuvab andmeid täpsemalt ja SunCalc.net ümardab need. SunCalc.net on tervenisti realiseeritud JavaScript'is, antud töös loodud päikesekalkulaatori rakendus koosneb JavaScripti ja C# ühendatud loogikast. See on puhtalt isiklik eelistus. Kaardi visuaalset osa oli tänu API dokumentatsioonidele üpriski mugav JavaScriptiga realiseerida, isegi kui puudus varasem põhjalikum kogemus. Küll aga oli ebamugav realiseerida serveri- ja kliendivahelist suhtlust C# ja JavaScripti vahel.

Teenusena loodud päikesekalkulaator täitis etteseatud nõudmisi korrektselt. Lõpptulemusena valmis ASP.NET veebirakendus, mis laeti üles Azure'i pilvekeskkonda.

Kokkuvõte

Käesoleva töö lõpptulemusena loodud pilvandmetel põhinev päikesekalkulaatori rakendus töötab ning seega valitud lahendus on nõuetele vastav ja toimiv. Põhieesmärgina uuriti, kuidas luua rakendust, mis sobib pilvandmetöötlusesse ning lihtsustamiseks seletati arendusprotsess etappide kaupa lahti. Edaspidi on võimalik rakendusse mitmesuguseid funktsionaalsusi juurde lisada, kuid rakendus rahuldab püstituse tingimusi ka nendeta.

Dokumendi esimesed peatükid olid teoreetilised ülevaated rakenduse loomiseks vajaminevatest terminitest ja nõuetest, viimases osas viidi teooria praktikasse ning realiseeriti töötav pilvandmetöötluse rakendus päikesekalkulaatori näitel.

Pilvandmetöötluse rakenduse loomine on lihtne protsess ning ei erine kuigi palju tavalise programmi kirjutamisest. Lisakvaliteeti annavad juba olemasolevad pilverakendused, mille kasutamisel on võimalik hõlpsasti arendatavale rakendusele juba olemasolevaid andmeid rakendada.

Kasutatud kirjandus

[1] Cloud Computing Service Metrics Description. (2011). NIST standard SP 500-307:2011. U.S. Department of Commerce : National Institute of Standards and Technology

[2] A Partnership Made For the Cloud. [WWW] https://ec.europa.eu/digital-agenda/en/news/partnership-made-cloud (28.04.2015)

[3] The Role of e-Infrastructures In Natural-Disaster Response. [WWW] http://www.isgtw.org/feature/role-e-infrastructures-natural-disaster-response (28.04.2015)

[4] Eestist sai maailma parimate e-valitsemist kasutavate riikide ühenduse D5 asutajaliige. [WWW] http://uudisvoog.postimees.ee/?DATE=20141210&ID=352488 (28.04.2015)

[5] D5 London: Open Markets. [WWW] https://www.gov.uk/government/publications/d5-london-summit-themes/d5-london-open-markets (28.04.2015)

[6] „Building a Digital Democracy“ - Lessons from the D5 London summit. [WWW] http://gsablogs.gsa.gov/innovation/2014/12/19/building-a-digital-democracy-lessons-from-the-d5-london-summit/ (28.04.2015)

[7] NIST definition for SaaS, PaaS, IaaS. [WWW] https://cloudinfosec.wordpress.com/2013/05/04/nist-definition-for-saas-paas-iaas/ (08.05.2014)

[8] Framework. [WWW] http://techterms.com/definition/framework (16.04.2015)

[9] Baker, M. What is a Software Framework? And why should you like ’em? [WWW] http://info.cimetrix.com/blog/bid/22339/What-is-a-Software-Framework-And-why-should-you-like-em (16.04.2015)

[10] Edwin, N. M. (2014) Software Frameworks, Architectural and Design Patterns. *Journal of Software Engineering and Applications*, 7, 670-678.

[11] Overview of the .NET Framework. https://msdn.microsoft.com/en-us/library/zw4w595w.aspx (16.04.2015)

[12] ASP.NET - Introduction. [WWW] http://www.tutorialspoint.com/asp.net/asp.net_introduction.htm (27.04.2015)

[13] Chapter 1: Introduction to .NET. [WWW] https://technet.microsoft.com/en-us/library/bb496996.aspx (27.04.2015)

[14] Introduction to .NET. [WWW] http://www.codeproject.com/Articles/1821/Introduction-to-NET (27.04.2015)

[15] Framework Design Guidelines. [WWW] https://msdn.microsoft.com/en-us/library/ms229042.aspx (27.04.2015)

[16] Introduction. [WWW] https://github.com/castleproject/MonoRail/wiki (27.04.2015)

[17] Microsoft Project Orleans. [WWW] http://dotnet.github.io/orleans/Introduction/ (27.04.2015)

[18] Introduction to CakePHP. [WWW] http://book.cakephp.org/1.1/en/introduction-to-cakephp.html (27.04.2015)

[19] CodeIgniter at a Glance. [WWW] http://www.codeigniter.com/user_guide/overview/at_a_glance.html (27.04.2015)

[20] Introduction to the Spring Framework. [WWW] http://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html (27.04.2015)

[21] Open Source Metadata -Based Java ORM Framework for Cloud SaaS Applications. [WWW] http://athenasource.org/java/ (28. 04.2015)

[22] Why Django? [WWW] https://www.djangoproject.com/start/overview/ (28.04.2015)

[23] Introducing OpenStack.[WWW] http://www.openstack.org/blog/2010/07/introducing-openstack/ (28.04.2015)

[24] What is Ruby on Rails? [WWW] http://railsapps.github.io/what-is-ruby-rails.html (28.04.2015)

[25] Ramaze. [WWW] http://ramaze.net/documentation/index.html (28.04.2015)

[26] Astronomy Answers Position of the Sun. [WWW] http://aa.quae.nl/en/reken/zonpositie.html (13.05.2015)

[27] Places search box. [WWW] https://developers.google.com/maps/documentation/javascript/examples/places-searchbox (13.05.2015)

[28] Julian Day Numbers [WWW] http://quasar.as.utexas.edu/BillInfo/JulianDatesG.html (13.05.2015)

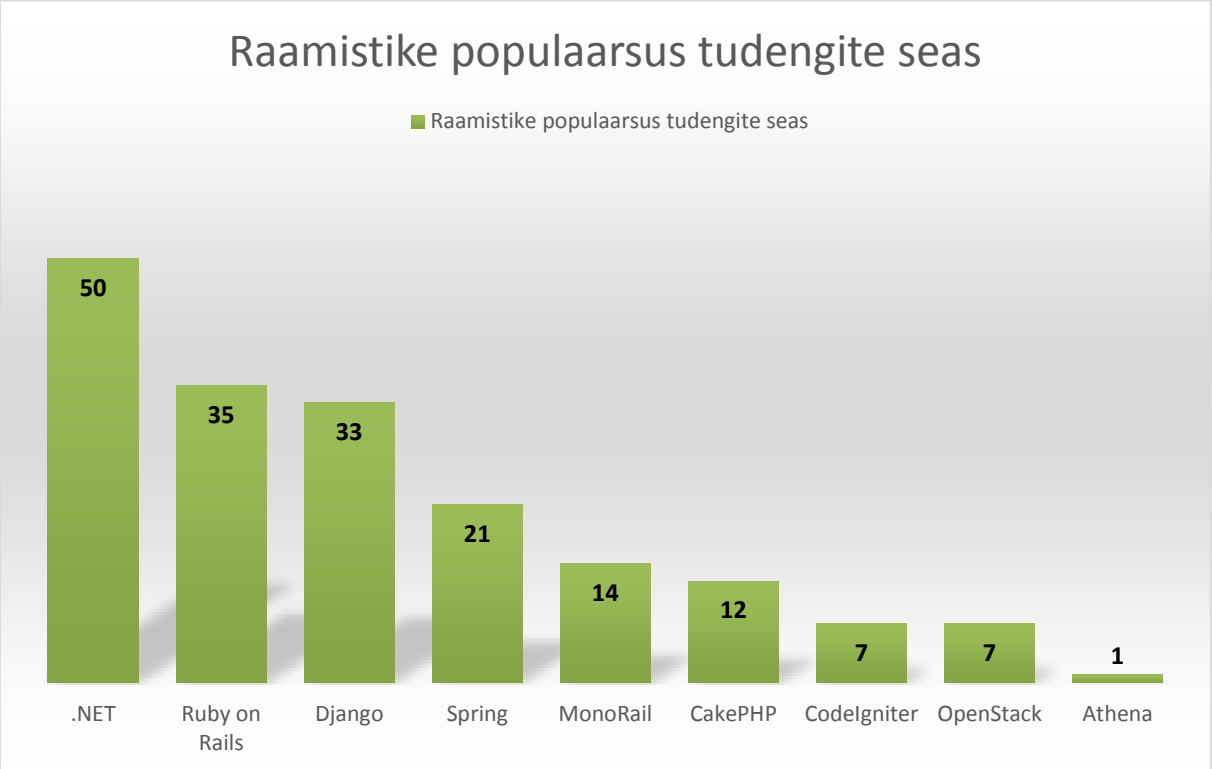
[29] Simple markers [WWW] https://developers.google.com/maps/documentation/javascript/examples/marker-simple (14.05.2015)

[30] Sunrise and Sunset in C#. [WWW] http://pointofint.blogspot.de/2014/06/sunrise-and-sunset-in-c.html (14.05.2015)

Lisa 1 Küsitlus teemal raamistikud

Küsitlus, mis oli suunitletud Eesti infotehnoloogia tudengitele, kogus 50 vastust. Eesmärgiks oli teada saada tudengite teadmised ja arvamused töös mainitud raamistikest, mida oli 11 - kõik töös mainitud raamistikud. Küsitluse põhjal teen järelduse kõige sobilikuma raamistiku leidmisel. Küsitlus oli lühike, kuid piisavalt informatiivne õige valiku tegemiseks.

Esimesena uuriti tudengitelt, millistest raamistikest nad üldse teadlikud ja kuulnud olid. Eesmärgiks oli teada saada eriraamistike populaarsuse osakaal tudengite seas põhimõttel, et mida populaarsem raamistik, seda rohkem sellest räägitakse ja teatakse. Samuti on loogiline, et mida suurema tuntusega on raamistik, seda rohkem on seda kasutatud ja leidub informatiivsemaid materjale, millele põhineda.

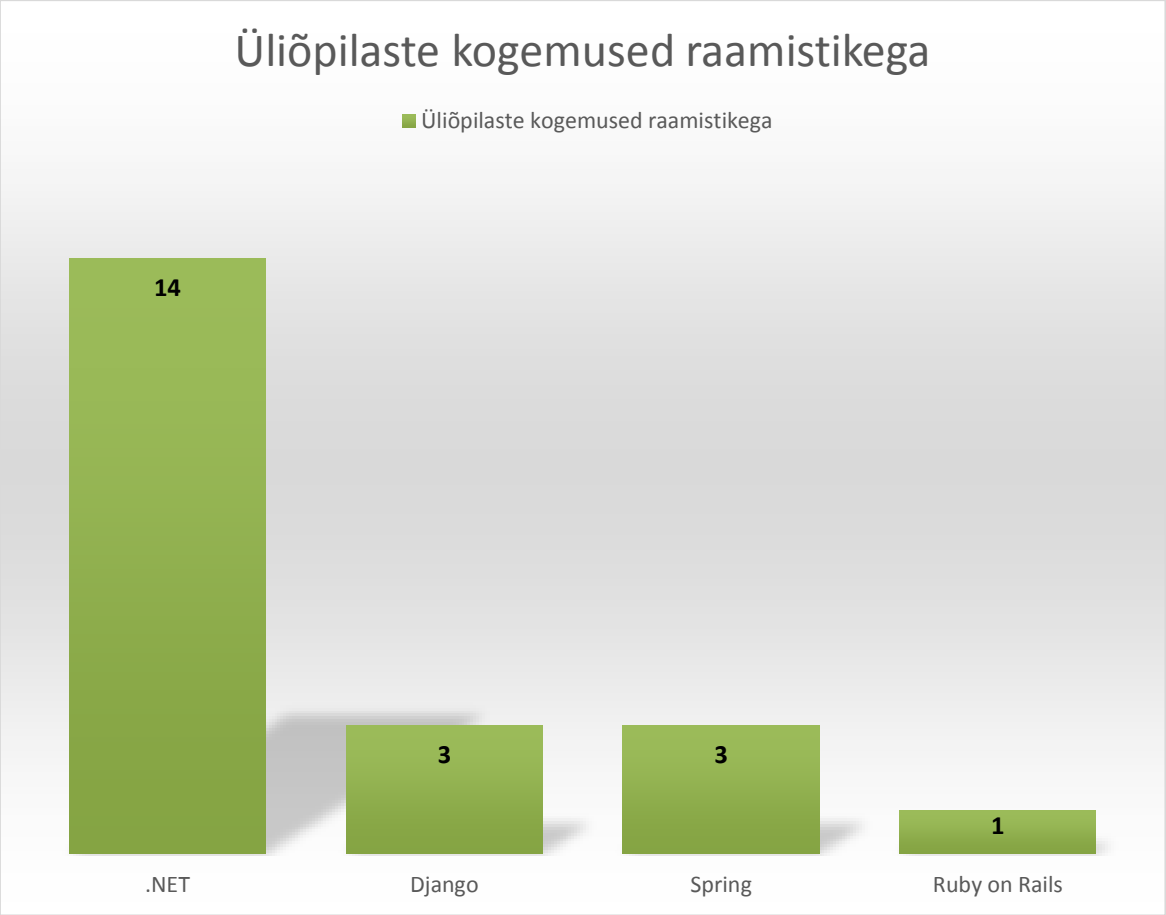


Selgub, et täpselt 100% vastanutest on tuttavad .NET raamistikuga. Järgnevad Ruby keele põhine Ruby on Rails 70% ja Pythoni põhine Django 66% vastanutega. Kõige vähem oli kuuldud Athena raamistikust, mille valis 1 vastaja ehk 2% vastanutest. Üldse ei teatud Ramaze ja Orleansi raamistikke. See võib olla põhjendatav sellega, et Ramaze on Ruby põhine raamistik, kuid eeldatavalt kasutatakse valdava enamuse poolt Ruby on Rails

raamistikku. Orleans on spetsiaalselt pilvandmetöötluse jaoks loodud platvorm ning kahjuks ei ole see veel suuremat populaarsust kogunud.

Järgnevalt küsiti tudengitelt, milliste raamistikega on nad isiklikult kokku puutunud ja tööd teinud ning seejärel arvamust nende poolt kasutatud raamistike kohta.

Kogemusi oli üliõpilastel vaid 4 raamistikuga 11st.



Taaskord oli enim kogemusi 28% vastanutest .NET raamistikuga, millele järgnesid 6% Django ja Spring ning 2% Ruby on Rails. Eelmise küsimuse tulemustega võrreldes näeme, et need 4 raamistikku olid ühtlasi ka kõige populaarsemad antud töö raamistike valikust.

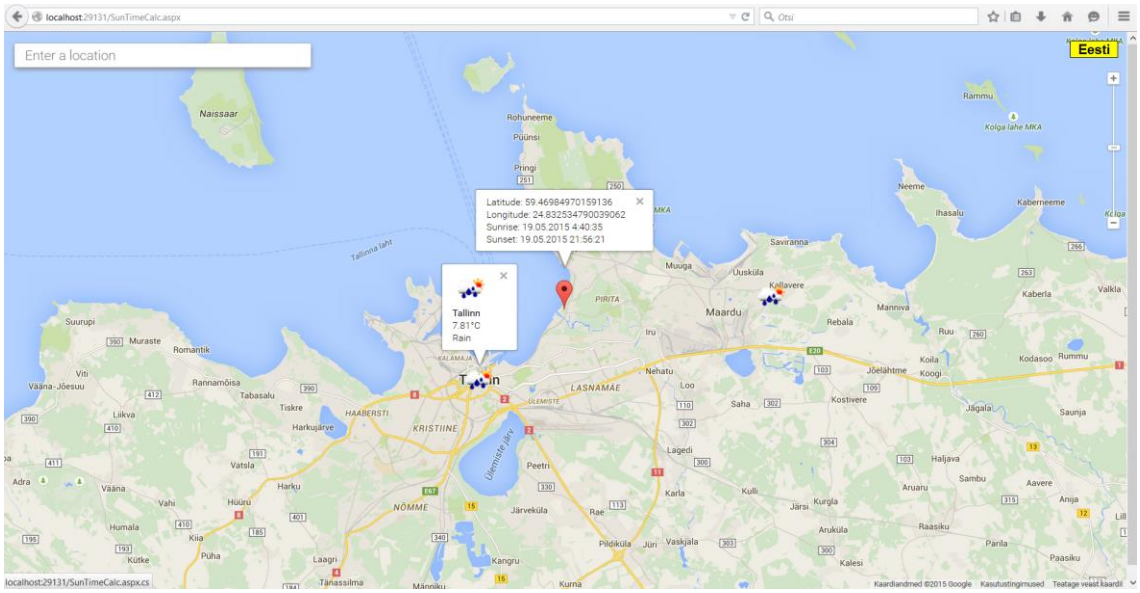
Viimasena uuriti arvamusi raamistike kohta, millega tudengitel kogemusi on. Alljärgnev tabel võtab kokku tudengite otsesed arvamused neist raamistikest.

Raamistik	Eelised	Puudused
.NET	• Täielikult välja arendatud • Hea kogukond ja toetus • Laialdaselt levinud • Lihtne kasutada • Paindlik • Palju funktsionaalsust • Objekt-orienteeritus • Sõltumatu keelest • Kiire rakenduste arendus	• Windowsi põhine • Fikseeritud ja muutumatu • Lihtne pöördprojekteerida • Palju allalaadimisi • Probleemid versioonidega • Raske ära õppida
Django	• Lihtne rakenduste loomine • Lihtne ära õppida • Hea dokumentatsioon ja õpetused • Hea kiireks prototüüpimiseks	-
Spring	• Head lahendused ettevõtetele • Lihtne ära õppida • Funktsionaalne	• Segadusttekitav alguses
Ruby on Rails	• Mugav kirjutada	-

Lisaks oli mainitud, et raamistikevahelised erinevused ei ole väga palju määravad ning rohkem loeb see, millist programmeerimiskeelt eelistatakse. Siiski, tuginedes tudengite arvamusele ning üleüldistele seisukohtadele, tundub loogiline valik pilvandmetöötluse rakenduse arendamiseks .NET raamistik.

Lisa 2 Päikesekalkulaatori programmikood ja ekraanikoopia

Ekraanikoopia töötavast rakendusest, mis kuvab valitud asukoha koordinaadid ja päikesetõusu- ja loojangu kellaajad ning Tallinna ilmastiku.



Koodi visuaalne HTML osa:

```
    <%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="SunTimeCalc.aspx.cs" Inherits="SunCalcNet.SunTimeCalc" %>
<script>
    // most code based on documentation found at
    https://developers.google.com/maps/documentation/javascript/
    // and http://openweathermap.org/api
</script>
<!DOCTYPE html>
<html>
  <head>
    <title>Sun Time Calculator</title>
    <meta name="viewport" content="initial-scale=1.0, user-
scalable=no">
    <meta charset="utf-8">
    <style>
      html, body, #map-canvas {
        height: 100%;
        margin: 0px;
        padding: 0px
      }
      .controls {
        background-color: #fff;
        border-radius: 2px;
        border: 1px solid transparent;
        box-shadow: 0 2px 6px rgb(128, 128, 128);
        box-sizing: border-box;
```

```
        font-family: Roboto;
        font-size: 18px;
        font-weight: 300;
        height: 32px;
        margin-left: 17px;
        margin-top: 16px;
        outline: none;
        padding: 0 11px 0 13px;
        text-overflow: ellipsis;
        width: 400px;
      }
      .controls:focus {
        border-color: #4d90fe;
      }
    </style>
    <script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyC-
DKTv2XZZ0YCZzPpzHg8oAtf4Wup8eI8"></script>
    <script
src="https://maps.googleapis.com/maps/api/js?v=3.exp&libraries=places&
signed_in=false"></script>
       <script>
        var map;
        var myCenter = new google.maps.LatLng(58.571098, 25.3699444);
//defines a location to a parameter
        var infowindow = new google.maps.InfoWindow();
        var currentLocation;
        var request;
        var gettingData = false;
        var openWeatherMapKey = "2b5a661e53ab2b9e414a32d2b407c93f"
//OpenWeatherMap API key
        // Add a control button that returns the user to Estonia
        function HomeButton(controlDiv, map) {
            controlDiv.style.padding = '12px';
            var controlUI = document.createElement('div');
            controlUI.style.backgroundColor = 'yellow';
            controlUI.style.border = '1px solid';
            controlUI.style.cursor = 'pointer';
            controlUI.style.textAlign = 'center';
            controlUI.title = 'Set map to Estonia';
            controlDiv.appendChild(controlUI);
            var controlText = document.createElement('div');
            controlText.style.fontFamily = 'Arial,sans-serif';
            controlText.style.fontSize = '18px';
            controlText.style.paddingLeft = '10px';
            controlText.style.paddingRight = '10px';
            controlText.innerHTML = '<b>Eesti<b>'
            controlUI.appendChild(controlText);
            // Setup click-event listener: simply set the map to
Estonia
            google.maps.event.addDomListener(controlUI, 'click',
function () {
                map.setCenter(myCenter)
            });
        }
```

```
        function initialize() {
            var mapOptions = {
                zoom: 8, //map zoom level
                center: myCenter, //centres the map to Estonia
                disableDefaultUI: true, //disables default elements on
the map
                zoomControl: true,//manually adds a zoom button
                zoomControlOptions: {
                    position: google.maps.ControlPosition.RIGHT_TOP
                },
                mapTypeId: google.maps.MapTypeId.ROADMAP //map type is
roadmap
            };
            map = new google.maps.Map(document.getElementById('map-
canvas'),
                mapOptions);
            google.maps.event.addListener(map, 'idle',
checkIfDataRequested);
            // Sets up and populates the info window with details
            map.data.addListener('click', function(event) {
                infowindow.setContent(
                 "<img src=" + event.feature.getProperty("icon") + ">"
                 + "<br /><strong>" +
event.feature.getProperty("city") + "</strong>"
                 + "<br />" + event.feature.getProperty("temperature")
+ "&deg;C"
                 + "<br />" + event.feature.getProperty("weather")
                 );
                infowindow.setOptions({
                    position:{
                        lat: event.latLng.lat(),
                        lng: event.latLng.lng()
                    },
                    pixelOffset: {
                        width: 0,
                        height: -15
                    }
                });
                infowindow.open(map);
            });
            // Create a DIV to hold the control button and call
HomeButton()
            var homeButtonDiv = document.createElement('div');
            var homeButton = new HomeButton(homeButtonDiv, map);
            //  homeButtonDiv.index = 1;
map.controls[google.maps.ControlPosition.TOP_RIGHT].push(homeButtonDiv
);
            var marker;
            function placeMarker(location) {       //places marker at
chosen location
                if (marker) {
                    marker.setPosition(location);
                } else {
```

```
                        marker = new google.maps.Marker({ //in case of
more than one button click, places a new marker
                            position: location,
                            map: map,
                            draggable: true
                        });
                        map.panTo(location); //pans to marker location
                    }
                    map.panTo(location);
                    currentLocation = location;
                    //data from sun calculation logic
                    var sunSetTime =
PageMethods.GetSunRiseAndSunSetTime(location.lat(), location.lng(),
OnSuccess, OnFail);
                }
                function OnSuccess(result, userContext, methodName) {
                    var sunRiseString = result[0];
                    var sunSetString = result[1];
                    if (parseFloat( result[2] ) >= 1 && parseInt(
result[4] ) == 0 )
                    {
                        sunSetString = 'Polar Night';
                    }
                    else if (parseFloat(result[3]) <= 1 &&
parseFloat(result[3]) != 0)
                    {
                        sunRiseString = 'Polar Day';
                    }
                    infowindow.setContent('Latitude: ' +
currentLocation.lat() +
                        '<br>Longitude: ' + currentLocation.lng() +
                        '<br>Sunrise: ' + sunRiseString +
                        '<br>Sunset: ' + sunSetString);
                    infowindow.open(map, marker); //opens a window with
information about latitude & longitude, sunrise & sunset
                }
                function OnFail(error, userContext, methodName) {
                    if (error !== null) {
                        alert("An error occurred: " +
error.get_message());
                    }
                }
                google.maps.event.addListener(map, 'click', function
(event) {
                    placeMarker(event.latLng);
                });
                //searchbox
                var input = /** @type {HTMLInputElement} */(
                document.getElementById('pac-input'));
                var autocomplete = new
google.maps.places.Autocomplete(input); //place autocomplete widget to
enable the user to search for a place
                autocomplete.bindTo('bounds', map);
map.controls[google.maps.ControlPosition.TOP_LEFT].push(input);
```

```
            google.maps.event.addListener(autocomplete,
'place_changed', function () {
                var place = autocomplete.getPlace();
                if (!place.geometry) {
                    return;
                }
                if (place.geometry.viewport) {
                    map.fitBounds(place.geometry.viewport);
                } else {
                    map.setCenter(place.geometry.location);
                    map.setZoom(17);
                }
            });
        }
        var checkIfDataRequested = function () {
            // Stop extra requests being sent
            while (gettingData === true) {
                request.abort();
                gettingData = false;
            }
            getCoords();
        };
        // Get the coordinates from the Map bounds
        var getCoords = function () {
            var bounds = map.getBounds();
            var NE = bounds.getNorthEast();
            var SW = bounds.getSouthWest();
            getWeather(NE.lat(), NE.lng(), SW.lat(), SW.lng());
        };
        // Make the weather request
        var getWeather = function (northLat, eastLng, southLat,
westLng) {
            gettingData = true;
            var requestString =
"http://api.openweathermap.org/data/2.5/box/city?bbox="
                                + westLng + "," + northLat + ","
//left top
                                + eastLng + "," + southLat + ","
//right bottom
                                + map.getZoom()
                                + "&cluster=yes&format=json"
                                + "&APPID=" + openWeatherMapKey;
            request = new XMLHttpRequest();
            request.onload = proccessResults;
            request.open("get", requestString, true);
            request.send();
        };
        // Take the JSON results and proccess them
        var proccessResults = function () {
            console.log(this);
            var results = JSON.parse(this.responseText);
            if (results.list.length > 0) {
                resetData();
```

```
                for (var i = 0; i < results.list.length; i++) {
geoJSON.features.push(jsonToGeoJson(results.list[i]));
                }
                drawIcons(geoJSON);
            }
        };
        var infowindow = new google.maps.InfoWindow();
        // For each result that comes back, convert the data to
geoJSON
        var jsonToGeoJson = function (weatherItem) {
            var feature = {
                type: "Feature",
                properties: {
                    city: weatherItem.name,
                    weather: weatherItem.weather[0].main,
                    temperature: weatherItem.main.temp,
                    min: weatherItem.main.temp_min,
                    max: weatherItem.main.temp_max,
                    humidity: weatherItem.main.humidity,
                    pressure: weatherItem.main.pressure,
                    windSpeed: weatherItem.wind.speed,
                    windDegrees: weatherItem.wind.deg,
                    windGust: weatherItem.wind.gust,
                    icon: "http://openweathermap.org/img/w/"
                          + weatherItem.weather[0].icon + ".png",
                    coordinates: [weatherItem.coord.lon,
weatherItem.coord.lat]
                },
                geometry: {
                    type: "Point",
                    coordinates: [weatherItem.coord.lon,
weatherItem.coord.lat]
                }
            };
            // Set the custom marker icon
            map.data.setStyle(function (feature) {
                return {
                    icon: {
                        url: feature.getProperty('icon'),
                        anchor: new google.maps.Point(25, 25)
                    }
                };
            });
            // returns object
            return feature;
        };
        // Add the markers to the map
        var drawIcons = function (weather) {
            map.data.addGeoJson(geoJSON);
            // Set the flag to finished
            gettingData = false;
        };
        // Clear data layer and geoJSON
        var resetData = function () {
```

```
            geoJSON = {
                type: "FeatureCollection",
                features: []
            };
            map.data.forEach(function (feature) {
                map.data.remove(feature);
            });
        };
        google.maps.event.addDomListener(window, 'load', initialize);
    </script>
  </head>
  <body>
    <input id="pac-input" class="controls" type="text"
        placeholder="Enter a location">
    <div id="map-canvas"></div>
      <form id="form1" runat="server">
          <div>
              <asp:ScriptManager ID='ScriptManager1' runat='server'
EnablePageMethods='true' />
          </div>
    </form>
  </body>

</html>
```

Päikesekalkulaatori loogika

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;

//calculates the sunrise and sunset times at given location
//based on logic of http://pointofint.blogspot.de/2014/06/sunrise-and-
sunset-in-c.html
//based on the formulas found at
http://aa.quae.nl/en/reken/zonpositie.html
// http://quasar.as.utexas.edu/BillInfo/JulianDatesG.html for
julianDayNumber

namespace SunLogic
{
    public class SunCalcLogic
    {
        public const double StraightAngle = 180.0000f;
        static public double RadiansToDegrees(double angleInRadians)
// converts radians to degrees
        {
            double angleInDegrees = (StraightAngle * angleInRadians /
Math.PI);
            return angleInDegrees;
        }
        static public double DegreesToRadians(double angleInDegrees)
// converts degrees to radians
```

```
        {
            double angleInRadians = (Math.PI * angleInDegrees /
StraightAngle);
            return angleInRadians;
        }
        static public double GetJulianDay(int year, int month, int
day) // function to calculate Julian day from calendar days
        {
            double a = Math.Floor( year / 100.0f );
            double b = Math.Floor( a / 4.0f );
            double c = Math.Floor( 2.0f - a + b );
            double eccentricity = Math.Floor(365.25f * (year + 4716));
            double f = Math.Floor(30.6001f * (month + 1));
            double julianDay = c + day + eccentricity + f - 1524.5f;
            return julianDay;
        }
        static public double GetJulianDay(DateTime date)
        {
            return GetJulianDay(date.Year, date.Month, date.Day);
        }
        static public double CalcJulianDayToCenturies(double
julianDay) //converts Julian Day to centuries
        {
            double centuries = (julianDay - 2451545.0f) / 36525.0f;
            return centuries;
        }

        static public double CalcCenturiesToJulianDay(double
centuries) //converts centuries to Julian Day
        {
            double julianDay = centuries * 36525.0f + 2451545.0f;
            return julianDay;
        }
        static public double CalcGeomMeanLongSun(double
centuriesSinceTwoThousand) //calculates the Geometric Mean Longitude
of the Sun in degrees
        {
            double L0 = 280.46646 + centuriesSinceTwoThousand *
(36000.76983 + 0.0003032 * centuriesSinceTwoThousand);
            while (L0 > 360.0)
            {
                L0 -= 360.0;
            }
            while (L0 < 0.0)
            {
                L0 += 360.0;
            }
            return L0;   // in degrees
        }
        static public double CalcGeomMeanAnomalySun() // calculates
the Geometric Mean Anomaly of the Sun in degrees
        {
            double M = -3.59 + 0.98560 * GetJulianDay(DateTime.Today);
            return M;    // in degrees
```

```
        }
        static public double CalcEccentricityEarthOrbit(double
centuriesSinceTwoThousand) // calculates the eccentricity of Earth's
orbit
        {
            double eccentricity = 0.016708634 -
centuriesSinceTwoThousand * (0.000042037 + 0.0000001267 *
centuriesSinceTwoThousand);
            return eccentricity;    // unitless
        }
        static public double CalcSunEquationOfCenter(double
centuriesSinceTwoThousand) //calculates the equation of the center for
the sun in degrees
        {
            double meanAnomalyDegrees = CalcGeomMeanAnomalySun();

            double meanAnomalyRadians =
DegreesToRadians(meanAnomalyDegrees);
            double sinm = Math.Sin(meanAnomalyRadians);
            double sin2m = Math.Sin(2 * meanAnomalyRadians);
            double sin3m = Math.Sin(3 * meanAnomalyRadians);

            double C = sinm * (1.914602 - centuriesSinceTwoThousand *
(0.004817 + 0.000014 * centuriesSinceTwoThousand)) + sin2m * (0.019993
- 0.000101 * centuriesSinceTwoThousand) + sin3m * 0.000289;
            return C;    // in degrees
        }

        static public double GetSunTrueLongitude(double
centuriesSinceTwoThousand) //calculate the true longitude of the sun
in degrees
        {
            double longitude =
CalcGeomMeanLongSun(centuriesSinceTwoThousand);
            double center =
CalcSunEquationOfCenter(centuriesSinceTwoThousand);

            double sunTrueLongitude = longitude + center;
            return sunTrueLongitude;      // in degrees
        }
        static public double CalcSunTrueAnomalyInDegrees(double
centuriesSinceTwoThousand) //calculate the true anomaly of the sun in
degrees
        {
            double meanAnomalyDegrees = CalcGeomMeanAnomalySun();
            double center =
CalcSunEquationOfCenter(centuriesSinceTwoThousand);

            double sunTrueAnomaly = meanAnomalyDegrees + center;
            return sunTrueAnomaly;  // in degrees
        }
        static public double
GetDistanceToTheSunInAstronomicalUnits(double
centuriesSinceTwoThousand) //calculate the distance to the sun in AUs
```

```
        {
            double sunTrueAnomaly =
CalcSunTrueAnomalyInDegrees(centuriesSinceTwoThousand);
            double eccentricity =
CalcEccentricityEarthOrbit(centuriesSinceTwoThousand);

            double sunRadiusVectorInAU = (1.000001018 * (1 - Math.Pow(
eccentricity, 2 ))) / (1 + eccentricity *
Math.Cos(DegreesToRadians(sunTrueAnomaly)));
            return sunRadiusVectorInAU;   // in AUs
        }
        static public double GetSunApparentLongitude(double
centuriesSinceTwoThousand) //calculate the apparent longitude of the
sun in degrees
        {
            double sunApparentLongitude =
GetSunTrueLongitude(centuriesSinceTwoThousand);

            double omega = 125.04 - 1934.136 *
centuriesSinceTwoThousand;
            double lambda = sunApparentLongitude - 0.00569 - 0.00478 *
Math.Sin(DegreesToRadians(omega));
            return lambda;    // in degrees
        }

        static public double
GetMeanObliquityOfEclipticInDegrees(double centuriesSinceTwoThousand)
//calculates the mean obliquity of the ecliptic
        {
            double seconds = 21.448 - centuriesSinceTwoThousand *
(46.8150 + centuriesSinceTwoThousand * (0.00059 -
centuriesSinceTwoThousand * (0.001813)));
            double meanObliquityOfEcliptic = 23.0 + (26.0 + (seconds /
60.0)) / 60.0;
            return meanObliquityOfEcliptic;     // in degrees
        }
        static public double GetCorrectedObliquity(double
centuriesSinceTwoThousand) //calculates the corrected obliquity of the
ecliptic
        {
            double meanObliquityOfEcliptic =
GetMeanObliquityOfEclipticInDegrees(centuriesSinceTwoThousand);

            double omega = 125.04 - 1934.136 *
centuriesSinceTwoThousand;
            double eccentricity = meanObliquityOfEcliptic + 0.00256 *
Math.Cos(DegreesToRadians(omega));
            return eccentricity;    // in degrees
        }
        static public double GetSunRightAscension(double
centuriesSinceTwoThousand) //calculates the right acscension of the
sun
        {
```

```
            double eccentricity =
GetCorrectedObliquity(centuriesSinceTwoThousand);
            double lambda =
GetSunApparentLongitude(centuriesSinceTwoThousand);
            double tananum = (Math.Cos(DegreesToRadians(eccentricity))
* Math.Sin(DegreesToRadians(lambda)));
            double tanadenom = (Math.Cos(DegreesToRadians(lambda)));
            double alpha = RadiansToDegrees(Math.Atan2(tananum,
tanadenom));
            return alpha;     // in degrees
        }
        static public double GetSunDeclination(double
centuriesSinceTwoThousand) // calculates the declination of the sun
        {
            double eccentricity =
GetCorrectedObliquity(centuriesSinceTwoThousand);
            double lambda =
GetSunApparentLongitude(centuriesSinceTwoThousand);
            double sint = Math.Sin(DegreesToRadians(eccentricity)) *
Math.Sin(DegreesToRadians(lambda));
            double theta = RadiansToDegrees(Math.Asin(sint));
            return theta;     // in degrees
        }

        static public double GetEquationOfTimeInMinutes(double
centuriesSinceTwoThousand) //calculates the difference between true
solar time and mean time
        {
            double epsilon =
GetCorrectedObliquity(centuriesSinceTwoThousand);
            double l0 =
CalcGeomMeanLongSun(centuriesSinceTwoThousand);
            double eccentricity =
CalcEccentricityEarthOrbit(centuriesSinceTwoThousand);
            double meanAnomalyDegrees = CalcGeomMeanAnomalySun();
            double y = Math.Tan(DegreesToRadians(epsilon) / 2.0);
            y *= y;
            double sin2l0 = Math.Sin(2.0 * DegreesToRadians(l0));
            double sinm =
Math.Sin(DegreesToRadians(meanAnomalyDegrees));
            double cos2l0 = Math.Cos(2.0 * DegreesToRadians(l0));
            double sin4l0 = Math.Sin(4.0 * DegreesToRadians(l0));
            double sin2m = Math.Sin(2.0 *
DegreesToRadians(meanAnomalyDegrees));
            double Etime = y * sin2l0 - 2.0 * eccentricity * sinm +
4.0 * eccentricity * y * sinm * cos2l0
            - 0.5 * y * y * sin4l0 - 1.25 * eccentricity *
eccentricity * sin2m;
            return RadiansToDegrees(Etime) * 4.0;    // in minutes of
time
        }
        static public double GetHourAngleOfSunrise(double lat, double
solarDec, ref double polarity ) //calculates the hour angle of the sun
at sunrise for the latitude
```

```
        {
            double latRad = DegreesToRadians(lat);
            double sdRad = DegreesToRadians(solarDec);

            double HAarg = (Math.Cos(DegreesToRadians(90.833)) /
(Math.Cos(latRad) * Math.Cos(sdRad)) - Math.Tan(latRad) *
Math.Tan(sdRad));
            double hourAngle =
(Math.Acos(Math.Cos(DegreesToRadians(90.833)) / (Math.Cos(latRad) *
Math.Cos(sdRad)) - Math.Tan(latRad) * Math.Tan(sdRad)));
            if ( RadiansToDegrees( latRad  ) > 66.55f ||
RadiansToDegrees( latRad ) < -66.55f)
            {
                polarity = Math.Tan(sdRad) - Math.Tan(latRad);
            }
            return hourAngle; // in radians
        }
        static public double GetHourAngleOfSunset(double lat, double
solarDec, ref double polarity) //calculates the hour angle of the sun
at sunset for the latitude
        {
            double latRad = DegreesToRadians(lat);
            double sdRad = DegreesToRadians(solarDec);

            double HAarg = (Math.Cos(DegreesToRadians(90.833)) /
(Math.Cos(latRad) * Math.Cos(sdRad)) - Math.Tan(latRad) *
Math.Tan(sdRad));
            double hourAngle =
(Math.Acos(Math.Cos(DegreesToRadians(90.833)) / (Math.Cos(latRad) *
Math.Cos(sdRad)) - Math.Tan(latRad) * Math.Tan(sdRad)));
            if ( RadiansToDegrees( latRad ) > 66.55f ||
RadiansToDegrees( latRad ) < -66.55f)
            {
                polarity = Math.Tan(sdRad) - Math.Tan(latRad);
            }
            return hourAngle; // in radians
        }
        static public double GetSunriseUTC(double julianDayNumber,
double latitude, double longitude) //calculates the UTC of sunrise for
the given day at the given location
        {
            double centuriesSinceTwoThousand =
CalcJulianDayToCenturies(julianDayNumber);
            // *** Find the time of solar noon at the location, and
use
            // that declination. This is better than start of the
            // Julian day
            double noonMin =
GetSolarNoonUTC(centuriesSinceTwoThousand, longitude);
            double tNoon = CalcJulianDayToCenturies(julianDayNumber +
noonMin / 1440.0);
            // *** First pass to approximate sunrise (using solar
noon)
            double equationTime = GetEquationOfTimeInMinutes(tNoon);
```

```
            double solarDec = GetSunDeclination(tNoon);
            double polarity = 0;
            double hourAngle = GetHourAngleOfSunrise(latitude,
solarDec, ref polarity);
            double delta = longitude - RadiansToDegrees(hourAngle);
            double timeDiff = 4 * delta; // in minutes of time
            double timeUTC = 720 + timeDiff - equationTime;      // in
minutes
            double newt =
CalcJulianDayToCenturies(CalcCenturiesToJulianDay(centuriesSinceTwoTho
usand) + timeUTC / 1440.0);
            equationTime = GetEquationOfTimeInMinutes(newt);
            solarDec = GetSunDeclination(newt);
            hourAngle = GetHourAngleOfSunrise(latitude, solarDec, ref
polarity);
            delta = longitude - RadiansToDegrees(hourAngle);
            timeDiff = 4 * delta;
            timeUTC = 720 + timeDiff - equationTime; // in minutes
            return timeUTC;
        }

        static public double GetSolarNoonUTC(double
centuriesSinceTwoThousand, double longitude) //calculates the UTC of
solar noon for the given day at the given location
        {
            // First pass uses approximate solar noon to calculate
eqtime
            double tNoon =
CalcJulianDayToCenturies(CalcCenturiesToJulianDay(centuriesSinceTwoTho
usand) + longitude / 360.0);
            double equationTime = GetEquationOfTimeInMinutes(tNoon);
            double solNoonUTC = 720 + (longitude * 4) - equationTime;
// min
            double newt =
CalcJulianDayToCenturies(CalcCenturiesToJulianDay(centuriesSinceTwoTho
usand) - 0.5 + solNoonUTC / 1440.0);
            equationTime = GetEquationOfTimeInMinutes(newt);
            // double solarNoonDec = GetSunDeclination(newt);
            solNoonUTC = 720 + (longitude * 4) - equationTime; // min
            return solNoonUTC;
        }
        static public double GetSunsetUTC(double julianDayNumber,
double latitude, double longitude, ref double polarity) //calculates
the UTC of sunset for the given day at given location in minutes
        {
            var centuriesSinceTwoThousand =
CalcJulianDayToCenturies(julianDayNumber);
            var equationTime =
GetEquationOfTimeInMinutes(centuriesSinceTwoThousand);
            var solarDec =
GetSunDeclination(centuriesSinceTwoThousand);
            var hourAngle = GetHourAngleOfSunset(latitude, solarDec,
ref polarity);
            hourAngle = -hourAngle;
```

```
            var delta = longitude + RadiansToDegrees(hourAngle);
            var timeUTC = 720 - (4.0 * delta) - equationTime;
            return timeUTC;
        }
        static public double GetSunriseUTC(double julianDayNumber,
double latitude, double longitude, ref double polarity) //calculates
the UTC of sunrise for the given day at given location in minutes
        {
            var centuriesSinceTwoThousand =
CalcJulianDayToCenturies(julianDayNumber);
            var equationTime =
GetEquationOfTimeInMinutes(centuriesSinceTwoThousand);
            var solarDec =
GetSunDeclination(centuriesSinceTwoThousand);
            var hourAngle = GetHourAngleOfSunrise(latitude, solarDec,
ref polarity);
            var delta = longitude + RadiansToDegrees(hourAngle);
            var timeUTC = 720 - (4.0 * delta) - equationTime;
            return timeUTC;
        }
        static public string GetTimeString(double time, int timezone,
double julianDayNumber, bool useDaylightSavingsTime)
        {
            var timeLocal = time + (timezone * 60.0);
            var riseT = CalcJulianDayToCenturies(julianDayNumber +
time / 1440.0);
            timeLocal += ((useDaylightSavingsTime) ? 60.0 : 0.0);
            return GetTimeString(timeLocal);
        }
        static public DateTime GetDateTime(double time, int timezone,
DateTime date, bool useDaylightSavingsTime)
        {
            double julianDayNumber = GetJulianDay(date);
            var timeLocal = time + (timezone * 60.0);
            var riseT = CalcJulianDayToCenturies(julianDayNumber +
time / 1440.0);
            timeLocal += ((useDaylightSavingsTime) ? 60.0 : 0.0);
            return GetDateTime(timeLocal, date);
        }
        static private string GetTimeString(double minutes)
        {
            string output = "";
            if ((minutes >= 0) && (minutes < 1440))
            {
                var floatHour = minutes / 60.0;
                var hour = Math.Floor(floatHour);
                var floatMinute = 60.0 * (floatHour -
Math.Floor(floatHour));
                var minute = Math.Floor(floatMinute);
                var floatSec = 60.0 * (floatMinute -
Math.Floor(floatMinute));
                var second = Math.Floor(floatSec + 0.5);
                if (second > 59)
                {
```

```
                    second = 0;
                    minute += 1;
                }
                if ((second >= 30)) minute++;
                if (minute > 59)
                {
                    minute = 0;
                    hour += 1;
                }
                output = String.Format("{0} : {1}", hour, minute);
            }
            else
            {
                return "error";
            }
            return output;
        }
        static private DateTime GetDateTime(double minutes, DateTime
date)
        {
            double doubleHour = minutes / 60.0;
            double hour = Math.Floor(doubleHour);
            double doubleMinute = 60.0 * (doubleHour -
Math.Floor(doubleHour));
            double minute = Math.Floor(doubleMinute);
            double doubleSecond = 60.0 * (doubleMinute -
Math.Floor(doubleMinute));
            double second = Math.Floor(doubleSecond + 0.5);
            if (second > 59)
            {
                second = 0;
                minute += 1;
            }
            if ((second >= 30)) minute++;
            if (minute > 59)
            {
                minute = 0;
                hour += 1;
            }
            if (double.IsNaN(hour) || double.IsNaN(minute) ||
double.IsNaN(second))
            {
                hour = 0.0f;
                minute = 0.0f;
                second = 0.0f;
            }
            var day = date.Day;
            if( hour >= 24 )
            {
                day -= 1;
                hour %= 24;
            }
            else if( hour <= 0 )
            {
```

```
                day += 1;
                hour *= -1;
            }
            return new DateTime(date.Year, date.Month, day, (int)hour,
(int)minute, (int)second);
        }
    }
}
```

HTML osa code-behind, mis ühendab loogikat ja visuaalset osa.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Drawing;
using SunLogic;
using System.Web.Services;
namespace SunCalcNet
{
    public partial class SunTimeCalc : System.Web.UI.Page
    {
        [WebMethod]
        public static string[] GetSunRiseAndSunSetTime(double lat,
double lon)
        {
            double JD = SunCalcLogic.GetJulianDay(DateTime.Today);
            double polaritySet = 0, polarityRise = 0;
            double sunSet = SunCalcLogic.GetSunsetUTC(JD, lat, lon,
ref polaritySet);
            double sunRise = SunCalcLogic.GetSunriseUTC(JD, lat, lon,
ref polarityRise);
            string[] result = new string[5];
            DateTime sunRiseDate = SunCalcLogic.getDateTime(sunRise,
2, DateTime.Today, true);
            DateTime sunSetDate = SunCalcLogic.getDateTime(sunSet, 2,
ADateTime.Today, true);
            result[0] = sunRiseDate.ToString();
            result[1] = sunSetDate.ToString();
            result[2] = polaritySet.ToString();
            result[3] = polarityRise.ToString();
            result[4] = sunRiseDate.Hour != 0 && sunRiseDate.Minute !=
0 && sunRiseDate.Second != 0 ? "1" : "0";
            return result;
        }
    }

}
```