Haaga-Helia
ammattikorkeakoulu Oy

Fatman Frame tuotteen globalisointi

Petri Kortelainen

Opinnäytetyö
Tietotekniikan koulutusohjelma
2017

Tiivistelmä

Haaga-Helia ammattikorkeakoulu Oy

Tekijä(t)
Petri Kortelainen

Koulutusohjelma
Tietojenkäsittely

Opinnäytetyön otsikko
Fatman Frame tuotteen globalisointi

Sivu- ja liitesivumäärä
19 + 6

Tämä opinnäytetyö on projektityyppinen. Projekti koskee Fatman Oy:n tuotetta Frame, joka on .NET viitekehyksellä toteutettu sovellus. Yritykselle syntyi tarve globalisoida Frame-sovellus, kun yritys sai asiakkaita Suomen aikavyöhykkeen ulkopuolelta.

Sovellus on toistaiseksi käyttänyt suoraan järjestelmän aikaa, joka on Suomen aikavyöhykkeellä. Tietokantaa käyttää rinnalla usea muu sovellus, joka tekee globalisoinnista haasteellista. Todetaan että suositeltava käytäntö .NET viitekehyksen projekteissa on käyttää UTC-tietokantaa. Kun esitetään tai otetaan vastaan aikoja, tulee tarvittava käsittely tehdä aina käsin. Kuitenkaan tässä projektissa ei kyetty käyttämään edellä mainittua suositusta vaan jouduttiin tekemään oma ratkaisu käsittelylle.

Sovellus käsittelee kellonaikoja ActionFilter Ja ModelBinder -laajennuksilla, jotka lähellä esitystasoa muokkaavat kellonajat vastaamaan käyttäjän aikavyöhykettä. Sovelluksen tietokanta jäi Suomen aikavyöhykkeelle, jotta rinnalla toimivat ratkaisut voidaan jättää koskemattomiksi. Kuitenkin toteutus tehtiin niin, että tulevaisuudessa on mahdollista siirtyä UTC-tietokantaan. Sovelluksen aikavyöhykekäsittelyä pyrittiin laajennuksilla automatisoimaan, mutta siinä ei täysin onnistuttu. Lopullisessa ratkaisussa suurimman osan sovelluksen toiminnallisuuksista hoitaa automatisoivat laajennukset, mutta on tiettyjä erikoistapauksia, joissa joudutaan käsin kääntämään kellonaikoja. Lopullinen ratkaisu saatiin toteutettua ja asetettua käyttöön.

Asiasanat
Aikavyöhyke, .NET Framework

Sisällys

1 Johdanto

Fatman Oy tarjoaa SaaS–ohjelmistopaketteja. Sovellukset ovat kiinteistöhallinnan työkaluja. Fatman Oy:n sovelluksista kootaan asiakkaille räätälöityjä ratkaisuja, jotka kehittävät asiakkaiden kiinteistöhallintoa. Sovellukset tarjoavat ratkaisuja muun muassa kiinteistön toiminnan ohjaukseen, kirjanpitoon, laskutukseen, raportointiin ja budjetoinnin suunnitteluun.

Opinnäytetyöprojekti koskee Fatman Oy:n uusinta tuotetta ”Framework”. Frame on yrityksen uusin tuote. Sovellus on toteutettu ”.NET MVC” -sovelluskehyksellä. Toistaiseksi Frame sisältää kiinteistöntoiminnan ohjauksen ja siihen liittyvän raportoinnin. Frame on tähän asti muiden yrityksen tuotteiden tavoin ollut sidonnainen järjestelmän aikavyöhykkeeseen.

Palvelin, jota sovellukset ovat tähän asti käyttäneet, on Suomen aikavyöhykkeellä. Asiakkaat ovat kaikki toistaiseksi olleet Suomen aikavyöhykkeen sisällä, joten ei ole ollut ongelmia. Nyt kun sovellusta on myyty Norjaan ja yritys harkitsee myynnin suuntaamista Euroopan markkinoille, yritykselle on syntynyt tarve globalisoida Frame-sovellus.

Frame käyttää järjestelmän aikaa, joten väliaikaisesti norjalaisille on perustettu oma palvelin, jonka aika on asetettu Norjan aikavyöhykkeeseen. Projektin on tarkoitus mahdollistaa Framen toiminta järjestelmän aikavyöhykkeestä riippumattomaksi, jotta jokaiselle aikavyöhykkeelle ei tarvitse asettaa erikseen omaa palvelinta. Lukuisat palvelimet aiheuttavat lisäkustannuksia ja tekevät ylläpidosta ja päivityksistä erittäin työlästä.

Opinnäytetyössä vastasin ratkaisujen selvityksestä ja valitsemisesta, sekä toteutuksen suunnittelusta. Ratkaisuvalintaan käytin myös Fatman Oy:n ohjelmistokehittäjien aikaa. Yhdessä keskusteltiin, jokaisen ratkaisun hyödyistä ja haitoista, mutta lopullinen vastuu päätöksestä jäi opinnäytetyön tekijälle.

2 Aikavyöhykkeet .NET-sovelluksissa

Projektin yhteydessä selvitettiin olemassa olevia ratkaisuja, joilla aikavyöhykkeet käsitellään .NET-viitekehys sovelluksissa. Kun etsitään omaa ratkaisua, on hyvä tiedostaa niin sanottu suositeltu käytäntö (”best practice”). Tyypillisesti siihen on hyviä argumentteja, miksi käytäntöä suositellaan. Käydään lyhyesti läpi selvitystyön aikana havaittu toimintatapa. Syyt, miksi toimintatapaa ei valittu, käsitellään myöhemmin tekstissä (Ratkaisun valinta 3.4).

”*When you are coding and desire to store current time represented as universal time, avoid calling DateTime.Now() followed by a conversion to universal time. Instead, call the DateTime.UtcNow function directly.*” - Microsoft Developer Network (2)

Microsoft Developer Networkin (MSDN) omilla sivuilla painotetaan, että tallentaessa aika arvoja tulisi ne aina tallentaa UTC-ajassa. UTC-aikaa suositellaan käytettävän, sillä laskenta ja vertailu aika-arvoille saattaa käyttäytyä odottamattomasti. Sivulla myös ehdotetaan, että jos käsitellään arvoja, jotka ovat tietyssä aikavyöhykkeessä, tulisi aina arvon mukana kantaa siirrosarvoa, joka kertoo kellonajan eron UTC aikaan. (Microsoft Developer Network (2))

”Always, always, ALWAYS store in UTC and display in the user's preferred or explicitly specified timezone.” - Stack exchange – Best storing practices

Sama dialogi UTC-ajan hyödyistä toistuu myös ”Stack exchange” -lähteessä. Lähteessä myös keskustellaan, kuinka käyttäjien tulisi antaa kellonaika-arvoja. Suositellaan kalenterivaraustyyppistä valintaa, jolloin annetaan sovelluksen päätellä oikea UTC-aika sen sijaan että sovellus ottaa tekstisyötteen, joka muutetaan aika arvoksi. (Stack exchange – Best storing practices)

Suositellaan, että kun esitetään aikoja, tehdään jokin helppokäyttöinen funktio, jolla konvertoida aika. Helppokäyttöfunktiota käytetään aina käsin jokaisessa paikassa, jossa halutaan näyttää käyttäjälle aikaa. Muuten sovellus käsittelee aina ja toimii UTC-ajassa. Manuaalisella käsittelyllä pyritään varmistumaan ohjelman oikeasta toiminnasta. Kun käsittelee aikoja, jotka ovat jollain aikavyöhykkeellä, on riski, että laskennat eivät toimi niin kuin niiden pitäisi. (Stack Overflow (8))

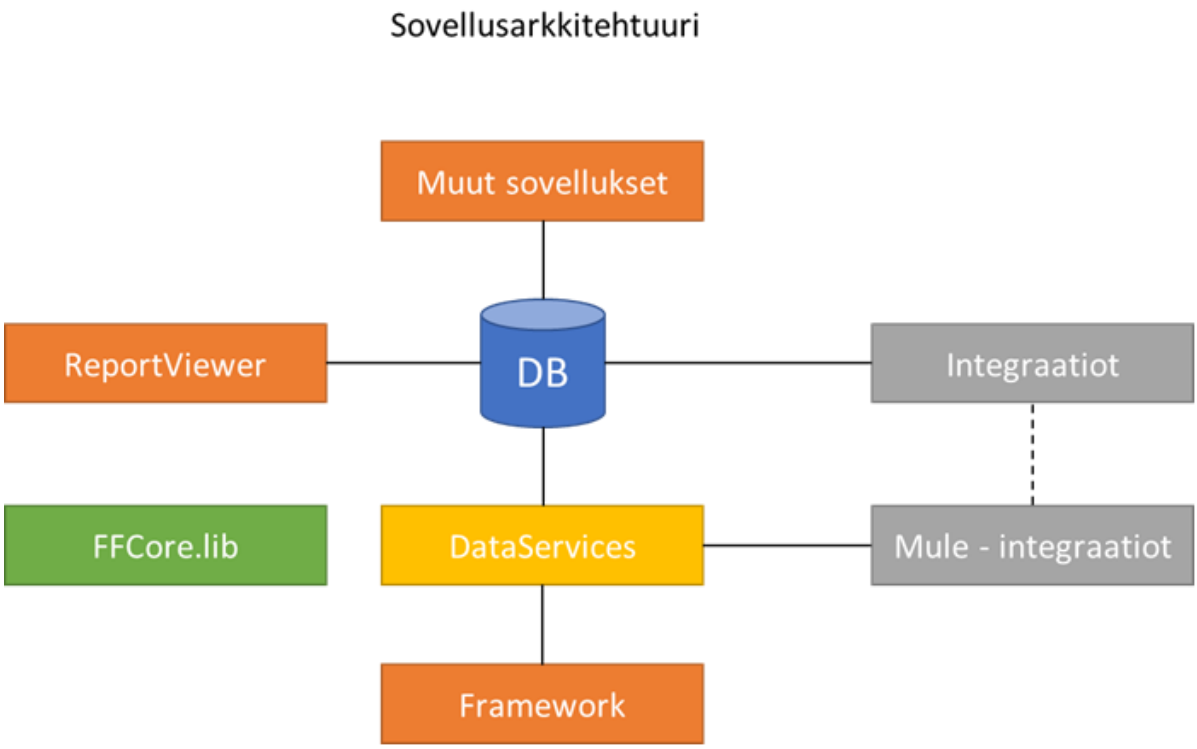
Kokonaisuudessaan selvitystyö osoitti, että suositeltavin tapa käsitellä aikavyöhykkeitä on aina tallentaa aika-arvot UTC-aikaan. Kun halutaan esittää aikaa, tulisi se muuntaa käsin UTC-ajasta käyttäjän aikaan. Kun suoritetaan laskentaa kellonajoilla, mikäli kellonaika on aikavyöhykkeellä, jossa on talvi- ja kesäaika, siirtymä kesä ja talviajan välillä saattaa aiheuttaa, että laskenta tai vertailu menee väärin, jos lasketa osuu sopivasti siirtymä kohdalle. Lähteissä painotettiin, että on hyvä käytäntö sopia yksi tietty aikavyöhyke tietokannalle. Tietokannalle loogisin aikavyöhyke on UTC, sillä se ei koskaan siirry kesä- tai talviaikaan ja ohjelmakoodi osaa tuottaa suoraan UTC-aikaa esimerkiksi aikaleimoja luodessa.

3 Selvitystyö

Ensimmäisessä kappaleessa (3.1 Sovellusympäristön esittely) esitellään projektiin liittyviä osia. Vaatimusmäärittely-kappaleessa (3.2) tarkennetaan projektin tarkoitus ja toivottu lopputulos. Sen jälkeen esitellään löytyneet ratkaisut aikavyöhykeongelmaan ja pohditaan niiden vahvuuksia ja heikkouksia. Esittelyn jälkeen vertaillaan ratkaisuja keskenään ja esitellään valittu ratkaisu sekä argumentoidaan miksi.

3.1 Sovellusympäristön esittely

Frame-sovellukseen liittyy useita sovelluksia, joiden tekniset rajoitukset ja mahdollisuudet tulee ottaa huomioon ratkaisun valinnassa (liite 1). Käydään lyhyesti läpi liitteessä esitetyt osat ja niiden vaikutusalueet. Suunnittelu-kappaleissa (3.3 - 3.4) käydään läpi tarkemmin mahdollisten ratkaisujen vaikutusalueet.



(Liite 1, Sovellusarkkitehtuuri)

Frame-sovelluksen viitekehys on .NET MVC Framework versio 4.5.2. Frame-sovellus on toteutettu noudattaen SOA-(service-oriented architecture) ja SaaS-(Software as a Service) ajattelumallia. Sovellus on koodattu modulaarisesti skaalattavaksi. Framework käyttää tietokannan käsittelyyn ainoastaan DataServices rajapintapalveluita.

FFCore on itse kirjoitettu staattinen kirjasto, joka sisältää luokkia ja funktioita. FFCore:n funktioita ja luokkia käytetään DataServices-, Framework- sekä ”ReportViewer”-projekteissa. FFCore mahdollistaa yleiskäyttöisten funktioiden jakamisen sovellusten välillä.

DataServices on palvelimella toimiva tietokantarajapinta. DataServices on toteutettu ”.NET MVC Framework” -viitekehyksellä ja vastaa kokonaan ”Frame”-sovelluksen tietokantakäsittelystä. DataServices toteuttaa palvelurajapintaa SOA-arkkitehtuurissa.

ReportViewer-projekti on toteutettu käyttäen Microsoftin tarjoamaa SSRS-(SQL Server Reporting Services) raporttien generointisovellusta. Sovelluksen viitekehyksenä toimii Microsoftin .NET MVC Framework. Projekti kutsuu tietokantaa suoraan, joten suuret tietokanta muutokset vaikuttavat sovellukseen.

Tietokanta on kaaviossa (liite 1) kuvattu tekstillä DB. Käytössä on SQL-tietokantapalvelin, joka on tyypillisesti uusin julkaistu versio. Poikkeuksena sääntöön muista irrotettu tietokantapalvelin on versiossa 2005, joka tullaan päivittämään lähitulevaisuudessa tulevaisuudessa, mutta tuskin tämän projektin aikana. Samaa tietokantaa käyttää Framework-sovelluksen lisäksi useat muut sovellukset. Koska useat sovellukset käyttävät samaa palvelinta, muutokset tietokantojen sisältöön vaikuttavat laajalti.

Muut sovellukset-laatikolla kuvataan kaikkia vanhempia Fatman Oy:n sovelluksia, jotka käyttävät samaa tietokantaa. Sillä että sovellukset käyttävät samaa tietokantaa, on mahdollistettu vanhojen ja uusien sovellusten sekakäyttö. Sovelluksia on koodattu erinäisillä kieliä, kuten PHP ja Delphi. Erikielisyys tulee ottaa huomioon ratkaisua valitessa, sillä muutokset vanhempiin sovelluksiin saattavat olla yllättävän työläitä.

Integraatiot asiakkaiden muihin sovelluksiin on toteutettu erinäisin tavoin. Vanhemmat integraatiot sisältävät erilaisia SQL-funktioita suoraan kantaan ja uudemmat käyttävät Mulen kautta dataservices-projektia. Tietokantaan tehdyt muutokset saattavat aiheuttaa korjausvaatimuksia näihin toteutuksiin, joten muutokset aiheuttaisivat paljon testausta.

3.2 Vaatimusmäärittely

Tavoiteltu tila, johon opinnäytetyö projektissa pyritään, mahdollistaa Frame-sovelluksen käytön luotettavasti kaikilla aikavyöhykkeillä. Projektin jälkeen tulisi olla mahdollista myydä sovellusta maailmanlaajuisesti. Sovelluksessa esitetään päivämääriä ja kellonaikoja sekä mahdollistetaan, se että käyttäjä syöttää sekä päivämääriä että kellonaikoja ja ne tallentuvat oikein. Tiedon syötön ja lukemisen lisäksi sovelluksessa on laskentaa ja hälytyksiä, joiden tulee ottaa huomioon aikavyöhykkeet.

Selvitysvaiheessa pyritään löytämään ratkaisu, joka käsittelee aikavyöhykkeen käsittelyn mahdollisimman luotettavasti ja vähällä koodauksella. Ratkaisun toteutuksen jälkeen toivotaan, että vanhat sovellukset toimivat yhdessä Frame-sovelluksen kanssa, jos mahdollista. Ei kuitenkaan tarvitse toteuttaa aikavyöhykekäsittelyä muille kuin Frame-sovellukselle. Hyväksytään myös, että vanhat sovellukset tulevat toimimaan oikein vain, jos sovellus toimii Suomen aikavyöhykkeellä.

Tällä hetkellä kaikki sovellukset ja tietokanta käyttävät järjestelmän aikaa, joka on määritetty olevan Suomen aikavyöhykkeellä (FLE Standard Time). Käyttäjien syötteet tallennetaan kantaan ilman muutoksia arvoon. Kellonajat ja päivämäärät esitetään käyttäjille kuin ne ovat tietokannassa ilman aikavyöhykekäsittelyä. Datatyyppi, jota käytetään Frame- ja DataServices-sovelluksissa on C#-koodin Datetime.cs, ja tietokannassa käytetyt tietotyypit ovat SQL:n Datetime, Datetime2 ja smalldate. Toistaiseksi jokainen uusi asiakas uudella aikavyöhykkeellä vaatii, että perustetaan uusi palvelin, joka on kyseisellä aikavyöhykkeellä.

3.3 Ratkaisuvaihtoehdot

Seuraavaksi käydään läpi varteenotettavimmat ratkaisuvaihtoehdot ja niiden heikkouksia ja vahvuuksia. Ratkaisuvaihtoehdot syntyivät perehtymällä olemassa oleviin ratkaisuihin, joita oli dokumentoitu internetiin sekä yrityksen sisäisiin keskusteluihin. Kokouksessa käsiteltiin kokonaisuudessaan kuusi ratkaisuvaihtoehtoa: UTC-tietokanta, tietokannan tietotyyppilaajennus, GUI-käsittely, tietokantakohtainen aikavyöhyke, datasource ja display-käsittely (Ratkaisun valintakokous 19.9.2016). Jätän osan ratkaisuista pois tästä dokumentista, sillä ne ovat varsin tilannekohtaisia korjauksia. Poisjääneet ratkaisut myös vaativat syvää ymmärrystä projektin sovellusympäristöstä, jotta kyetään ymmärtämään, miksi ja miten ne toimivat. Ratkaisut, jotka esitellään, on toistettavissa muillakin sovelluksilla. Tosin kuten kappaleessa kaksi todettiin, on suositeltavaa käyttää MSDN:n käytäntöjä.

3.3.1 UTC-Tietokanta

”UTC Tietokanta” -ratkaisu tarkoittaa, että kaikki tietokantapäivämäärät ja kellonaika-arvot ovat UTC-aikaa. Ratkaisu vaatii, että kaikki tietokanta arvot käännetään UTC-aikavyöhykettä vastaavaan arvoon. Lisätään tarvittavat käsittelyt Frameen, jotta ajat näytetään oikein käyttäjille. Irrotetaan Frame palvelimen lokaalista aikavyöhykkeestä, jolloin ohjelman sisäisesti aina käytetään UTC-aikaa. Kun sovellus on UTC-ajassa, voidaan käyttää MVC .NET-viitekehyksen omia työkaluja aikavyöhykkeen esityksen hallinnoimiseen.

UTC tietokanta on suositeltu ratkaisu (Microsoft Developer Network (2)) aikavyöhykeriippuvaisille ohjelmistoille. Ratkaisu olisi selkeä alusta loppuun, kun se noudattaa MSDN:n suositeltua käytäntöä. Viitekehyksen tarjoamat ratkaisut ovat laajalti testattuja ja valmiiksi koodattuja palikoita, mikä vähentää omaa työtä ja vastuuta.

Ratkaisu aiheuttaa kuitenkin vanhempien ohjelmien aikakenttien vääristymisen, ellei sitä varten tehdä vanhoihin ohjelmistoihin korjauksia. Muut sovellukset myös tallentaisivat ajan väärin, kun käyttäjä syöttää aikaa tietämättä, että syötteen tulisi olla UTC-aikavyöhykettä. Kaikki raportit pitää korjata tämän ratkaisun yhteydessä, ja on todennäköistä, että raporttien viitekehyksen työkaluilla ei olisi mitään järkeviä valmiita malleja vaan joudutaan raporttikohtaisesti käsin kääntämään kaikki esitettävät ja vertailtavat arvot.

3.3.2 Tietokantakohtainen aikavyöhyke

”Tietokantakohtainen aikavyöhyke” -ratkaisussa hyväksytään, että tietokanta arvoja ei toistaiseksi voida muuttaa UTC-aikaa. Mutta sovellusten sisällä siirrytään käyttämään UTC-aikaa, kun tarvitaan syystä tai toisesta järjestelmän aikaa, sillä asiakkuuksilla on omat tietokannat. Jokaiselle asiakastietokannalle voidaan määritellä aikavyöhyke tietokantakohtaisesti. Tallennettaessa järjestelmän aikaa tulee kääntää aika asiakkaan ajaksi ennen tallennusta. Suositeltavaa on ilmoittaa käyttäjälle, missä aikavyöhykkeessä ajat esitetään ja tallennetaan.

Ratkaisun ansiosta sovelluksissa voitaisiin esittää ajat, kuten tietokannassa ne on tallennettu ja laskennat toimivat kuten ennenkin. Syötteet päivämäärille ja kellonajoille voidaan jättää käsittelemättä, sillä voidaan aina olettaa, että ne ovat asiakkaan aikavyöhykettä. Kun tietokanta-arvot ovat käsittelemättä luettavaa, voidaan esittää nykyisiä raportteja ilman muutoksia koodiin. Asiakkaat, jotka ovat Suomen aikavyöhykkeellä, voisivat jatkaa vanhojen softien käyttöä Frameworkin rinnalla. Ratkaisu mahdollistaa tietokantojen siirtämisen UTC-aikaan tulevaisuudessa, ja vieläpä asiakaskohtaisesti.

Määriteltäessä tietokanta yhteen aikavyöhykkeeseen sidotaan kaikki käyttäjät samaan aikavyöhykkeeseen, jolloin ratkaisu ei suoraan korjaa ongelmaa asiakkaille, joilla on kiinteistöjä usealla aikavyöhykkeellä. Ylläpito kasvaa haasteelliseksi, kun asiakaskannat ovat eri aikavyöhykkeillä. Ongelmaselvitykset laskentoihin ja vertailuihin, jotka ottavat päiviä tai kellonaikoja huomioon, muuttuvat entistä kompleksisemmaksi. Ratkaisu vaatii, että sovelluksen sisällä ei saa enää kutsua ”Datetime.Now”-funktiota. Funktiokutsun esto on haasteellista, mutta mahdollista esimerkiksi Roslyn analyzer tai ReSharpper lisäosalla. Vaikka funktiokutsun voi estää, tulee ohjelmoijien tulevaisuudessa ottaa huomioon laskennoissa tietokannankannan aikavyöhyke. On vaikea varmentua, että laskentalogiikka on oikein kaikille asiakkuuksille ilman että kirjoittaa kattavat testit, joka vie aikaa. Täytyy vain toivoa, että koodin kirjoittaja on ottanut aikavyöhykkeen huomioon oikein, sillä on mahdotonta tehdä koodin sisäisiä tarkastuksia, joilla voisi vartioida aikavyöhykkeiden oikeaa käyttöä.

3.3.3 GUI-käsittely

”GUI-käsittely”-ratkaisussa pyritään viemään aikavyöhykekäsittely esitystasolle ja jätetään tietokanta- ja dataservice-muutokset mahdollisimman pieniksi. Kun sovellus esittää käyttäjälle aikaa, se konvertoidaan käyttäjälle sopivaksi. Käyttäjän syöttäessä aikaa käsitellään se Suomen aikavyöhykkeelle. Kun käyttäjä syöttää päivämäärän ja/tai kellonajan, voidaan käyttää laajennettuna jo olemassa olevaa ”FFDateTimeModelBinder” luokkaa. Luokka on itse kirjoitettu luokan sitoja, jota käytetään käsittelemään esimäärättyä päivämääräformaattia, kun luetaan käyttäjän syötettä. Esityksen ratkaisun automatisointia voi toteuttaa usealla eri tavalla, joista listaan neljä:

- ”Property Annotation” -tekniikka, joka laajentaa asetettua luokan ominaisuutta funktiolla ”ToDisplayValue”. Kyseistä funktiota voidaan tällöin kutsua esitystasolla, kun halutaan esittää päivämääriä tai kellonaikoja käyttäjän ajassa.

- ”ActionFilter” -laajennus, jonka viitekehys suorittaa jokaisen funktion jälkeen. Laajennus käsittelee metodeiden palautusarvot vastaamaan käyttäjän aikavyöhykettä.

- ”Moment.js” javascript-kirjasto, jolla voidaan kääntää date- ja datetime-arvoja käyttäjän aikavyöhykkeelle. Käyttäen ”class”-määritteitä voidaan automatisoida käsittelyä.

- ”DateFuntions” -luokka, joka sisältää staattiset käsittelyfunktiot, joita ohjelmoija kutsuu käsin arvokohtaisesti esitystasolla, kun on tarve näyttää käyttäjälle jotain.

Ratkaisu mahdollistaa käyttäjäkohtaisen aikavyöhykevalinnan. Tietokantaan ei tarvitse tehdä tässä ratkaisussa konversiota jolloin kaikki vanhat sovellukset toimivat kuten ennen. GUI automaatiotason mukaan ratkaisu voi olla hyvinkin pienitöinen suhteessa muihin ratkaisuihin.

Ohjelmointikokemukseni perusteella sanon, että varmistuminen automatisoinnin virheettömästä toiminnasta on haasteellista. Automaatioratkaisut eivät ota huomioon raportteja eikä integraatioita. Raporteissa ja integraatioissa joudutaan ”GUI-käsittely”-ratkaisussa tekemään aikakäsittely tapauskohtaisesti käsin, mikä vaatii ohjelmoijalta ymmärrystä ja muistamista.

3.4 Ratkaisun valinta

Ratkaisu valittiin kokouksessa (Ratkaisun valintakokous 19.9.2016). Kokouksessa käytiin läpi eri vaihtoehdot ja verrattiin niitä vaatimusmäärittelyssä esiteltyihin tarpeisiin. Ratkaisuksi valikoitui ”GUI-käsittely”-ratkaisu. Valintaa ei kuitenkaan tehty varauksetta. Kokouksessa tiedostettiin ratkaisun riskit ja rajoitteet todeten sen olevan mieluisin.

3.4.1 UTC-tietokanta vs. GUI-käsittely

”UTC-tietokanta”-ratkaisu tiedostettiin olevan useiden lähteiden perusteella suositeltavin ratkaisu. Ratkaisu on luotettava, sillä tietokanta-arvot olisivat puhtaasti aikavyöhykeriippumattomia. Kun tietokantaan tallennetaan aikavyöhykkeellisiä aikoja, aikavyöhykkeiden väliset konversiot ovat riski. Osa aikavyöhykkeistä siirtyy talvi- ja kesäaikaan aiheuttaen niin sanottuja ”Ambiguous time”-arvoja, eli kellonaikoja, joita ei ole olemassa, ja kellonaikoja, jotka toistuvat useasti saman vuoden aikana. Kyseiset epämääräisyydet aiheuttavat luonnollisesti ohjelmallista epävarmuutta. ”UTC tietokanta”-ratkaisun mahdollistamat valmiit työkalut olisivat kolmannen osapuolen työkaluja, joten niitä voidaan luotettavasti käyttää ja ongelmatilanteisiin löytyy apua netistä. Ratkaisu olisi myös uusille ohjelmoijille helppo sisäistää toisin kuin lopulta valittu ”GUI-käsittely”-ratkaisu. (Ratkaisun valintakokous 19.9.2016)

”UTC tietokanta” -ratkaisua ei kuitenkaan valittu, koska työtaakka vanhempiin sovelluksiin ja integraatioihin Frame-sovelluksen muutosten lisäksi oli ylitsepääsemätön. Aikaisemmin mainittujen valmiiden työkalujen opiskeluun ja käyttöön olisi kulunut aikaa. Todettiin myös mahdolliseksi, että työkalut eivät ratkaisisi kaikkia tilanteita ja jouduttaisin itse kuitenkin koodaamaan omia aika-arvo käsittelijöitä. Valmiit ratkaisut olisivat rajoittaneet, miten täyttölomakkeet koodataan rikkoen useat nykyiset näkymät, kuten ”laitelista”, jossa valitaan laite JavaSriptillä piirretystä resurssipuusta. (Ratkaisun valintakokous 19.9.2016)

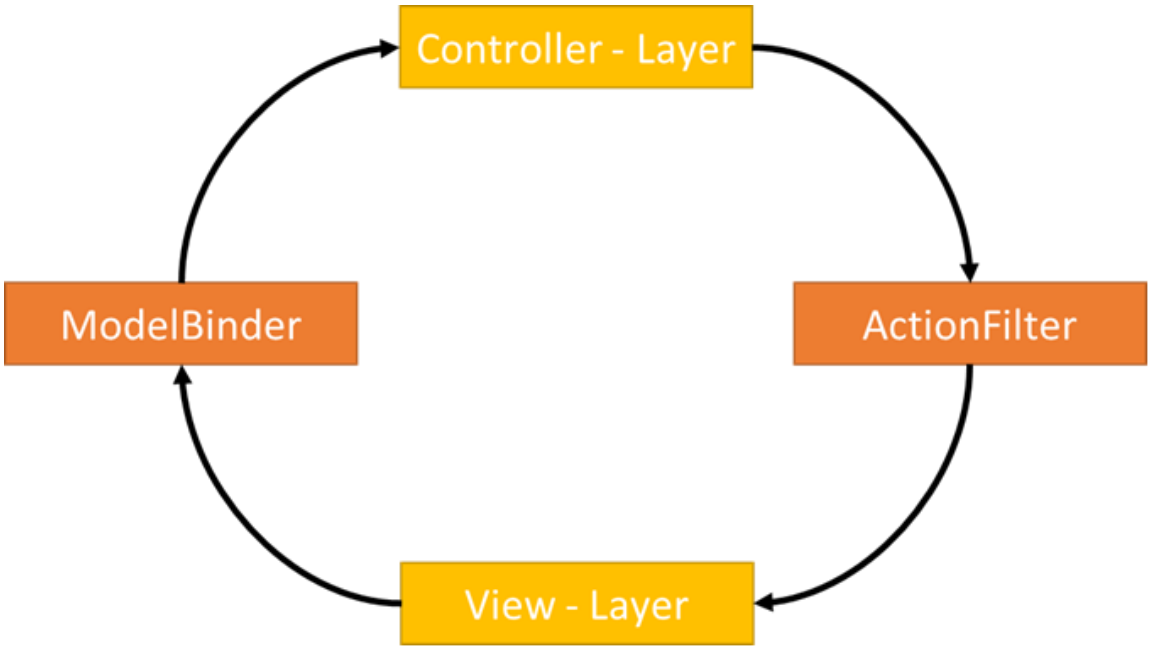
3.4.2 Tietokantakohtainen aikavyöhyke vs. GUI-käsittely

”Tietokantakohtainen aikavyöhyke” -ratkaisu sisälsi joitakin vahvuuksia ”GUI-käsittely”-ratkaisuun verrattuna. Jos asiakastietokannat olisi asetettu eri aikavyöhykkeille, raportit, jotka tyypillisesti esittävät dataa suoraan kannasta, olisivat toimineet ilman mitään muutoksia. Riskit aikavyöhykekäsittelystä olisivat koskeneet ainoastaan Suomen ulkopuolella olevia kantoja, kun käytännössä Suomen aikavyöhykkeellä toimiville asiakkaille ei olisi tehty lähes mitään. Ainoastaan olisi pitänyt tiedostaa, että vain Frame-sovellus toimii Suomen aikavyöhykkeen ulkopuolella, mikä nyt ei varsinaisesti eroa ”GUI-käsittely”-ratkaisusta. Myöskin ”Tietokantakohtainen aikavyöhyke” -ratkaisun mahdollisuus esittää sovelluksessa kaikki kanta-arvot käsittelemättä ja ottaa syötteet käsittelemättä vähentäisi ratkaisuun liittyviä riskejä ja kannan virheellisyyttä huomattavasti. (Ratkaisun valintakokous 19.9.2016)

”Tietokantakohtainen aikavyöhyke” -ratkaisu kuitenkin jätettiin tekemättä, sillä pitkällä aikavälillä nähtiin ikäviä haasteita. Kun tietokannat ovat eri aikavyöhykkeillä, ylläpidosta tulee varsin haasteellista. Esimerkiksi tuotannon ongelmaselvityksissä ja korjaustöissä kehittäjä joutuisi ottamaan myös huomioon eri kantojen aikavyöhykkeet. Sovellukset käyttävät myös yhteistietokantoja, kuten tapahtumaloki ja autentikointi -kannat, jotka olisivat olleet Suomen ajassa. Ratkaisu olisi myös sitonut asiakkaat yhteen aikavyöhykkeeseen, mikä olisi pitkällä aikavälillä kuitenkin tullut vastaan, esimerkiksi Euroopan laajuisten asiakkaiden kanssa. Jos tähän ratkaisuun, olisi lähdetetty olisi pitänyt olla valmis hyväksymään, että jos joskus tulevaisuudessa halutaan vaihtaa ratkaisua, kanta-arvojen konversio olisi ollut erittäin riskialtis projekti. Lisäksi varmuus, että sovellus ei käytä järjestelmän aikaa missään, olisi ollut varsin vaikea todentaa. (Ratkaisun valintakokous 19.9.2016)

3.4.3 Tarkennusta lopulliseen ratkaisuun

Keskustelujen jälkeen päädyttiin ”GUI-käsittely”-ratkaisuun. Valinnan yhteydessä sovittiin, että esitystason automatisointia tutkitaan vielä lisää ja siitä sovitaan erillisessä kokouksessa (Ratkaisun valintakokous 19.9.2016). Automatisointiratkaisu, johon päädyttiin, oli ”ActionFiltter”-ratkaisu. Se on ratkaisu, joka käsittelee päivä- ja kellonaika-arvoja viewin ja controller actionin välillä (ks. liite 2). Kokouksessa todettiin ratkaisun olevan varsin suhteutettuna pienitöinen, ja kokousta varten tehty Proof of Concept (PoC) herätti positiivisia mielipiteitä ratkaisusta. Kokouksessa painotettiin vaatimusmäärittelyssä sovittua asiaa, että työmäärät tulee olla mahdollisimman pienet suhteessa riskeihin. Haluttiin eroon siitä, että joudutaan käsin esitystasolla muuntamaan aikaa, koska pelättiin, että kehittäjä saattaa unohtaa käsittelyn, joka tulevaisuudessa voisi aiheuttaa inhottavia ongelmaselvitys tehtäviä. Toinen korkeatasoinen automatisointivaihtoehto ”Moment.js” ei valikoitunut sillä enemmistön kokouksessa olleiden kehittäjien mielestä Javascript-toteutus ei ole tarpeeksi luotettava. (Esitystasoratkaisun valintakokous 30.9.2016)



(Liite 2, Ratkaisun arkkitehtuuri)

Valittu ”GUI käsittely”-ratkaisu (liite 2) käsittelee aika-arvot ”Controller”- ja ”View” -tasojen välillä. Kun aika-arvot lähetetään ”View”-tasoon, ne käsitellään käyttäjän aikavyöhykkeelle ”FFDisplayDataTimeFiltter”-luokan, käsittely tapahtuu kaikille funktioille, joilla on palautusarvoja. Luokka käy palautettavan olion kaikki arvot läpi etsien aika-arvoja. Arvoa käsitellään aika-arvolle asetetun datatyypin määrityksen mukaan. Aika-arvo voi olla joko päivämäärä tai kellonaika. Jos arvo on kellonaika, tehdään muunnos käyttäjän aikavyöhykkeelle. Jos arvo on päivämäärä, ei tehdä muutosta. Kun ”View”-taso lähettää aika-arvoja

”Controller”-tasolle, konversio tehdään päinvastoin. ”ModelBinderExteder”-luokka kääntää lähetetyt kellonajat käyttäjän aikavyöhykkeeltä Suomen aikavyöhykkeelle käyttäen samoja päivämäärä ja kellonaikadatatyyppimäärityksiä.

Lopulliseksi ratkaisuksi valikoitui siis, että sovellus käsittelee päivä- ja kellonaika-arvot vastaamaan käyttäjän aikavyöhykettä Controller- ja View-tasojen välillä. Ratkaisu tuntui varsin hyvältä, sillä se oli varsin pienitöinen, mutta kuitenkin voitiin olla varmoja sen mahdollisuudesta toteuttaa sovelluksen tarve käsitellä aikoja. Arvioitiin että ratkaisussa saattaa tulla toteutusvaiheessa vielä haasteita, mutta uskottiin, että ne saadaan käsiteltyä. (Esitystasoratkaisun valintakokous 30.9.2016)

4 Toteutus

Ratkaisu toteutettiin kahden viikon sprintillä, johon osallistui lähes kaikki Fatman Oy:n sovelluskehittäjät. Koodaus tehtiin erilliseen versiohallinnan kehityshaaraan siltä varalta, että koko konversiota ei ehditä tekemään ennen seuraavaa versiopäivitystä tai esiintyy ongelma, joka vaatii enemmän aikaa. Konversiotyöhön kuului eri moduuleissa data-attribuuttien määritys kaikille luokille, joita käytettiin Frame-sovelluksessa, ja raporttien korjaus käsin. Toteutus onnistui ilman suurempia ongelmia, mutta tiettyjä mainitsemisen arvoisia ohjelmointivirheitä esiintyi, joille kehitettiin ratkaisut.

- Sovelluksen sisäiset käyttäjän uudelleenohjaukset aiheuttivat erikoistapauksissa ylimääräisiä käsittelyjä aika-arvoille.

- Sessioon tallennetut päivämäärä- ja aika-arvot konvertoituivat joka kerta, kun kontrollerimetodi palautti viittauksen esitystasolle kyseiseen sessiomuuttujaan. Käyttäjälle tämä esiintyi niin, että jokaisella sivulatauksella sessioon tallennettu arvo muuttui aikavyöhyke-eron verran.

- Sovelluksen lähettämät aikoja sisältävät sähköpostit eivät automaattisesti käsitelleet aikavyöhykkeitä, sillä sähköpostilogiikka toteutettiin metodin sisällä.

- Sovelluksessa palautettiin joskus päivämäärä tai kellonaika merkkijonona, ne jäivät luonnollisesti ActionFiltter -laajennukselta käsittelemättä.

- Joskus sovelluksessa palautettiin sama DateTime-muuttujaviittaus useaan kertaan saman palautuksen aikana aiheuttaen sen konversion myös tapahtuvan useasti samalle arvolle. Kyseessä oli vanhempia metodeita, joita oli muutettu useasti muuttaen toteutuksen hieman spagetti koodiksi.

Toistuvista konversioista päästiin eroon, kun kirjoitettiin oma luokka kellonaikoja ja päivämääriä varten. Luokan nimi on DateTimeExtended, joka sisältää arvot Datetime ja TimezoneInfo. Luokalla on omat metodit käsittelyä varten. Ne varmistavat, että aika- ja aikavyöhykearvot pysyvät eheinä. Luokkaa ei kuitenkaan laitettu käyttämään kaikkialla, vaan sitä käytettiin ongelmatilanteissa. Kaikkien datetime-arvojen muutos DatetimeExtended-muotoon olisi vaatinut lisää aikaa toteutukseen ja testaukseen. Sähköpostien osalta jouduttiin käsin tekemään konversio käyttäjän aikaan, mikä tosin oli hyödyllistä. Oli hyvä miettiä, sähköposteissa millä aikavyöhykkeellä tulisi esimerkiksi ilmoitusajat näyttää.

Useita viittauksia varten kehitettiin datapankki, joka seuraa viittauksia konversioiden yhteydessä. Datapankilla saatiin kaikki paikat kiinni, joissa samaa arvoa palautettiin useasti, mikä kehitti koodia myös parempaan suuntaan. Suorituskykysyistä Datapankki-seuranta on käytössä vain, kun sovellus käännetään kehittäjän lokaalille koneelle.

On hieman harmi, että käännöksiä ei saatu täysin automatisoitua. Edellä mainitut ongelmat saattavat esiintyä myöhemmin uudelleen, jos ja kun ohjelmistokehittäjä unohtaa käyttää ”Extended”-luokkaa tai palauttaa ajan muuttamattomasti sähköpostissa. Kuitenkin kokonaisuudessaan toteutus onnistui hyvin ja sovellus saatiin toimimaan aikavyöhykeriippumattomasti.

5 Testaus

Yrityksessä toteutetut toiminnallisuudet testataan tekijän ja katselmoijan puolesta. Kumpikin pyrkii testaamaan kattavasti toiminnallisuuden eri muuttujilla. Katselmoija on toinen ohjelmistokehittäjä, joka ei ole ollut toteuttamassa toiminnallisuutta. Katselmoija lukee läpi uuden lähdekoodin ja testaa toiminnallisuuden kykyjensä mukaan. Kun katselmoija on hyväksynyt toteutuksen, se siirtyy demo palvelimelle, jossa toteutus altistuu lisätestaukselle projektipäälliköiden toimesta.

Ratkaisun toteutuksen yhteydessä tehtiin paljon tavallisuudesta poikkeavaa lokaalia testausta, jossa selvitettiin kattavasti mahdollisia ohjelmointivirheitä. Kehittäjät löysivät toteutuskappaleessa (kappale 4) mainitut haasteet. Sovellukseen on toteutettu kokoelma yksikkötestejä. Yksikkötestit pyrkivät varmistamaan, että tulevaisuuden muutokset ratkaisun koodiin eivät riko perustoiminnallisuutta.

Lisäksi toteutuksen jälkeen tehtiin suunniteltu testaus, jossa seurattiin etenkin tunnettujen ohjelman ongelmakohtien toimintaa ratkaisun kanssa (Aikavyöhykekonversio-testaussuunnitelma). Testaussuunnitelmassa käytiin systemaattisesti läpi kaikki sovelluksen osat ja niiden toiminnallisuudet käsin. Testeissä luotettiin suorituksen kaatuvan, kun funktio yrittää palauttaa tyypittämättömän ”DateTime” -arvon. Kun ratkaisu läpäisi testaussuunnitelman, se asetettiin demo palvelimelle, jossa tehtiin ns. palvelintason testausta, ja sen jälkeen ratkaisu päivitettiin tuotantoon.

6 Ylläpito

Ylläpidollista ongelmaselvitystä tapahtuu Frame-sovellukseen liittyen aika-ajoin, sillä asiakkuuksien kokoonpanot ja asetukset vaihtelevat paljon. Tyypillisin ylläpidollinen työ on, kun ohjelmistokehittäjiltä jää joskus jokin asetusten summa huomioimatta, joka esiintyy virheenä sovelluksessa. Harvinaisempi mutta tapahtuva ylläpito työ on ongelmaselvitys, jossa sovelluksen käyttäjät ottavat yhteyttä hämmennyksestä sovelluksen toiminnassa.

Aikavyöhykeratkaisu aiheuttaa hieman lisää ylläpidollista taakkaa. Ylläpitotaakka kasvaa lähinnä niissä osioissa ohjelman toiminnassa, joista mainittiin aikaisemmin haasteina (Toteutus 3.5). Data-attribuutit tulee lisätä aika- ja päivämäärätiedoille kehitettäessä uutta koodia. Tosin se ei vaadi muistamista, sillä suoritus kaadetaan, jos kyseiset attribuutit puuttuvat. Haasteellisinta on muistaa koodissa ne paikat, joita ”ActionFilter”-ratkaisu ei käsittele automaattisesti.

Tietokanta on vieläkin täysin suomen ajassa, ja se saattaa aiheuttaa tulevaisuudessa hämmennystä kesä- ja talviaikojen kanssa, etenkin kun käyttäjän ja Suomen aikavyöhykkeellä siirrytään kyseisiin aikoihin eri hetkinä. Sovelluksen ottaessa huomioon käyttäjän ajan luonnollisesti raportit ja listaukset esiintyvät eri arvoilla eri käyttäjille, jos kyseisten käyttäjien aikavyöhyke-asetus on eri. Hämmennys saattaa esiintyä kasvavana määränä yhteydenottoja ylläpitoon. Yhteydenottoja pyritään lieventämään kertomalla aikavyöhyke-asetuksesta koulutuksissa. Oletuksina vanhoille asiakkaille asetus on Suomen aikavyöhykkeellä ja asiakkaan toiveesta voidaan myös piilottaa asetus sovituilta käyttäjä ryhmiltä.

7 Jatkokehitys

Vaikka ratkaisu toteutuksen jälkeen onkin toimiva, voitaisiin tulevaisuudessa vielä kehittää ratkaisua lisää. Käydään läpi toteutuksen yhteydessä ja sen jälkeen nousseita toiveita jatkokehitykselle.

- Tietokanta siirretään UTC-aikaan. Kun arvot tallennetaan UTC-aikaan, voidaan yksinkertaistaa ratkaisun rakennetta, jolloin lisätään toimintavarmuutta ja parannetaan suorituskykyä.

- ”FFDateTimeModelBinder” -luokaa voisi kehittää ottamaan huomioon niin sanotut ”Ambiguous time” -virheet. Kyseessä on virhetilanne, jossa käsitellään aikaa, jota ei ole käytännössä olemassa. Virheellistä aikaa esiintyy, kun aikavyöhykkeellä siirretään aikaa tunnilla eteenpäin tai kun aikaa siirretään tunnilla taaksepäin, se aiheuttaa sen, että sama kellonaika toistuu kahdesti. Virhekäsittely pitää suunnitella ottamaan huomioon kaikki mahdollisuudet ja antamaan käyttäjälle mahdollisuus korjata tilanne.

- Laskennat, jotka suoritetaan ohjelman toiminnasta, kuten SLA-laskenta, ovat vieläkin suomen ajassa. Jotta laskennoista tehtäisiin helpommin kehitettäviä, tulisi niihin kehittää jonkintyyppistä aikavyöhykelogiikkaa.

- On ollut keskustelua, että sovellus muutetaan Single Page-sovellukseksi. Jos sovelluksen esitystaso siirretään kyseiseen tekniikkaan, joudutaan uudelleen harkitsemaan ”Moment.js” -kirjastoa.

8 Projektin kokemukset

Kokonaisuudessaan koen, että projekti onnistui. Projektin yhteydessä saatiin toteutettua ratkaisu, joka tekee mitä pyydettiin. On kuitenkin harmillista, että syntynyt ratkaisu ei ole täydellinen ja en ole täysin tyytyväinen ratkaisun tuottamiin sivuvaikutuksiin. On siis todennäköistä, että ratkaisua kehitetään tai muutetaan tulevaisuudessa.

Projektissa olisi voinut kyseenalaistaa vaatimusmäärittelyä enemmän. Toive saada automatisoitua aikavyöhykekäsittely, vaikka teknisesti mahdollista osoittautui ongelmaherkemmäksi kuin suora käsittely aina kun esitellään arvoja. Pohdin, onko vielä jokin ratkaisu, jota en keksinyt ja olisinko keksinyt sen vielä syvällisemmällä pohtimisella. Dokumentointi projektin etenemisestä olisi varmaan pitänyt aloittaa paljon aikaisemmin, jotta tämän dokumentin kirjoitus olisi ollut helpompaa. Työ oli konkreettinen esimerkki ohjelmoinnin todellisuudesta. Joskus pitää tehdä kompromisseja kehityksen suhteen, jolloin ei voida ottaa niin sanottua suositeltua ratkaisua.

Kuitenkin olen tyytyväinen projektiin kokonaisuudessaan. Selvitykselle ja tutkimustyölle oli annettu hyvin aikaa. Toteutus oli tehokas ja onnistui toteuttamaan ratkaisun koko sovelluksen laajuisesti. Selvityksen aikana sain yritykseltä apua haastatteluiden ja kokousten muodossa aina kun tarvitsin. Projektin ansiosta koen, että kehityin ohjelmistokehittäjänä. Luin paljon mm. SQL-tietokannoista, tietokannan eheyssäännöistä, .NET-viitekehyksestä ja C#-koodista. Opin paljon myös Fatman Oy:n sovelluksista ja niiden välisistä riippuvuuksista.

Lähteet

Arkkitehtuuripalaveri 18.8.2016, Fatman Oy, Läsnä: J Kärnä, M Makkonen, M Walhroos ja V-M Nyfors

Devcurry - Custom Templates, Data Annotations and UI Hints in ASP.NET MVC. Luettavissa: http://www.devcurry.com/2013/04/custom-templates-in-aspnet-mvc.html. Luettu: 10.9.2016.

Esitystasoratkaisun valintakokous 30.9.2016, Fatman Oy, Läsnä: J Kärnä, M Makkonen, Mika Wennonen ja M Walhroos

Jani Kärnä – Aikavyöhyke konversio testaussuunnitelma

Microsoft Developer Network (1). Luettavissa: https://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinebuffer.item.aspx. Luettu: 2.9.2016.

Microsoft Developer Network (2). Luettavissa: https://msdn.microsoft.com/en-us/library/ms973825.aspx. Luettu: 13.9.2016.

Microsoft Developer Network (3). Luettavissa: https://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.datatype.aspx. Luettu: 4.9.2016.

Microsoft Developer Network (4). Luettavissa: https://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.displayformatattribute(v=vs.110).aspx. Luettu: 8.9.2016.

Microsoft Developer Network (5). Luettavissa: https://msdn.microsoft.com/en-us/library/system.web.mvc.html.editorextensions.editorfor(v=vs.118).aspx. Luettu: 11.9.2016.

Microsoft Developer Network (6). Luettavissa: https://msdn.microsoft.com/en-us/library/w86s7x04.aspx. Luettu: 7.9.2016.

Microsoft Developer Network (7). Luettavissa: https://www.asp.net/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/creating-a-more-complex-data-model-for-an-asp-net-mvc-application. Luettu: 2.9.2016.

Microsoft Developer Network (8). Luettavissa: https://www.asp.net/mvc/overview/older-versions-1/controllers-and-routing/understanding-action-filters-cs. Luettu: 5.9.2016.

Momentjs. Luettavissa: http://momentjs.com/timezone/. Luettu: 15.9.2016.

Projektin aloitus kokous 20.6.2016, Fatman Oy, Läsnä: J Kärnä, M Makkonen, Mika Wennonen, M Walhroos ja V-M Nyfors

Ratkaisun valintakokous 19.9.2016, Fatman Oy, Läsnä: J Kärnä, M Makkonen, Mika Wennonen ja M Walhroos

Stack exchange – Best storing practices. Luettavissa: http://programmers.stackexchange.com/questions/209421/best-practice-to-store-datetime-based-on-timezone. Luettu: 1.9.2016.

Stack Overflow (1). Luettavissa: http://stackoverflow.com/questions/13576571/how-to-make-configurable-displayformat-attribute. Luettu: 1.9.2016.

Stack Overflow (10). Luettavissa: http://stackoverflow.com/questions/9492377/datatype-dataannotation. Luettu: 5.9.2016.

Stack Overflow (11). Luettavissa: https://stackoverflow.com/questions/30431901/datepicker-tag-default-value-in-specific-timezone. Luettu: 7.9.2016.

Stack Overflow (2). Luettavissa: http://stackoverflow.com/questions/18624766/range-and-displayformat-attributes-on-timespan. Luettu: 2.9.2016.

Stack Overflow (3). Luettavissa: http://stackoverflow.com/questions/20456712/how-to-get-current-time-with-jquery. Luettu: 2.9.2016.

Stack Overflow (4). Luettavissa: http://stackoverflow.com/questions/3289198/custom-attribute-on-property-getting-type-and-value-of-attributed-property. Luettu: 1.9.2016.

Stack Overflow (5). Luettavissa: http://stackoverflow.com/questions/5700049/changing-date-display-format-based-on-the-selected-time-zone. Luettu: 3.9.2016.

Stack Overflow (6). Luettavissa: http://stackoverflow.com/questions/5958662/mvc-utc-date-to-localtime. Luettu: 5.9.2016.

Stack Overflow (7). Luettavissa: http://stackoverflow.com/questions/6637679/reflection-get-attribute-name-and-value-on-property. Luettu: 3.9.2016.

Stack Overflow (8). Luettavissa: http://stackoverflow.com/questions/7577389/how-to-elegantly-deal-with-timezones. Luettu: 4.9.2016.

Stack Overflow (9). Luettavissa: http://stackoverflow.com/questions/8997129/how-to-show-a-custom-error-message-for-displayformat-attribute-in-asp-net-mvc. Luettu: 4.9.2016.

Työmääräarviointi kokous 19.10.2016, Fatman Oy, Läsnä: J Kärnä, M Makkonen, Mika Wennonen ja M Walhroos

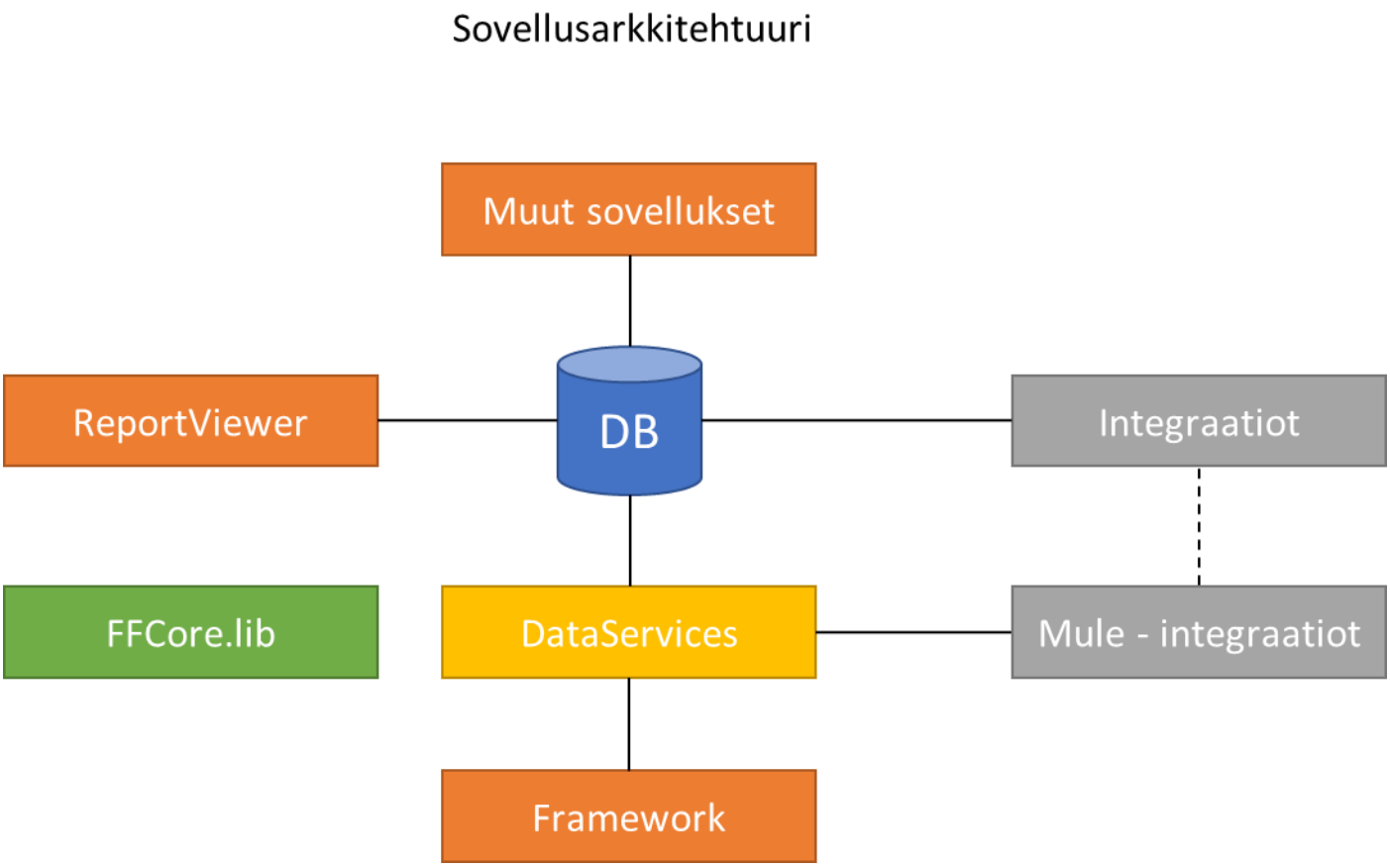
W3schools - Time tag. Luettavissa: http://www.w3schools.com/tags/tag_time.asp. Luettu: 20.9.2016.

Wikipedia - SRSS. Luettavissa: https://en.wikipedia.org/wiki/SQL_Server_Reporting_Services. Luettu: 2.9.2016.

Liitteet

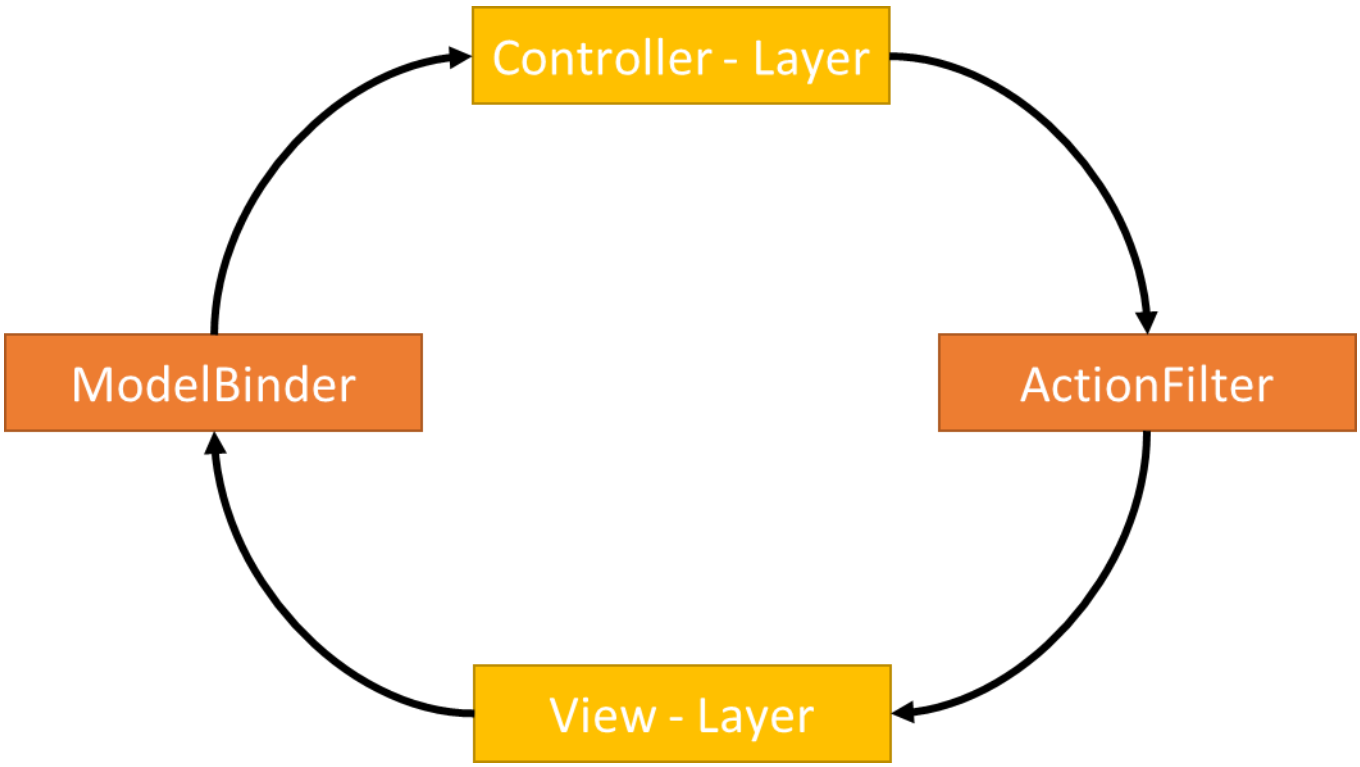
Liite 1. Sovellusarkkitehtuuri

Avattu kappaleessa: 3.1 Sovellusympäristön esittely



Liite 2. Ratkaisun arkkitehtuuri

Käytetty ratkaisun valintakappaleessa kuvaamaan ratkaisun toteutusta.



Liite 3. Käsiteluettelo

.NET MVC Framework	Microsoftin kehittämä sovelluskehys web-sovellusten luontiin. Aiheesta lisää: https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx
Action Filtter	.NET MVC -sovelluskehyksessä mahdollisettuja funktiotyyppiin perustuvia metodi seuloja. Tiettyyn metodityyppiin voidaan tämän avulla sitoa koodia joka suoritetaan aina metodin suorituksen yhteydessä. Aiheesta lisää: https://msdn.microsoft.com/en-us/library/dd410209(v=vs.100).aspx
Aikavyöhyke	Aikavyöhyke on alue maapallolla, jossa käytetään samaa kellonaikaa. Aiheesta lisää: https://fi.wikipedia.org/wiki/Aikavy%C3%B6hyke
C#	Microsoftin kehittämä ohjelmointikieli. Aiheesta lisää: https://fi.wikipedia.org/wiki/C_sharp
Controller	.NET MVC -sovelluskehyksen tapa käsitellä sovellukseen kohdistettuja HTTP-pyyntöjä. Aiheesta lisää: https://msdn.microsoft.com/en-us/library/dd410269(v=vs.100).aspx
Delphi	Ohjelmointikieli
Esitystaso	Kolmitasoisen sovelluskehityksen osa, joka vastaa käyttäjälle näkyvästä osasta sovellusta. Aiheesta lisää:https://msdn.microsoft.com/en-us/library/ff648105.aspx
Fatman Framework	Fatman Oy:n kehittämä sovellus
GUI-Käsittely	Työssä käsitelty ratkaisuvaihtoehto aikavyöhykeen käsittelylle
Integraatio	Toteutus kahden tai useamman sovelluksen välisestä yhteistoiminnasta
JavaScript	Web-ympäristölle suunniteltu komentosarjakieli. Aiheesta lisää: https://fi.wikipedia.org/wiki/JavaScript
Luokka (class)	Luokka on olio-ohjelmoinnissa käytetty palikka, jota pääasiassa käytetään arvojen säilyttämiseen kokoelmina. Aiheesta lisää: https://fi.wikipedia.org/wiki/Luokka_(ohjelmointi)
Mule	Java-ohjelmointikielellä toteutettu sovellusalusta, joka on suunniteltu sovellusintegraatioille. Aiheesta lisää: https://www.mulesoft.com/
PHP	Ohjelmointikieli

SaaS	Tulee lyhenteestä Software as a service. Termi tarkoittaa yrityksen liiketoiminta staregiaa, jossa sovellus on yrityksen palvelimella ja asiakkaat käyttävät sovellusta selaimella tai vastaavalla ohjelmiston käytön mahdollistavalla sovelluksella. Aiheesta lisää: https://en.wikipedia.org/wiki/Software_as_a_service
Single page sovellus (SPA)	Web-ohjelma joka suunniteltu simuloimaan työpöytä sovelluksia. SPA toteutetaan dynaamisesti niin että sovellus ei tee sivulatauksia vaan kaikki sovelluksen toiminnallisuus tapahtuu samalla sivulla. Aiheesta lisää: https://en.wikipedia.org/wiki/Single-page_application
SOA	"Service-oriented architecture". Sovellus arkkitehtuuri ratkaisu jossa sovelluksen business- ja esitys-logiikka on irroitettu toisistaan. Aiheesta lisää: https://en.wikipedia.org/wiki/Service-oriented_architecture
SOAP	"Simple Object Access Protocol". Sovellusten välinen kommunikaatio protokola joka tyypittää miten sovellukset keskustelevat keskenään. Aiheesta lisää: https://en.wikipedia.org/wiki/SOAP
Sprint	Scrum sovelluskehitykseen liittyvä kiinteä aikaväli, jonka aikana tapahtuu sovelluskehitystä. Aiheesta lisää: https://www.scrum.org/resources/scrum-glossary?gclid=Cj0KEQjwzd3GBRDks7SYuNHi3JEBEiQAIm6EI-flQro3tdfMhcWepK8n6vWLI66PXtTJO0cxnxIXbaoaAtRw8P8HAQ
SQL	Standardoitu kyselykieli, jota käytetään relaatiotietokannan hallintaan.
Tietokantakohtainen aikavyöhyke	Työssä käsitelty ratkaisuvaihtoehto aikavyöhykeen käsittelylle
UTC	Yleisaika johon aikavyöhykkeitä verrataan.
UTC Tietokanta	Työssä käsitelty ratkaisuvaihtoehto aikavyöhykeen käsittelylle
Versiohallinta	Tekniikka jolla säilytetään useita versioita ohjelman toteutuksesta. Tekniikalla mahdollistetaan hallittu kehitys, joka sallii useat samanaikaiset ohjelmistokehittäjät.