Oskari Virtanen

Geolokaatioelementti EBX järjestelmään

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

04.05.2021

Tiivistelmä

Tekijä:	Oskari Virtanen
Otsikko:	Geolokaatioelementti EBX-järjestelmään
Sivumäärä:	26 sivua
Aika:	04.05.2021
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikan tutkinto-ohjelma
Ammatillinen pääaine:	älykkäät järjestelmät
Ohjaajat:	Lehtori Sami Sainio
	-

Insinöörityön tarkoituksena oli rakentaa Tibco Softwaren EBX-järjestelmään komponentti, jonka avulla voi tallentaa paikkatietoja järjestelmän tietokantaan. Projekti tuotettiin EnfoOy:lle ja heidän versioonsa EBX-järjestelmästä.

Komponentin on tarkoitus täyttää kaksi roolia: sen pitää täyttää järjestelmän tietokannan jäsenet geolokaatiodatalla ja näyttää nämä tiedot osana järjestelmän UI-elementtiä. Molempiin tarkoituksiin se käyttää EBX:n sisäisiä ja Google Maps API:n tarjoamia ominaisuuksia. EBX-järjestelmänä toimii Master Data Management (MDM) työkaluna, jonka tarkoitus on yhdistää ja hallita jonkun organisaation kaikkea jaettua dataa.

Projektin tuotantoprosessi koostui EBX:ään ja Googlen API:iin tutkimisesta, opettelemisesta ja implementoinnista. Suurin osa raportin sisällöstä koostuu komponentin rakennuksessa kohdatuista haasteista ja ratkaisuista. Projektin tuottaminen vaati osaamista Java- ja Javascript-ohjelmointikielissä, Apache Tomcat -ohjelmistossa, ja EBX-järjestelmässä.

Lopputulos projektista vastaanotettiin positiivisesti. Kaikki Enfon asettamat tarvittavat päämäärät ja muutama ylimääräinen tavoite saavutettiin projektin aikana. Lopputuloksena oli komponenti EBX-järjestelmään, joka pystyy käsittelemään ja esittämään
geolokaatiodataa.

Avainsanat:	MDM, master data management, Geolokaatio, Google Maps API, Maps JavaScript API, Maps Static API, Geocoding API, EBX, Java, JavaScript

Abstract

Author:	Oskari Virtanen
Title:	Geolokaatio elementti EBX järjestelmään
Number of Pages:	26 pages
Date:	04 05 2021
Degree:	Bachelor of Engineering
Degree Programme:	Tieto- ja viestintätekniikan degree programme
Professional Major:	Smart Systems
Supervisors:	Sami Sainio, Lecturer

The purpose of the project was to produce a component for Tibco Softwares EBX data management system, that could fill a data record with geolocation data and display that data for the user. This project was produced for Enfo oyj and their version of the EBX software system

Component must fulfill two responsibilities: it must be able to fill EBX dataset records with geolocation data and then display that data within the EBX UI element. For both purposes it uses EBX internal functionality and Google Maps API. EBX itself is a Master Data Management (MDM) system designed for unifying and managing data across an organization.

The process for this project consisted of investigating EBX functionality and Google Maps API, learning how to use them and implementing them in a solution. This report mostly consists of challenges and solutions encountered during this project. This project required knowledge in Java and Javascript, Apache tomcat application and EBX data structure & functionality.

The end product of the project fulfilled all necessary and some optional requirements set, response to the results was positive. The end product was a component fo the EBX-system that could manipulate and display geolocation data.

Keywords: MDM, master data management, Google Maps API, Maps JavaScript API, Maps Static API, Geocoding API, EBX, Java, JavaScript

Sisällys

Lyhenteet

EBX: On nimi Tibco Softwaren tuottamalle MDM-järjestelmälle, johon on projektissa tarkoitus rakentaa geolokaatioelementti.

MDM: *Master Data Management.* On järjestelmä, jonka tarkoitus on yhdistää organisaation jaettu data ja pitää se yhdenmukaisena ja tarkkana.

UI: *User Interface*. Rajapinta, jonka kautta ohjelmistoa käytetään. Tässä projektissa UI-elementti tarkoittaa HTML:ään pohjautuvaa EBX-käyttäjärajapintaa.

API: *Application programming interface*. Ohjelmistorajapinta, määritelmä, jonka mukaan muut ohjelmat voivat käyttää API:n sisältämiä funktioita ja päästä käsiksi sen dataan.

JAR: *Java Archive.* On tiedostopakettimuoto, jota käytetään Java-luokkien ja tarvittavien tiedostojen levittämiseen.

WAR: *Web Application Resource/ Web application Archive.* On tiedostopakettimuoto, jota käytetään nettipohjaisten applikaatioiden tiedostojen levittämiseen. Voi sisältää JAR -, Java-, XML-tiedostoja jne.

IDE: *Integrated Development Environment.* On ohjelmisto, jonka sisällä voi ohjelmoida ja ajaa koodia. IDE-ohjelmat usein sisältävät työkaluja koodin testaamiseen, virtaviivaistamiseen ja korjaamiseen.

1 Johdanto

Työn tavoitteena oli tuottaa Enfo Oy:lle komponentti heidän Tibco Softwarelta lisensoituun EBX-järjestelmään, jonka tarkoitus on mahdollistaa geolokaatiodatan lisääminen EBX:ään tallennettuihin paikkatietoihin. Kun järjestelmässä on datajäsen, joka sisältää tiedon jostain osoitteesta on komponentin tarkoitus pystyä näyttämään osoite kartalta ja lisätä osoitteesta puuttuvat tiedot järjestelmään.

Komponentti on rakennettu seuraten EBX-järjestelmän master data management -periaatteita ja toimii tutkimuksena sen toteuttamisesta.

Komponentti rakennettiin paketiksi, jonka EBX lataa käynnistyessään. Paketti tuo järjestelmään kaksi toiminnallisuutta: Toinen ohjelma lukee annetun ryhmän dataa ja tekee ryhmän jäsenien tietojen perusteella kutsun Google Maps API -rajapintaan ja täyttää vastauksella puuttuvat tiedot. Toinen paketissa tuleva ohjelma ottaa dataryhmän jäsenen tiedot ja niiden perusteella näyttää saman API:n avulla interaktiivisen tai passiivisen kartan, joka näyttää annetun sijainnin.

Projekti ohjelmisto komponentteina toimivat EBX-järjestelmä, Google Maps API:n ja Apache Tomcat -ohjelmiston. EBX ja API toimivat pohjana, jonka päälle projektin koodi rakennettiin ja Apache Tomcat on alusta, jonka päällä EBX-web applikaatio toimii.

Tämä raportti on jaettu seitsemään lkuun. Ensimmäinen luku on johdanto. Toinen luku käsittelee projektin alkua: kenelle ja miksi työ on tehty. Esittelen myös järjestelmän, johon komponentti rakennettiin. Kolmas luku käsittelee master data management -teoriaa ja miten se liittyy projektiin. Neljäs luku käsittelee kaikkia työkaluja, mitä projektissa on käytetty. Viides luku käsittelee projektin teknistä suunnittelua ja toteutusta. Kuudes luku käsittelee projektin kulkua ja sen aikana muuttuneita tavoitteita ja suunnitelmia. Seitsemäs luku on yhteenveto koko projektista.

2 Projektin alkuasetelma

2.1 Enfo Oyj

Enfo Oyj on pohjoismainen IT-alan julkinen osakeyhtiö, joka tarjoaa erimuotoisia datapalveluja esimerkiksi datan hallinta-, analyysi- tai käyttöpalveluja. Sen asiakkaina toimivat muut yritykset, joita Enfolla on asiakkaina 359. [1.]

2.2 EBX-järjestelmä

EBX on TIBCO Software Inc:n tuottama ohjelmistoratkaisu, joka on suunniteltu organisaation datan keräämiseen ja yhdistämiseen ns. masterdatan hallitsemiseen. [2.] Enfo toimii TIBCO:n edustajana Pohjoismaissa ja Baltiassa. Yhteistyön myötä Enfolla on oikeus käyttää TIBCO EBX -tuotetta omaan tuotekehitykseen.

2.3 Projektin tavoitteet

Projektin tavoitteitta voi tarkastella kahdesta näkökulmasta. Ensimmäinen näkökulma on isinöörityön näkökulma ja toinen on projektin tilaajan eli Enfon näkökulma.Työn tavoite projektissa on tuottaa komponentti, joka täyttää master data management -kehitysfilosofian vaatimukset MDM -järjestelmästä.

Enfon määrittämä tavoite projektille on tuottaa Enfolle komponentti, jonka avulla he voivat esitellä EBX:n ominaisuuksia ja mahdollisuuksia asiakkaille (mahdollisesti käyttöön). Projektin tavoitteeksi määriteltiin luoda käyttäen Java-ohjelmointikieltä paketti, jonka tarkoitus on käsitellä ja näyttää geolokaatiodataa käyttäjälle. Projektin yleisen päämäärän lisäksi asetettiin ominaisuuksia, jotka vaaditaan projektin hyväksymiseksi.

Näitä tavoitteita jaettiin kriittisiin, tarvittaviin ja haluttuihin ominaisuuksiin. Jaon tarkoitus oli helpottaa ja ohjata kehitysprosessia.

Taulukko 1. Lista ominaisuuksista.

Omaisuus	**Tärkeys**	**Tila**
Kartta datan käsittely	Kriittinen	Implementoitu
Staattinen kartta	Kriittinen	Implementoitu
Konfiguraatio tiedosto	Kriittinen	Implementoitu osittain
Asennuspaketti	Kriittinen	Implementoitu
Workflow implementaatio	Kriittinen	Implementoitu
Dynaaminen kartta	Tarvittava	Implementoitu
Datan tallennus valinta	Tarvittava	Implementoitu
Dynaamisen kartan parantaminen	Haluttu	Implementoitu osittain
Käyttäjä kokemuksen parantaminen	Haluttu	Implementoitu osittain
Käyttäjä workflow	Haluttu	Ei implementoitu

Suurin osa tavoitteista saavutettiin projektin aikana. Vain käyttäjäpohjainen workflow jäi täysin implentoimatta ja osittain implementoidut ominaisuudet ovat avoimia, joita voi periaatteessa kehittää loputtomasti

3 Master Data Management ja projektin teoria

3.1 Master Data Management (MDM) ja master record

Yksinkertaisesti ilmaistuna Master Data Management (MDM) on pyrkimys pelkistää organisaation data yksinkertaisimpaan ja hyödyllisimpään muotoon. MDM pyrkii helpottamaan datan käsittelyä standardoinnilla, datan laadusta huolehtimisella ja automaatiolla.

Esimerkiksi jos yrityksellä on varasto, josta se myy tuotteita, sen datan hallinta on kriittinen osa yrityksen toimintaa. Hyvin hallitussa varastossa jokainen lista varaston sisällöistä sisältää samat tiedot ja, kun varastolle tulee tai lähtee

tavaraa, varaston yleinen tietokanta päivittyy. MDM-toteutus olisi tässä tilanteessa, vaikka sisään tulevien ja lähtevien tavaroiden listojen vertaaminen yleiseen listaan varaston tavaroista ja yleislistan päivittäminen muutoksien mukaan. Pitkälle kehitetty MDM-järjestelmä myös pyrkisi prosessien automaatiolla vähentämään ihmisvirheitä.

Kuten luvussa 2.2 lyhyesti mainittiin, EBX on MDM-systeemi eli sen käyttötarkoitus organisaation yhteisen datan hallitseminen. MDM voi viitata joko datan yhdenmukaistettuun hallintaperiaatteeseen tai järjestelmään, joka toteuttaa sitä. MDM-järjestelmän keskiössä on niin sanottu master record, joka sisältää ”oikean” version datasta. [3.] Työtä esimerkkinä käyttäen EBX-järjestelmän data model on rakenne, joka toimii mallina master record-datajäsenelle. Varastoesimerkissä master record olisi viittaus varastolla olevaan yksittäiseen tavaraan.

Tavoitellut hyödyt MDM-järjestelmässä liittyvät datan laatuun, käytön parantamiseen, datan hallinnan parantamiseen ja analyysin helpottamiseen. Onnistuneen MDM-toteutuksen keskellä on datan yksinkertaistaminen ja yhdenmukaistaminen. [4.] Varastoesimerkissä onnistunut MDM-implementaatio toteutuu, kun yleislistaa vastaavat tavarat löytyvät varastosta sellaisina kuin ne yleislistassa on kuvailtu ja kaikki tieto, joka tavarasta tarvitaan, löytyy yleislistasta.

3.2 MDM järjestelmänä

Knut Jürgensenin artikkelissa (ks. Lähde 4) kuvailtiin MDM-järjestelmän toimintaa suomennettuna seuraavasti: MDM-järjestelmä saavuttaa potentiaalisen hyötynsä organisaatiolle, jos sen data on luotettavaa ja saatavilla kaikille organisaation osastoille. Tämän saavuttaakseen MDA (master data admin):n on säilytettävä tiukkoja laatu standardeja ja valvoa hallinta sääntöjä.

Projektin näkökulmasta datan luotettavuus on tärkein elementti, mihin keskittyä. Jürgensen antaa artikkelissaan esimerkit huonosta ja hyvästä MDM-

järjestelmästä. Huonon järjestelmän ongelma on, että dataa MDM-järjestelmään tulee kahden eri rajapinnan läpi, kahdesta eri datalähteestä. Tämä Jürgensenin mukaan aiheuttaa ongelmia datan käytössä ja saatavuudessa. Koska data ei ole yhdenmukaista, master record -rakenne ei toteudu. Hyvässä esimerkissä data tulee yhdestä lähteestä yhden rajapinnan kautta. Tämä on ideaali tilanne, jossa data on mahdollisimman yhdenmukaista ja minimoi riskin virheille tai datajäsenkopioille, jotka laskevat datan laatua.

3.3 MDM:n eri tyypit

MDM-järjestelmät luokitellaan yleisesti neljään eri tyyppiin. Näihin kuuluvat:

- rekisterityyli (Registry)
- yhdistettytyyli (Consolidation)
- rinnakkaistyyli (Coexistence)
- keskitettytyyli (Centralized).

Rekisterityyli rakentuu puhdistus- ja vertausalgorytmien ajamiseen eri lähteistä kerättyyn dataan. Sen keskeinen toiminto on antaa uniikki ja globaali tunniste algorytmien löytämille tiedoille yhtenäisen tiedon löytämiseksi. Yhdistetyssä tyylissä data otetaan useasta lähteestä master datan muodostamiseksi. Rinnakkaistyyli vastaa yhdistettyä tyyliä, mutta sen eroavaisuutena on, että master record rakennetaan eri tietojärjestelmään. Toisin kuin yhdistetyssä, rinnakkaistyyli jättää lähdedatan koskematta eli muutoksia voi tapahtua erikseen master recordiin ja lähdedataan. Keskitetty data kehittää masterdataa ja julkaisee sen takaisin lähteelle kaksisuuntaisissa interaktioissa. [5].

3.4 Geolokaatio ja MDM

Luvussa 2.3 käytiin läpi, että projektin tekninen tavoite on luoda EBX-järjestelmään geolokaatioelementti. Elementin tarkoitus siis on sisältää jonkun entiteetin fyysinen lokaatio. Tämä elementti on siis hyvä esimerkki master record -tyyppisestä datasta, sillä yksittäinen entiteetti ei voi olla kahdessa

paikassa samaan aikaan. Esimerkiksi voi ottaa Samarian terveysaseman, joka sijaitsee vain Espoon keskuksessa. Elementin on tarkoitus säilyttää tämän datajäsenen sisällä sen paikkatiedot ja täydentää tai korjata niitä tarpeen tullen.

Projektin voi laskea onnistuneeksi, jos tuotettu elementti kykenee ottamaan dataa itseensä, varmistamaan datan laadun ja pitää datanjäsenen uniikkina master record -periaatteen mukaan.

3.5 MDM ja datan saastuminen

Kuten luvussa 3.2 on selitetty MDM-järjestelmän sydän on master record. Dataentiteetti esittää järjestelmän näkökulmasta ainutta totuutta sen referoimasta datasta. Tästä johtuen kaikki, mikä voi aiheuttaa virheitä datassa, on riski järjestelmälle ja on minimoitava toteutuksessa.

Näitä riskejä voi olla esimerkiksi usean eri rajapinnan käyttö, huono datan lähde, huonosti suunniteltu datan käsittely tai sopimattomat datan ylikirjoitukset. Joitakin riskejä, kuten käyttäjävirhettä ei voi poistaa automatisoimatta järjestelmää kokonaan, mikä tuo taas omat riskinsä.

Komponenttien suunnittelussa on siis otettava erityisesti huomiota yksinkertaisuuteen, jotta edellä mainitut sekä muu mahdollisten riskien määrä ja todennäköisyys minimoidaan.

4 Projektin työkalut, rajapinnat ja kehitysympäristö

4.1 Eclipse IDE

Projektin kehitysympäristönä toimi Eclipse IDE, joka on Eclipse Foundationin ylläpitämä ohjelmistosuunniteltu erityisesti Java-projektien kehittämiseen. Eclipse sisältää paljon muitakin ominaisuuksia, mutta tässä projektissa käytettiin sen ominaisuutta hallinnoida Java-projekteja.

Projektin koodi kirjoitetaan Eclipsessä ja prosessoidaan WAR-paketiksi, joka puolestaan lisätään EBX-ympäristöön. Web Application Resource/ Web application Archive (WAR) on tiedostopakettimuoto, jota käytetään nettipohjaisten applikaatioiden tiedostojen levittämiseen. EBX-järjestelmän toiminta ei vaadi paketin toimintaa, joten koodissa olevan virheen vaikutus järjestelmän toiminnallisuuteen on rajattu. Pakettirakenne myös helpottaa projektin käyttöönottoa eri EBX-ympäristöissä.

4.2 EBX-ympäristö

4.2.1 EBX: johdanto

EBX-järjestelmä on tietokantahallintatyökalu, joka käyttää pohjana Apache Tomcat HTTP -verkkopalvelinympäristöä, joka pyörittää EBX-koodia ja tietokantaa. EBX-ympäristö, johon projekti on tehty käyttää, Apache Tomcat -versiota 9.0.41. EBX-ympäristö on suunniteltu modulaariseksi tarkoittaen, että jos käyttäjä haluaa muokata toiminallisuutta tai kehittää uusia ominaisuuksia järjestelmään, voi ne lisätä JAR- (Java Archive) tai WAR-pakettien muodossa Apache Tomcatin kautta osaksi ympäristöä.

Ympäristön UI elementtiin pääsee käsiksi webselaimen kuten Goglen Chromen tai Windowsin Edgen kautta.

EBX-järjestelmän käyttäjän rajapinta koostuu siis HTML-sivuista. Java-koodia ajettaessa sivut toimivat lomakkeina, joten sivuihin tehdyt erikseen tallentamattomat muutokset eivät siirry tietokantaan, mutta EBX sisältää myös javascript-toiminallisuutta, joka mahdollistaa tiettyjen asioiden tekemisen HTML-sivun sisällä.

Käyttäjän rajapinnan kautta voi selata, muokata ja luoda tietokannan jäseniä sekä luoda ja käyttää automaattisia operaatioita. Näitä operaatioita kutsutaan nimellä workflow. EBX-järjestelmä myös sisältää kyvyn luoda käyttäjiä, joille voi asettaa, mitä työkaluihin tai osaa tietokannasta käyttäjä voi käyttää tai muokata.

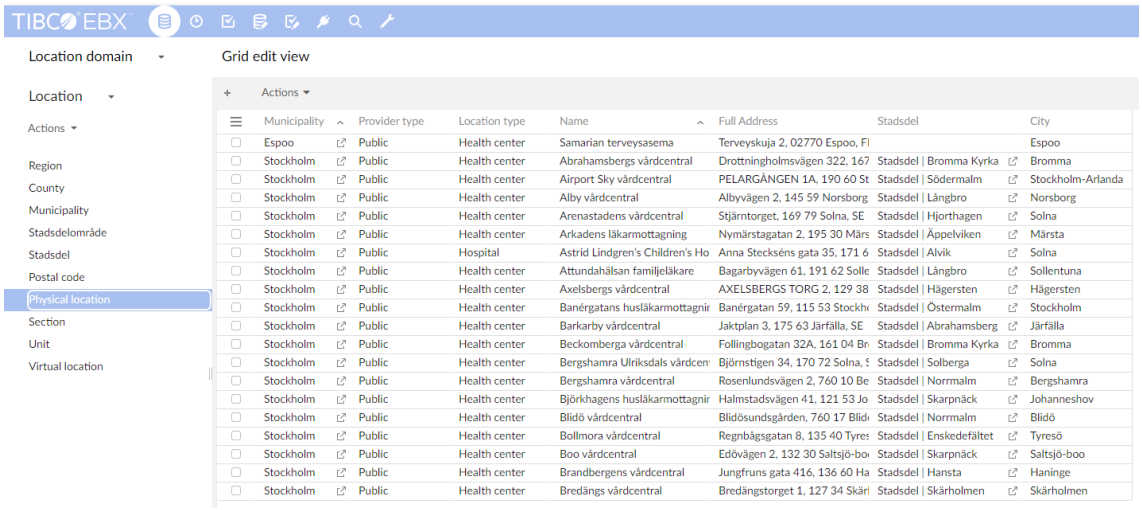
4.2.2 EBX: MDM-tyyppi

EBX ympäristönä tukee kaikkia luvussa 3.3 kuvailtuja MDM-tyyppejä. Tibcon EBX-ympäristöä esittelevien sivujen mukaan, MDM-alustojen tulisi tukea kaikkia neljää master data management -päätyyliä.[6]

Projektin tarkoituksiin EBX on konfiguroitu käyttäytymään yhdistetyllä tyylillä. Master record säilytetään datajäsenenä, jota referoidaan, kun dataa halutaan käyttää. Datasta luodaan kopio ja operaation jälkeen kopion data hylätään tai yhdistetään master recordiin,

4.2.3 EBX: Data, Dataset ja Dataspace

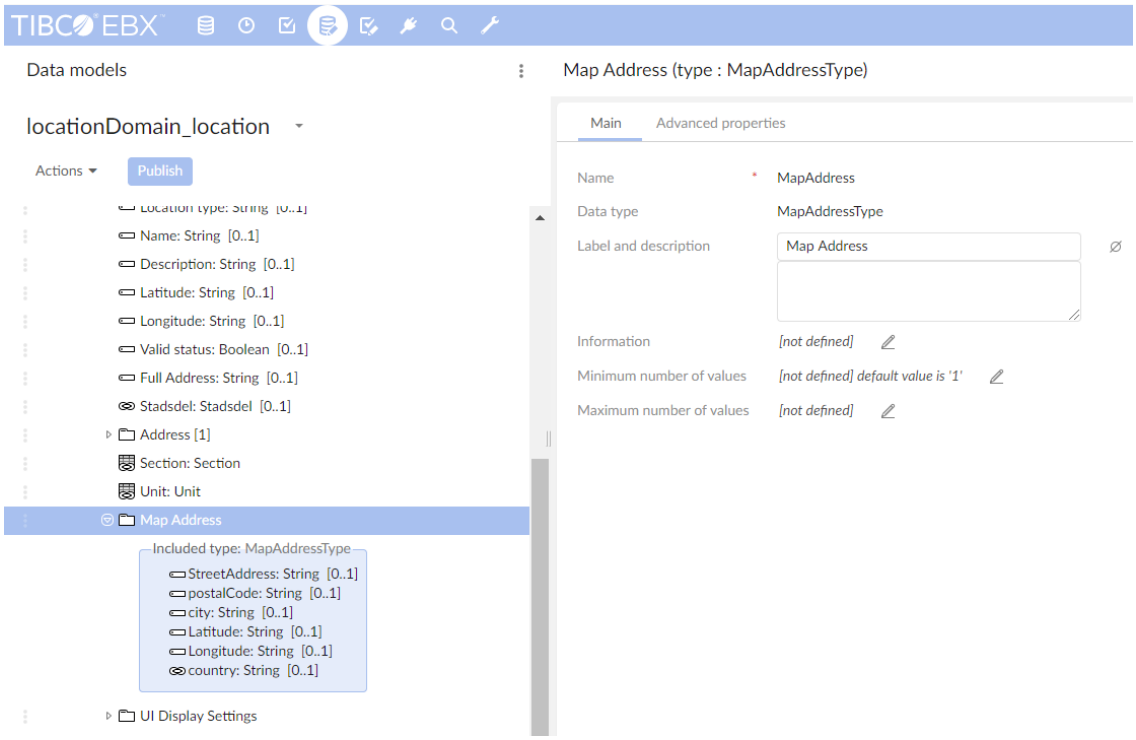
Tietokanta koostuu dataset-rakenteista, jotka sisältävät datajäseniä. Datajäsen vastaa luvussa 3 käsiteltyä master record -konseptia eli on uniikki entiteetti, joka sisältää jonkun datan. EBX käsittelee datasettejä dataspace-rakenteessa, joka edustaa sillä hetkellä käsiteltävää versiota datasetistä. Kun dataa muokataan tietokannassa, luodaan väliaikainen datapace, joka yhdistetään todelliseen dataspaceen operaation loppuessa. [7.]



Kuva 1. EBX-järjestelmän Data näkymä. Location domain dataset

Datasetjäsenet määritellään datamodel-rakenteen mukaan. Datamodel on rakenne, joka määrää, minkälaiset tiedot datajäsen pitää sisällään. Rakenteen

tarkoitus on pitää data yhdenmukaisena, ja se on rakennekaava MDM-mallin mukaiselle master recordille.

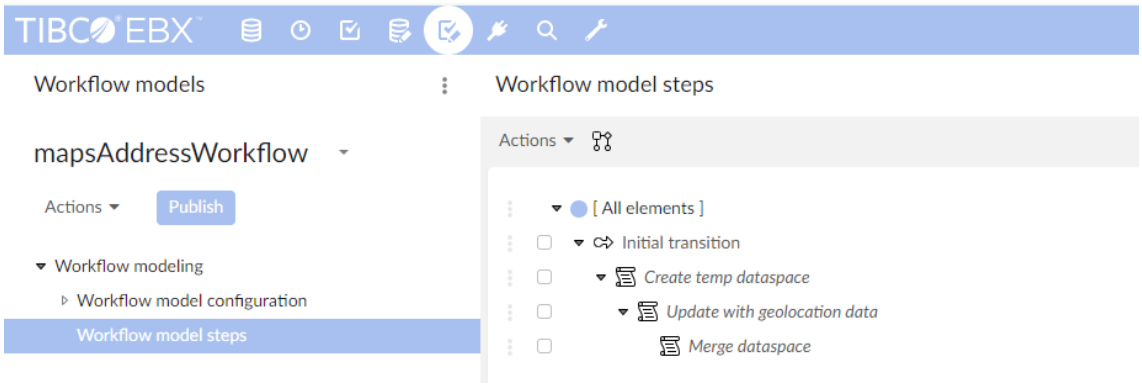


Kuva 2. Datamodel, jota Location domain dataset -jäsenet seuraavat

Datamodel voi koostua muuttujista, listoista muuttujia ja voi ottaa osaksi muita datamodel-rakenteita. Kuvassa 2 esillällä on projektissa käytetty datamodel locationDomain_location -rakenne, jonka sisällä datamodel Map Address määrää muodon, missä geolokaation kautta saatu paikkatieto ilmaistaan.

4.2.4 EBX: Workflow

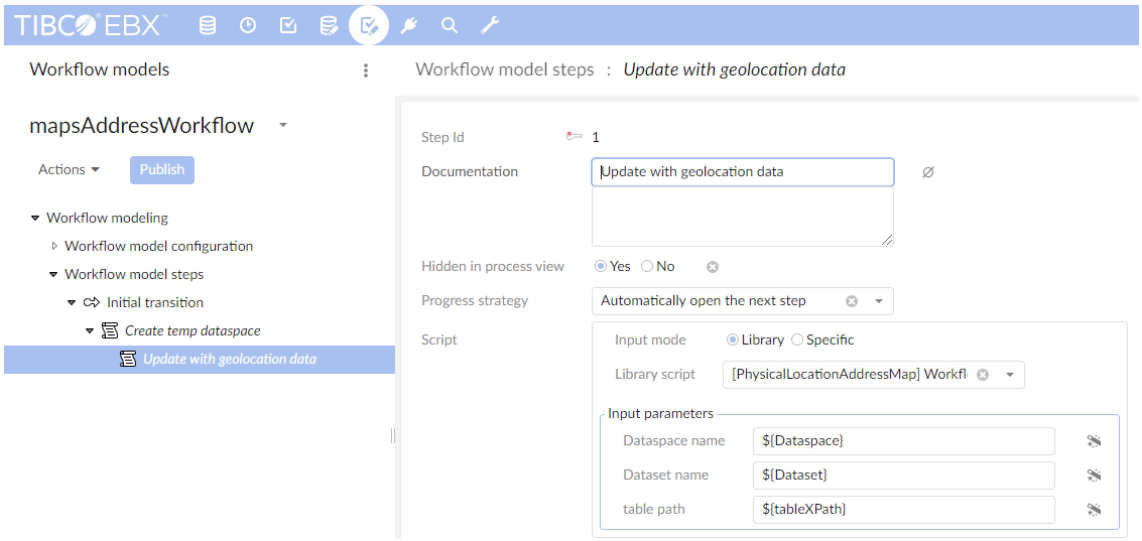
Workflow on EBX-järjestelmän sisäinen operaatio, joka voi olla automaattinen tai vaatia käyttäjän osallistumista. Workflow rakennetaan sarjaan askelia, jotka ovat joko sisäänrakennettuja operaatioita tai erikseen koodattuja operaatioita.



Kuva 3. EBX workflow model -näkymä

Kuvassa 3 on mapAddressWorkflow workflow model -näkymässä, jossa workflow -malleja voi muokata tai luoda. Kuvasta ilmenee yleinen askelrakenne work-flow -mallille. Jokaiselle askeleelle Initial transitionia lukuun ottamatta voi määritellä koodin, jonka se ajaa sisäänrakennetuista tai itsetehdyistä koodeista sekä määritellä operaatiossa käytettävät muuttujat.

Workflow-rakenteella voidaan toteuttaa MDM-mallin vaatima operaatioita, kuten datan laadun valvominen ja datan korjaaminen. Prosesseja voi luoda kaikkiin datanhallintaan tarvittaviin operaatioihin. Kuten luvussa 2.3 määriteltiin, yksi tavoitteista oli workflow-rakenteen, joka käy datasetin läpi ja lisää puuttuvan geolokaatiodatan jäseniin, rakentaminen.



Kuva 4. Workflow model -askeleen muokkaus näkymä

4.3 Google Maps API

Google Maps Platform on joukko API-rajapintoja, joita voi käyttää Googlen karttapalveluiden implementoimiseen omassa sovelluksessa. Googlen API-avaimella voi tehdä kyselyjä Googlen tietokantoihin, joista voi hakea esimerkiksi paikka- ja reitti-dataa. Projektissa on käytössä kolme rajapintojen tarjoamaa ominaisuutta: Maps Static Apin tarjoamaa staattista karttaa, Maps JavaScript Apin tarjoamaa dynaamista karttaa ja Geocoding Apin tarjoamaa geokoodausominaisuutta.

4.3.1 Maps Static API

Maps Static API antaa kyvyn liittää Google Maps -kuvan HTML-tiedostoon tarvitsematta JavaScript-koodia tai dynaamista sivun lataamista. API-palvelu luo kartan http-pyynnössä lähetettyjen URL-parametrien mukaan, jota voi käyttää kuvana HTML-tiedostossa. [8.]

4.3.2 Maps JavaScript API

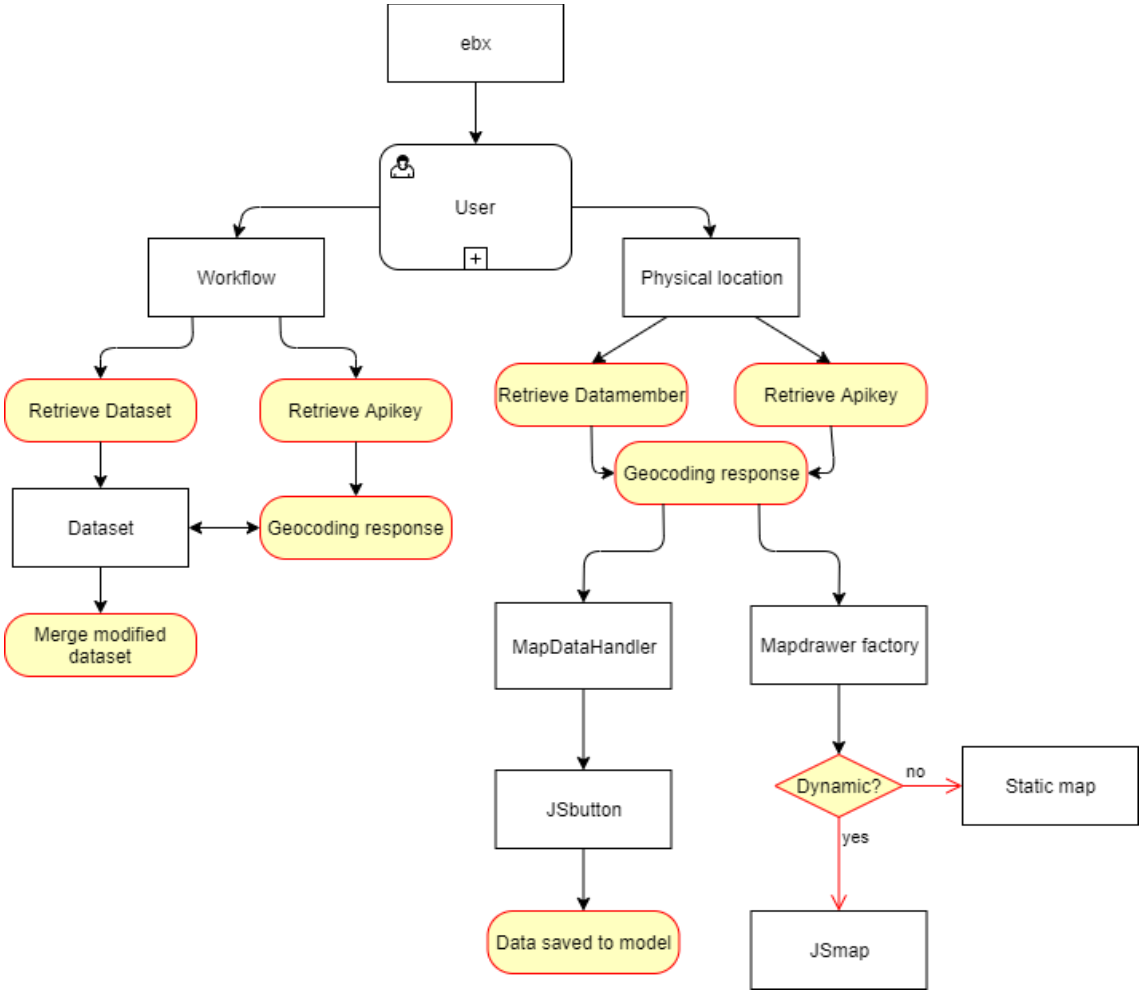
Maps JavaScript API mahdollistaa JavaScript-pohjaisen kartan upottamisen HTML-tiedostoon. API antaa dynaamisen kartan, jonka ominaisuuksia voi muokata ja määrittää sen kutsussa. [9.]

4.3.3 Geolocation API

Geolokaatio API antaa kyvyn geokoodaamiseen, mikä googlen määritelmän mukaan on prosessi, joka muuttaa osoitteen maantieteellisiksi koordinaateiksi. [6.] Sitä käytetään projektissa yhdessä Apien kanssa ottamaan ylös annetun osoitteen koordinaatit ja näyttämään sen kartalla.

5 Projektin toteutus

Projekti suunniteltiin toteutettavaksi kahdessa osassa. Ensimmäinen osa koostui yksittäisen datajäsenen käsittelystä ja visuaalisesta esittämisestä käyttäjälle. Toinen osa taas koostui koko datasetin käsittelystä taustalla workflow-rakenteen kautta.



Kuva 5. Projektin toetukseen yksinkertaistettu malli. Luotu projektin alussa toteutuksen suunnittelun aikana.

5.1 Geodata

Yksinkertaisin elementti projektista on geokoodauselementti. Se toteutetaan valitsemalla ensin datajäsen tietokannasta. Valitusta jäsenestä valitaan paras mahdollinen hakutermi seuraavalla funktiolla:

```
private String selectBestSearchTerm(Adaptation address) {

String searchTerm = null;
final String streetName =
(String) address.get(AddressPaths._Property._Address_City StreetName);
// short circuit AND in use: streetName length only evaluated if
streetName
//is not null or empty
if (streetName != null && !streetName.isEmpty() && streetName.length()
> 3) {

     searchTerm = (String) new FullAddressString().getValue(address);

} else {
StringBuilder sb = new StringBuilder((String) address.get(Ad-
dressPaths._Property._Name) + ", "
+ (String) address.get(AddressPaths._Property._Address_City)
+ ", "+
 (String) address.get(AddressPaths._Property._Address_Country));

searchTerm = sb.toString();
}
return searchTerm;
}
```

Esimerkkikoodi 1 Hakutermin valintafunktio, kirjoitettu Java-koodilla. Etsii datajäsenestä sen datamodel-rakenteen mukaisista paikoista tietoa, jolla tehdä geolokaatiokutsu.

Kun sopiva termi löytyy, tehdään kutsu Googlen geolokaatiopalveluun. Hakutermin lisäksi tarvitaan API-avain ja kieli ,jolla API tuottaa vastauksen. API-avainta säilytetään projektissa .PROPERTIES-tyyppisessä tiedostossa, jotta avaimen voi vaihtaa tarvittaessa muuttamatta koodia.

```
GeocodingResult[] geoCodingRes = null;

final String geoApiLanguage = "fi";
GeoApiContext geoApiCtx = null;

try {
geoApiCtx = new GeoApiContext.Builder().apiKey(getpiKey()).build();
geoCodingRes = GeocodingApi.geocode(geoApiCtx, searchTerm).
language(geoApiLanguage).await();
} catch (final Exception e) {
     w.add("<div>").add("Exception getting coordinates: ").
add(e.getLocalizedMessage()).add("</div>");
     w.add("<div>").add("Stack trace:<br>").
add(e.getStackTrace().toString()).add("</div>");
}
```

Esimerkkikoodi 2 osa koodista, joka tekee geolokaatiohaun. Koodi on Javaa ja se on osa funktiota, joka rakentaa HTML-tiedoston, joka sisältää datajäsenen tiedot ja lokaation kartalla.

Koodiesimerkki on osa karttakoodia ja esiintyy hyvin samanlaisena workflow-koodissa. Ero näiden kahden koodin välillä on try-catch-rakenteessa, jossa workflow-koodi tekee vikailmoituksen eri tavalla, koska workflow-koodi ei rakenna HTML-tiedostoa.

Geokoodauksen vastaus tulee JSON array -muodossa. Vastauksen purkaminen käsitellään tarkemmin luvussa 5.2.3 karttadatan tallennus.

5.2 Datan visuaalinen esittäminen

Toinen keskimmäisitä tavoitteista projektissa on datan näyttäminen visuaalisesti. Projektin tavoitteisiin kuului sekä staattisen että dynaamisen kartan näyttäminen ja tietojen tallentaminen karttahaun tuloksesta. Karttojen luominen on toteutettu factory konstruktorirakenteen kautta, jossa karttojen välillä valitaan tietokannassa olevan muuttujan perusteella.

```
if (geoApiCtx != null && geoCodingRes != null) {
// Creates the map
MapDrawer md = null;
if (mapChoice) {
  md = MapDrawerFactory.createFactory(INTERACTIVEJSMAP).create(geoAp-
iCtx,geoCodingRes[0].geometry.location, MapDrawer.Zoom-
Level.STREETS_HIGHER, new Size(640, 480), geoCodingRes[0].formattedAd-
dress, getApiKey());

md.setMapUIElements(true, true, false, false, false, false);

} else {
  md = MapDrawerFactory.createFactory(STATICIMAGEMAP).create(geoAp-
iCtx, geoCodingRes[0].geometry.location,
MapDrawer.ZoomLevel.STREETS_HIGHER, new Size(640, 480), ge-
oCodingRes[0].formattedAddress,getApiKey());

}
// Add marker for the location. "X marks the spot".
  md.addMarker(new MapMarker(geoCodingRes[0].geometry.location, "X"));
  w.add(md.getMapHTMLString());
}
```

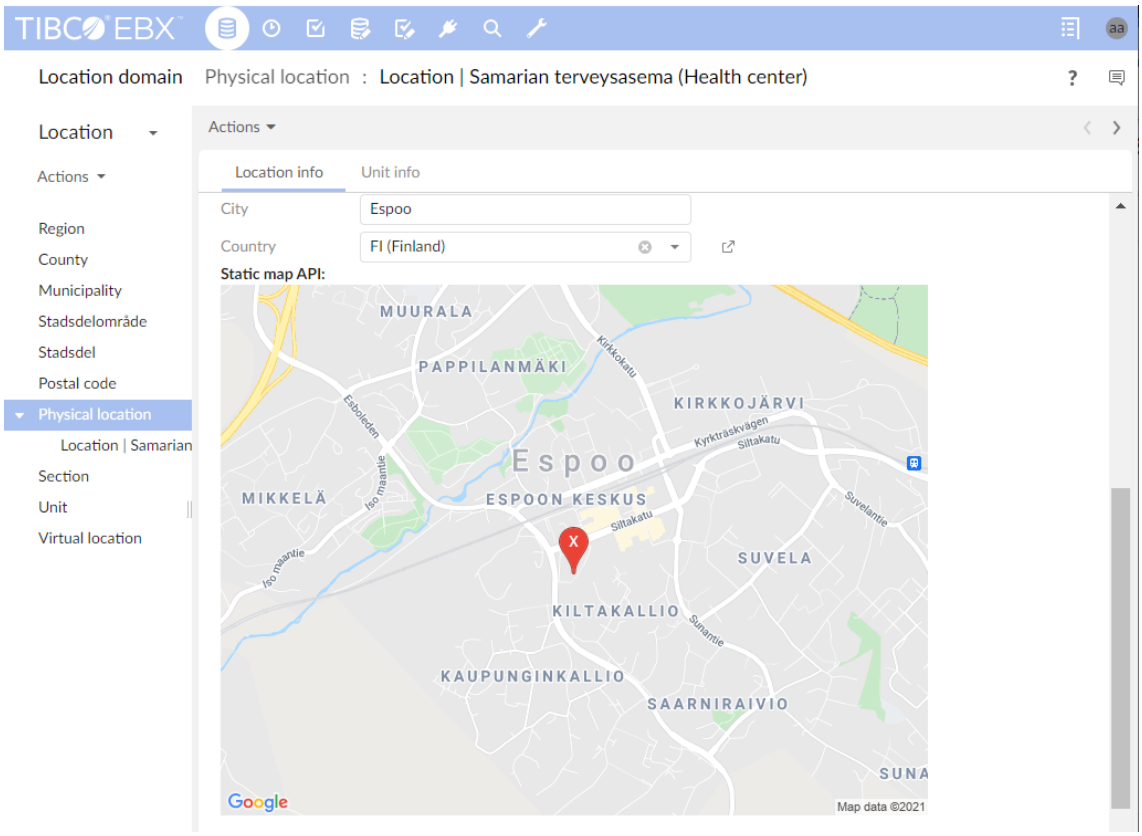
Esimerkkikoodi 3 osa koodista, joka valitsee ,upottaako koodi tuotuun HTML-tiedostoon dynaamisen tai staattisen kartan. Valinta tehdään datajäsenen boolean-muuttujan perusteella.

5.2.1 Staattinen kartta

Staattinen kartta vastaanottaa Static Map API -kutsun vastauksen ja muokkaa siitä png-muotoisen tiedoston, johon se upottaa geodatan koordinaattien mukaisen pisteen. Tämän jälkeen kartan luova funktio rakentaa HTML-koodin String-objektina, jonka se palauttaa ohjelmalle. Tämä String-objekti liitetään käyttäjälle lähetettävään HTML-tiedostoon.

```
private ImageResult getStaticMapImage() throws ApiException, Inter-
ruptedException, IOException
{
  StaticMapsRequest mapRq = StaticMapsApi.newRequest(getGeoApiCtx(),
getMapSize());
  mapRq.format(ImageFormat.png32);
  mapRq.maptype(mapType);
// if there are markers, the map center will be set by them.
  if (!markers.isEmpty()) {
    markers.forEach((mapMarker) -> mapMarker.setGoogleMaps-
Markers(mapRq));
// In case of multiple markers we let the zoom level be automatically
set so
// that all markers are visible.
    if (markers.size() == 1) {
// Only one marker --> we need to set zoom value.
    mapRq.zoom(getZoomLevelIntValue());
  }
} else {
// no markers, we need to set zoom and map center
    mapRq.center(getLocation());
    mapRq.zoom(getZoomLevelIntValue());
}
// request the image
ImageResult imgRes = mapRq.await();
return imgRes;
}
```

Esimerkkikoodi 4 Staattisen kartan hakufunktio.



Kuva 6. Staattinen kartta EBX-järjestelmässä. Esimerkkinä Samarian terveysasema.

5.2.2 Dynaaminen kartta

Dynaaminen kartta upottaa javascript-pohjaisen elementin HTML-tiedostoon. Nimensä mukaisesti käyttäjä pystyy vaikuttamaan karttaan ilman ,että sivua tarvitsee ladata uudestaan. Staattisesta kartasta poiketen dynaaminen kartta sisältää funktion, joka kytkee kartan ominaisuuksia päälle tai pois päältä. Dynaaminen kartta myös latautuu vasta HTML-tiedostoa ajettaessa, mutta käyttäjälle tällä ei ole merkitystä.

```
let map;

function initMap() {
  map = new google.maps.Map(document.getElementById("map"), {
    center: { lat: -34.397, lng: 150.644 },
    zoom: 8,
  });
}
```

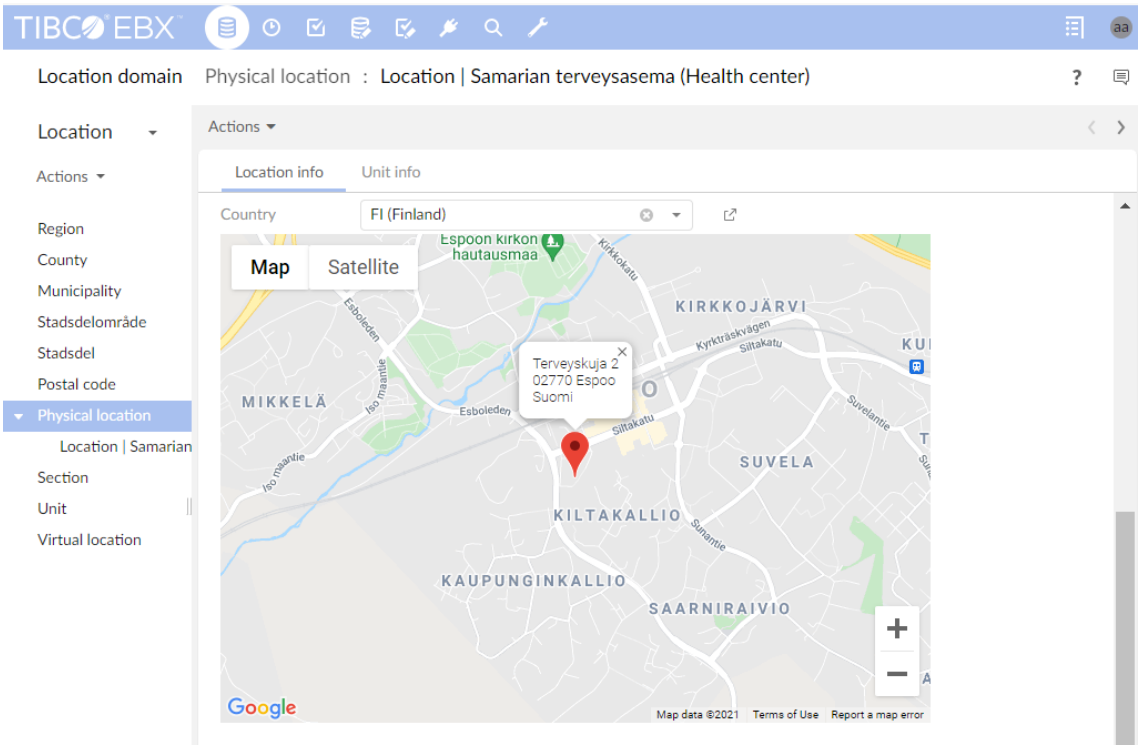
Esimerkkikoodi 5 Googlen dokumentaatiosta lainattu yksinkertainen dynaamisen kartan luontifunktio. [10.] Tämä koodi luo kartan annettuihin koordinaatteihin kaikilla Apin tarjoamilla ominaisuuksilla.

Dynaamista karttaa on muokattu rajaamalla sen ominaisuuksia käyttämisen yksinkertaistamiseksi ja lisäämällä karttapisteeseen rakenteen, joka näyttää pisteen katuosoitteen pistettä painamalla.

Dynaamisen kartan luova funktio rakentaa HTML- ja Javascript-koodin String-objektina, jonka se palauttaa. Tämä palautettu objekti liitetään osaksi HTML-tiedostoa, joka näytetään käyttäjälle.

```
if (validateParams(sbErrors))
     {
     //Creates the div for map
          jsMapOutput.append("<style>#mappe{min-height: 440px;min-
height: 440px;max-height:"+mapDimensions.height+"px;max-width:"+mapDi-
mensions.width+"px;}</style>");

          jsMapOutput.append("<div id=\"mappe\"></div>");

     //Creates a script for filling the map div
          String jsmap = "\n<script>" + "\nfunction initMap()" + "\n{ "
+ "\nvar options = { zoom:" + getZoomLevelIntValue() + ", cen-
ter:{lat:" + getLocation().lat + ",lng:" + getLocation().lng+ "},";

     jsmap += "};";

     //Javascript functions and variables for map generation
          jsmap += "\nvar map = new google.maps.Map(document.getEle-
mentById('mappe'), options);"
+ "\nconst contentString = "+InfowindowFiller(markerInfoTopicture)+";"
+ "\nconst infowindow = new google.maps.InfoWindow({content: content-
String});"
+ "\nconst marker = new google.maps.Marker({position:{lat:" + getLoca-
tion().lat + ",lng:" + getLocation().lng + "}, map});"
+ "\nmarker.addListener(\"click\", () => {in-
fowindow.open(map,marker);}); "
+ "\n}" + "\n</script>";
```

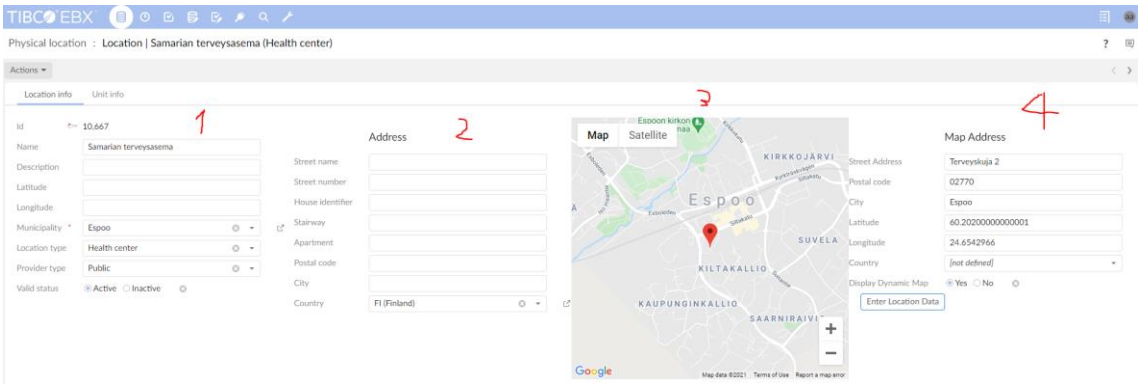
Esimerkkikoodi 6 Osa dynaamisen kartan rakentavaa funktiota



Kuva 7. Dynaamisen kartan näkymä EBX-järjestelmässä. Katuosoitteen sisältävän ikkunan voi avata tai sulkea. Ikkuna on sivua ladattaessa kiinni.

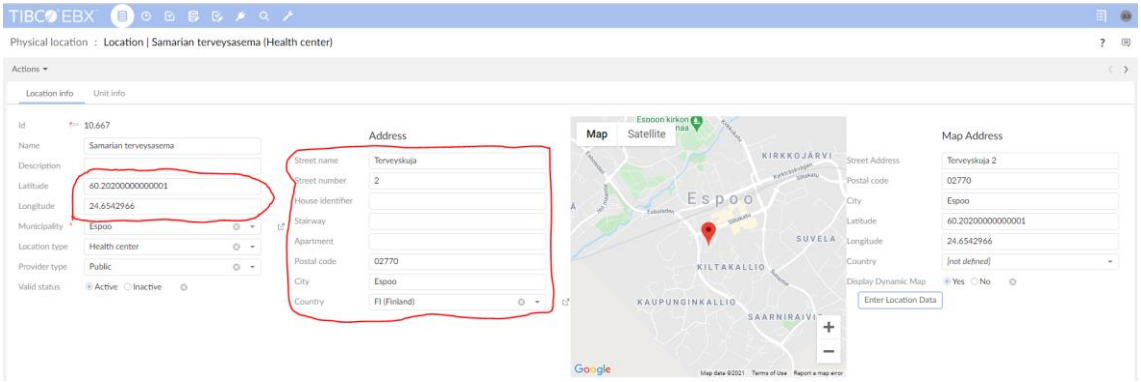
5.2.3 Karttadatan tallennus

Karttadatan tallennus rakentuu kolmesta osasta. Ensimmäisenä esimerkkikoodissa 2 esitelty funktio tekee geolokaatiohaun ja saa vastauksena geoCodingRes-nimisen JSON-taulukon. Seuraavaksi geoCodingRes JSON puretaan String arrayksi, joka sisältää halutut tiedot. Kolmanneksi puretun arrayn jäsenet jaetaan väliaikaiseen listaan ja HTML-sivuun lisätään nappula, jota painettaessa väliaikaisessa listassa olevat tiedot kirjoitetaan tietokannan tietojen päälle.



Kuva 8. Kuvassa datajäsenen tiedot ennen karttahaun tietojen tallentamista.

Kuvan 8 elementit 1 ja 2 ovat tietokantaan tallennettuja tietoja kohteesta. Elementti 3 on kartta, joka perustuu geolokaatiohakuun ja elementti 4 on karttahausta purettujen tietojen säiliö. Sinisellä ympäröity ”Enter Location Data” on nappi, joka siirtää datan väliaikaisesta säilöstä tietokantaan.



Kuva 9. Datajäsenen tiedot karttahaun tietojen tallentamisen jälkeen.

Geolokaatiohaun tulokset näin tallentamalla vältetään tilanne, jossa vahingossa ylikirjoitettaisiin tietokannan lokaatiodata uudella tai virheellisellä geolokaatiohaulla. Koska geolokaatiohaku tehdään kartan näyttämiseksi joka kerta uudestaan datajäsentä tarkasteltaessa, päätettiin projektin aikana lisätä jäsenen datamodel rakenteeseen väliaikainen datansäilytysosa, jonka ylikirjoittamisesta ei ole haittaa. Tämä ratkaisu on tehty suojaamaan datan laatua MDM:n näkökulmasta, sillä jos datajäsentä tarkasteltaessa ylikirjoitus suoritettaisiin automaattisesti, datan laatua ei voisi taata.

Datan siirtonappulaa painettaessa geoCodingRes JSON -objektista purettu data kirjoitetaan ensin JavaScript-muuttujiin. Tämä data tallennetaan tietokantaan EBX-järjestelmään rakennetun ebx_form_setNodeValue(String aPrefixedPath, Object aValue) JavaScript-funktion avulla. Funktio asettaa node-objektin arvon javascript-muuttujasta. [11.]

```
String addresCompStrings[] = parseGeoRes(geoCodingRes);

// insert values to model trough JS
w.addJS_cr();
w.addJS("var addressPartCountry = \"" + addresCompStrings[0] + "\";");
w.addJS_setNodeValue("addressPartCountry", countryTemp);
```

Esimerkkikoodi 7. Geolokaatiohaun tulosten tallentaminen javaScript muuttujiin.

```
String buttonJs = "ebx_form_setNodeValue(\"" + pathJScountry.format()
+ "\", addressPartCountry);\r\n"

UIButtonSpecJSAction mapsButton = new UIButtonSpecJSAction(UserMes-
sage.createInfo("Enter Location Data"), buttonJs);

mapsButton.setRelief(UIButtonRelief.EMBOSSED);
w.addButtonJavaScript(mapsButton);
```

Esimerkkikoodi 8. ”Enter Location Data” JavaScript -napin luominen.

Datan käsittely karttaa tarkastellessa täyttää MDM-järjestelmän vaatimukset ,sillä datajäsen on uniikki, eikä järjestelmään voi jäädä kopioita koodista. Datan lähde ja rajapinta eivät vaihdu, joten Googlen virhettä lukuun ottamatta data on turvassa saastumiselta.

Tähän elementtiin jää näistä toimenpiteistä huolimatta haavoittuvuus käyttäjän muodossa. Koska tämä elementti antaa käyttäjälle kyvyn muokata datajäsentä, ei sen datan laatua voi täysin turvata.

5.3 Datan käsittely workflow-rakenteessa

Kuten aiemmin on mainittu, yksi projektin pää tavoitteista luoda workflow, joka automaattisesti tekee datajäsenelle geolokaatiohaun ja tallentaa täydennetyt osoitetiedot tietokantaan. EBX-järjestelmä tukee joko täysin automaattisia tai käyttäjän toimintaa vaativia workflow-rakenteita. Projektia varten päätettiin kehittää automaattinen workflow, joka käy koko datasetin läpi.

Workflow saa geolokaatiodatan samalla tavalla kuin karttaelementti, mutta sen tallentaminen tapahtuu eri tavalla. Workflow käynnistyessään luo ensin kopion käsiteltävästä datasetistä ja välittää tämän datankäsittelyfunktiolle. Tämän jälkeen funktio käynnistää taustaprosessin ja hakee API-avaimen geolokaatiohakuja varten. Tämän jälkeen prosessi eristää datasetin listaksi

datajäseniä ja käy ne yksitellen läpi. Jokaisen jäsenen tiedoilla tehdään geolokaatiohaku ja tulokset tallennetaan prosessin kopioon tietokannasta. Kun prosessi on käynyt datasetin läpi, siirtyy workflow seuraavaan askeleeseen ja ylikirjoittaa vanhan datasetin muokatun kopion tiedoilla.

```
     final ProcedureResult result = service.execute(new Procedure() {
               public void execute(final ProcedureContext procedureCon-
text) throws Exception {

     AdaptationHome dataspaceForProcess = procedureContext.getAdapta-
tionHome();

     Adaptation Addressdataset = toDataset(dataspaceForProcess, da-
tasetname);

     AdaptationTable table = toTable(Addressdataset, tablePathName);

     List<Adaptation> elements = table.selectOccurrences(null);

     for (Adaptation unmodifiedRecord : elements)
     {

          final String searchTerm = selectBestSearchTerm(unmodi-
fiedRecord);

          GeocodingResult[] geoCodingRes = null;
          final String geoApiLanguage = "fi";
          GeoApiContext geoApiCtx = null;

          try {
               geoApiCtx = new GeoApiCon-
text.Builder().apiKey(api).build();
               geoCodingRes = GeocodingApi.geocode(geoApiCtx, search-
Term).language(geoApiLanguage).await();

          } catch (final Exception e) {
               System.out.println("geocoding failed apikey= " + api);
          }

          String parsedRes[] = parseGeoRes(geoCodingRes);

          ValueContextForUpdate AddressValueContext = procedureContext
          .getContext(unmodifiedRecord.getAdaptationName());

          AddressValueContext.setValue(parsedRes[0],MapAd-
dressPaths._Root_Property._Root_Property_Address_Country);


// modifiedRecord contains the changed values
          Adaptation modifiedRecord = procedureContext.doModifyCon-
tent(unmodifiedRecord, AddressValueContext);
          }

     }
});
     if (result.hasFailed()) {
          final OperationException exception = result.getException();

          throw exception;
     }
}
```

Esimerkkikoodi 9 Funktio, joka käy kaikki annetun datasetin jäsenet läpi ja tekee niille geolokaatiohaun ja tietojen täydennyksen.

Esimerkkikoodissa 9 ilmenevät objektit edustavat seuraavia asioita:

- AdaptationHome edustaa instanssia tietokannasta.
- Adaptation Addressdataset edustaa datasettiä, jota käsitellään.
- AdaptionTable table edustaa kaikkia datajäseniä, joita halutaan muuttaa.
- Adaption un- ja modifiedRecord edustavat datajäseniä ennen ja jälkeen tietojen muokkauksen.
- ValueContextForUpdate AddressValueContext on rajapinta, jonka kautta datajäsentä voi muokata,

Workflown datan tallennus on tehty käyttäen EBX-järjestelmän Adaptation-objekteja, jotka ovat read-only-esityksiä joko dataseteistä tai datajäsenistä [12.] ja ValueContextForUpdate-objekteja, jotka puolestaan mahdollistavat Adaptation objektin edustamien arvojen muuttamisen. [13.] Kuten esimerkkikoodissa näkyy, koko datasetistä valitaan jäsenet, joiden tietoja halutaan muuttaa ensin eristämällä ne AdaptationTable-objektiin ja sitten muuttamalla sen listaksi Adaptaatio-objekteja, jotka vastaavat datasetin jäseniä. ValueContextForUpdate-objektin kautta datajäsenen tiedot muokataan ja datasetin tiedot ylikirjoitetaan procedureContext.doModify()-funktiolla.

Tämä toimiva workflow saavuttaa kontekstissaan MDM-kehitysfilosofian ideaalitilanteen. Data tulee yhdestä lähteestä, yhden rajapinnan kautta ja yhdenmuotoisena. Koska prosessi on automaattinen, se ei ole haavoittuva käyttäjä virheelle.

Workflow-ratkaisussa piilee kuitenkin ongelma, jota ei voi automaatiolla täysin korjata. Niin sanottu GIGO (Garbage In Garbage Out) -ongelma ilmenee, jos geolokaatiohausta tuleva data on huonoa. Ihminen voi heti huomata, jos suomalainen sairaala on keskellä Australiaa, mutta järjestelmä ei ilman referenssidataa pysty samaa arviota tekemään.

6 Työn vaiheet

6.1 Työprosessi

Projekti aloitettiin marraskuussa 2020, mutta käyttöoikeuksien hankkimisen ja EBX-järjestelmään perehtymisestä johtuvien viivästyksien takia varsinainen projektityö saatiin aloitettua vasta joulukuussa.

Ensimmäinen työstetty asia projektissa oli dynaamisen kartan tuottaminen. 29.12.2020 saatiin kartan ensimmäinen versio toimivaksi ja alettiin muokkaamaan osoitetietojen datamodel-rakennetta, jotta siihen saataisiin väliaikainen datansäilytyselementti. 8.1.2021 saatiin datamodel oikeaan muotoon ja alettiin työstämään datan tallentamista, joka puolestaa valmistui 13.1. 14.1–18.1 käytettiin dynaamisen ja saattisen kartan valitsemiskyvyn tekemiseen. 19.1 eteenpäin aika meni projektin ulkonäön ja workflow-mallin tekemiseen, kunnes 29.1 piti työympäristö asentaa uudelleen tietokannan korruptoiduttua. Korruptoituneen tietokannan uudelleenrakentamisen jälkeen työ keskittyi workflown tekemiseen, kunnes projekti saatiin työnantajan puolesta valmiiksi 9.4.2021.

6.2 Ongelmat ja puutteet

Suurin ongelma projektissa tapahtui, kun tietokanta korruptoitui. Tämä johtui nykyisen ymmärryksen mukaan poistetuista WAR-paketeista. Jokin ongelma EBX-järjestelmän sisällä esti sen käynnistämisen, kun turhia tiedostopaketteja oli poistettu ympäristöstä. Vaikka paketit oli poistettu käytöstä järjestelmän sisältä, ei järjestelmä enää käynnistynyt, vaikka poistetut paketit lisättiin takaisin. Ongelma jouduttiin kiertämään Apache Tomcatin asetuksista, joissa estettiin turhien pakettien lataamisen järjestelmän käynnistyksessä.

EBX-järjestelmässä on kyky siirtää sisään ja ulos datamodel-rakenteita .xsd-tiedosto muodossa, joten olisi ollut järkevää varmuuskopioida sellainen juuri tämän tilanteen varalta.

Dokumentaatio itsessään oli myös hankaluus projektissa. Koska alkuperäinen ohjelmiston rakentanut tiimi oli ranskalainen ja yrityksen johto toimii Amerikasta, on järjestelmän dokumentaatio osittain vaikealukuinen. Dokumentaation JavaScript-osuus on esimerkiksi mielestäni oudosti kasattu, mikä koostuu näemmä lähinnä lähteestä 7.

Koodauksessa suurin ongelma oli Adaptaatio-objektien manipulointi ja tiedostopolkujen käyttäminen datamodel-rakenteessa. Esimerkiksi rakenteessa esiintyvien objektien (kuten ValueContext.... ja Adaptation) tapa käsitellä tiedostopolkuja poikkeaa toisistaan (ks. lähde 8). Adaptation-objektin absoluuttinen polku alkaa datasetin juuresta, paitsi jos se on datajäsen, se alkaa itsensä juuresta. Nämä ongelmat veivät monta työtuntia.

Kesken projektin oli myös tarkoitus tehdä erillinen css-tiedosto ulkonäön muokkaamiseen, mutta EBX-järjestelmän sisäisiä tyyliasetuksia ei saatu toimimaan yhdessä, joten css-tiedoston käyttö hylättiin, jotta workflow saataisiin nopeammin valmiiksi.

API-avaimen säilytystä varten lisätty .PROPERTIES-tiedostoa olisi voinut käyttää enemmän projektissa esimerkiksi dynaamisen kartan ominaisuuksien muokkaamiseen, mutta tätä ei nähty tärkeänä ominaisuutena, joten sen kehitystä ei jatkettu avaimen toiminnallisuuden takaamisen jälkeen.

Dynaamista karttaa olisi voinut kehittää pidemmälle esimerkiksi näyttämällä useamman kohteen kerralla. Tämä tosin olisi vaatinut suurta muokkausta siihen, miten käyttöliittymä näyttää datajäseniä.

7 Yhteenveto

Projektin tekninen päämääränä oli luoda kaksi ominaisuutta EBX-järjestelmään. Ensimmäinen eli datajäsenten visuaalinen esittäminen kartalla. Paikkatietojen täydentäminen onnistui mielestäni hyvin, vaikka karttaelementtiin olisi voinut korjata ja lisätä ominaisuuksia aiemmin projektin tuotantovaiheessa.

Toinen ominaisuus järjestelmään oli workflow-ratkaisun tekeminen, joka käy koko datasetin läpi täyttäen datajäsenten puuttuvat osoite- ja paikkatiedot. Tämänhetkisten tietojen mukaan komponenttiin ei saisi samaa toiminallisuutta paremmalla tavalla ja komponentti saavuttaa tavoitteensa.

Molemmat komponentit pystyttiin toteuttamaan MDM-teoriaa seuraten. Datajäsenet eivät voi tuottaa kopioita toisistaan ja geolokaatiohausta tuleva data on aina yhdenmukaista eikä altistu vaaroille komponenttien prosessien aikana. Käyttäjävirhe ja roskadatan mahdollisuus säilyvät, mutta niiden riski on pieni ja varsinaisesta järjestelmästä riippumaton.

Työnantajan vastaanotto projektin tuotoksiin oli positiivinen ja ensisijaiset tavoitteet saavutettiin ja muutama ylimääräinen toivottu ominaisuus saatiin implementoitua.

Opinnäytetyön aikana sain kehitettyä Java- ja JavaScript -osaamista sekä tutustuin EBX-järjestelmään, josta voi olla tulevaisuudessa hyötyä. Vaikka olen lopputulokseen tyytyväinen, jos voisin mennä ajassa taaksepäin, tekisin työn tehokkaammin sen aikana opituilla taidoilla.

Lähteet

1 Meistä, Enfo. 2021. Verkkodokumentti. <https://www.enfo.fi/meista>. Luettu 27.04.2021.

2 TIBCO EBX™ Software, TIBCO Software Inc. 2021. Verkkodokumentti. <https://www.tibco.com/products/tibco-ebx-software>. Luettu 27.04.2021.

3 Master Data Management: What It Is and Why It Matters, Informatica, 2019. Verkkodokumentti. <https://www.informatica.com/in/resources/articles/what-is-master-data-management.html> Luettu 07.05.2021.

4 Master Data Management (MDM): Help or Hindrance? , Knut Jürgensen, 16.05.2016. Verkkolähde. <https://www.red-gate.com/simple-talk/devops/database-devops/master-data-management-mdm-help-or-hindrance/> Luettu 07.05.2021.

5 4 Common Master Data Management Implementation Styles. Michael Lonnon. 25.05.2018. Verkkolähde <https://www.stibosystems.com/blog/4-common-master-data-management-implementation-styles> Luettu 07.05.2021.

6 What is Master Data Management (MDM)?. Tibco. 2021. Verkkolähde. 2021. <https://www.tibco.com/reference-center/what-is-master-data-management> Luettu 07.05.2021.

7 Introduction to dataspaces, TIBCO Software Inc. 2021.Verkkodokumentti. <https://docs.tibco.com/pub/ebx/5.9.5/doc/html/en/index.html?page=user_interface/coding_recommendations.html> Luettu 03.05.2021.

8 Maps Static API, Overview, Google 2021.Verkkodokumentti. <https://developers.google.com/maps/documentation/maps-static/overview> Luettu 03.05.2021.

9 Maps Javascript API, Overview, Google 2021.Verkkodokumentti. <https://developers.google.com/maps/documentation/javascript/overview> Luettu 03.05.2021.

10 Geocoding API, Overview, Google 2021.Verkkodokumentti. <https://developers.google.com/maps/documentation/geocoding/overview > Luettu 03.05.2021.

11 Interface JavaScriptCatalog, TIBCO EBX Documentation, Tibco 2019 <https://docs.tibco.com/pub/ebx/5.9.5/doc/html/en/Java_API/com/orchestranetworks/ui/JavaScriptCatalog.html#ebx_form_setValue-java.lang.String-java.lang.Object-> Luettu 04.05.2021.

12 Interface Adaptation, TIBCO EBX Documentation, Tibco 2019, <https://docs.tibco.com/pub/ebx/5.9.7/doc/html/en/Java_API/com/onwbp/adaptation/Adaptation.html#createValueContext--> Luettu 04.05.2021.

13 Interface ValueContextForUpdate, TIBCO EBX Documentation, Tibco 2019, <https://docs.tibco.com/pub/ebx/5.9.7/doc/html/en/Java_API/com/orchestranetworks/service/ValueContextForUpdate.html#setValueEnablingPrivilegeForNode-java.lang.Object-com.orchestranetworks.schema.Path-> Luettu 04.05.2021.