Long Hoang

AIS-järjestelmän reaaliaikatietojen esittäminen

Metropolia Ammattikorkeakoulu

IT-insinööri

Ohjelmistotekniikka

Opinnäytetyö

1.9.2017

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Long Hoang AIS-järjestelmän reaaliaikatietojen esittäminen 28 sivua + 5 liitettä 1.9.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Juha Kämäri

Insinöörityön tarkoituksena on ohjelmoida AIS-sovellus, jolla haetaan reaaliaikadataa Digitraffic REST -rajapinnalta ja esittää Suomen merialueella olevien alusten sijainnit SpatialWebin kartan tasolla. Sovellusta kehitetään Visual Studio -ympäristössä.

Dokumentin alussa kerrotaan lyhyesti AIS-järjestelmästä. Käydään läpi järjestelmän toiminnot, AIS-laitteet ja sen lisäksi, miten dataa lähetään alusten AIS-laiteilta tietokantaan. Sen lisäksi käydään läpi myös, mitä kaikkea teknologioita on käytetty sovelluksen kehitysvaiheessa. Sen jälkeen käydään sovelluksen kehitysvaiheita läpi koodiesimerkkeineen.

Jatkokehityksenä pyritään luomaan tietokanta, jonne tallennetaan haetut reaaliaikaiset sijaintidatat viimeisen päivän aikana. Tietokannan datat voi käyttää hyväkseen aluksen sijaintihistorian esittämiseen.

Avainsanat	AIS, REST-rajapinta, SpatialWeb, reaaliaikatieto

Author(s) Title Number of Pages Date	Long Hoang AIS Real Time Data on Map 28 pages + 5 appendices 1 September 2017
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Specialisation option	Software Engineering
Instructor(s)	Juha Kämäri, Senior Lecturer

The purpose of this thesis was to create an AIS application that retrieves real time data from Digitraffic's REST API and shows the locations of ships in the Finnish marine area on a SpatialWeb's map layer.

The AIS system is briefly described at the beginning of this document, going through system's functions, AIS devices, and how these devices send ships' real time data to the AIS database. Also, all the technologies used in the application's development are introduced.

Further development is to create a database where the location data of the previous days of the vessels is saved. These data can be used to show any vessel's location history.

Keywords	AIS, REST API, SpatialWeb, real time data

Sisällys

Liitteet

Liite 1. Lopputulos

Liite 2. Json-muoto

Liite 3. Tiedosto ”app.component.html”

Liite 4. C# malli

Liite 5. Angular malli

Lyhenteet

AIS	*Automatic Identification System*. Järjestelmä, jota käytetään alusten tunnistamiseen ja sijainnin määrittämiseen.
API	*Application Programming Interface*. Ohjelmointirajapinta, jonka avulla ohjelma tarjoaa muille ohjelmille ominaisuuksien käyttömahdollisuuden.
IMO	*International Marinetime Organization.* Kansainvälinen merenkulunjärjestö.
MMSI	*Maritime Mobile Service Identifier.* Aluskohtainen tunnistusnumero.
GPS	*Global Positioning System.* Maailmanlaajuinen paikallistamisjärjestelmä.
SOLAS	*Safety of Life at Sea.* Kansainvälinen yleissopimus ihmishengen turvallisuudesta merellä.
VTS	*Vessel Traffic Services.* Liikennevirasto.
VHF	*Very high frequency.* Radiotaajuusalue 30 MHz:stä 300 MHz:iin saakka.
ARPA	*Automatic Radar Plotting Aid.* Automaattinen tutkan piirtämistuki.

1 Johdanto

Lopputyö on toteutettu SITO Oy:n työharjoittelun aikana. Projektin tavoitteena on luoda sovellus, joka ottaa vastaan Digitraffic-palvelulta vesiliikenteessä olevien alusten reaaliaikatietoja ja esittää niitä Siton SpatialWeb -kartalla. Projektin backend -ohjelmointikielinä käytetään C# ja frontendissa Angular 2. Projektin lopputuloksena syntyy SpatialWebin karttataso, jossa näkyy alusten sijainnit reaaliaikaisesti kartalla, ja niiden lisätietoja esitetään pikatieto-ikkunassa, joka saa näkyviin aluksen ikonin kautta.

Tämän lopputyön dokumentin toisessa luvussa tutustutaan AIS-järjestelmään ja otetaan selvää, mitä ominaisuuksia sillä on tarjolla käyttäjille. Sen lisäksi otetaan selvää, mistä alusten datat tulevat, mitä hyötyjä datoista on ja mistä sovelluskehittäjät voivat saada datoja käyttöönsä. Kolmannessa luvussa kerrotaan, mitä kaikkea teknologiaa on käytetty hyödyksi tämän sovelluksen kehittämisessä. Sen lisäksi tutkitaan myös teknologian arkkitehtuuri ja käydään läpi, miten teknologia toimii yleisellä tasolla. Neljännessä luvussa katsotaan sovelluksen tiedostorakenne. Viidennessä luvussa käydään läpi koodiesimerkkien avulla, miten sovellus on kehitetty. Lopuksi katsotaan, mitä suunnitelmia on mietitty sovelluksen jatkokehitykseen ja miten niitä toteutetaan.

2 AIS

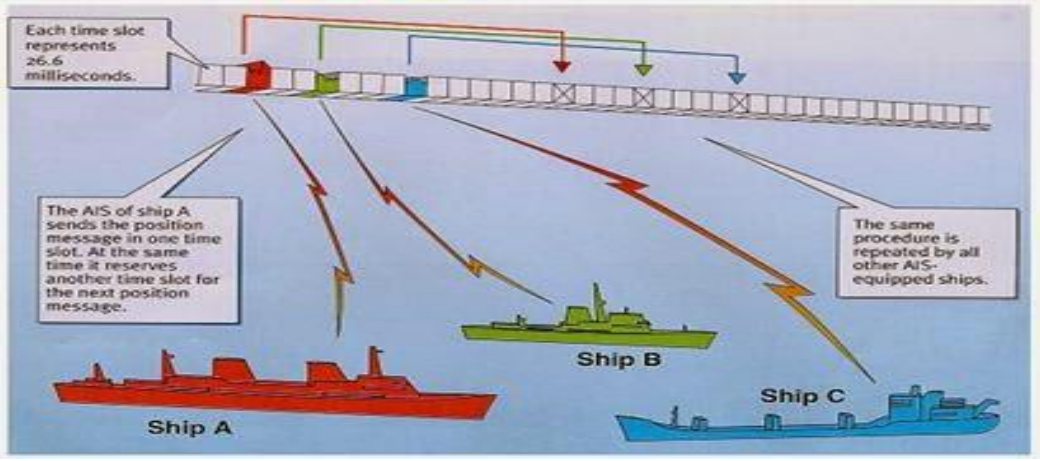
2.1 Mikä on AIS?

AIS-järjestelmä on Kansainvälisen merenkulunjärjestön (IMO) kehittämä merenkulun hienostunut radioteknologia, joka tarjoaa aluksille keinon vaihtaa elektronisesti navigointitiedot läheisten alusten ja maakeskuksien kanssa.

Vuonna 2000 Kansainvälinen merenkulunjärjestö hyväksyi uuden vaatimuksen, jonka mukaan kaikilla aluksilla on oltava AIS-järjestelmä pystyäkseen tarjoamaan aluksen tietoja automaattisesti muille aluksille ja rannikkoviranomaisille. IMO:n SOLAS-sopimus edellyttää AIS-järjestelmän asennusta kaikkiin aluksiin, jotka ovat kansainvälisellä matkalla ja bruttovetoisuus on 300 tonnia tai enemmän, ei-kansainvälisellä matkalla ja bruttovetoisuus on 500 tai enemmän ja sen lisäksi kaikki matkustaja-alukset kokoluokasta riippumatta. Vaatimus astui voimaan joulukuussa 2004 [1].

2.2 Kuinka AIS toimii

A-luokan AIS-laitteet koostuvat yhdestä VHF-lähettimestä, kahdesta VHF-vastaanottimesta ja yhdestä VHF-digitaaliselektiivikutsulaitteesta. AIS-laitteen toiminta tapahtuu tavallisesti itsenäisesti ja tauottomassa tilassa riippumatta aluksen tilasta. Lähetykset tapahtuvat kahdella taajuudella häiriöiden välttämiseksi ja mahdollistaakseen kanavien vaihdon ilman yhteyskatkoja muihin laivoihin.



Kuva 1. Järjestävä aikajakokanavointi

AIS-laitteet käyttävät itsejärjestävää aikajakokanavointia (SOTDMA) ilmoittaakseen ennalta muille aikavälin, jota laite aikoo itse käyttää asemointiraporteissa, eivätkä käytä muiden AIS-laitteiden varattuja aikavälejä (ks. Kuva 1). Nämä tietoliikennesolut kattavat yhdeksän "yhden minuutin aikakehykset", joista kukin kehys koostuu 2 250 26,6 millisekunnin aikaväliä radiotaajuuskanavaa kohden [1].

AIS -lähetin lähettää automaattisesti tietoa aluksesta, kuten sijainnin, nopeuden, kääntymisnopeuden ja navigointistatuksen säännöllisin väliajoin VHF-lähettimen kautta. Aluksen tieto on peräisin laivan navigointilaitteista, yleensä satelliittipaikannusvastaanottimesta ja hyrräkompassista. Aluksen nopeudesta riippuen näitä tietoja lähetetään 2-10 sekunnin välein ja kolmen minuutin välein olleessa ankkuroituna. Sen lisäksi muut tiedot, kuten MMSI, aluksen nimi ja VHF-kutsumerkki, ohjelmoidaan AIS-laitteen asennuksen yhteydessä, ja niitä lähetetään säännöllisesti kuuden minuutin välein [3].

2.3 AIS-järjestelmän käyttötarkoitus

Navigoidessa merellä on tärkeää, että aluksella on tarpeeksi tietoja ympäristöstään, jotta navigoija pystyy tekemään päätöksiä välttääkseen yhteentörmäystä muiden alusten kanssa tai muita vaaroja. Tähän tarkoituksen on perinteisesti käytetty erilaisia tähystystekniikoita, äänimerkkejä ja VHF-radiota sekä ARPA-toiminnolla varustettua tutkaa. Kuitenkaan tutkan näytöllä ei pystytä tunnistamaan kohteita tarkemmin, ja muut ympäröivien alusten liikkeiden ja reaktioiden tarkkailuun liittyvät rajoitukset estävät joskus yhteentörmäyksen välttämiseksi tehtävän toimenpiteen suorittamisen ajoissa. Parantaakseen navigoinnin turvallisuutta AIS:sta vastaanotetut tiedot voidaan lisätä aluksen tutkaan tai karttaplotteriin, jossa voi esittää esim. muiden alusten sijainnin lisäksi myös niiden nimi, etäisyys ja kutsumerkki. Vilkkailla meri- ja satama-alueilla rannikkoviranomaiset voivat hyödyntää AIS:sta saatuja tietoja laivaliikenteen valvomiseen.

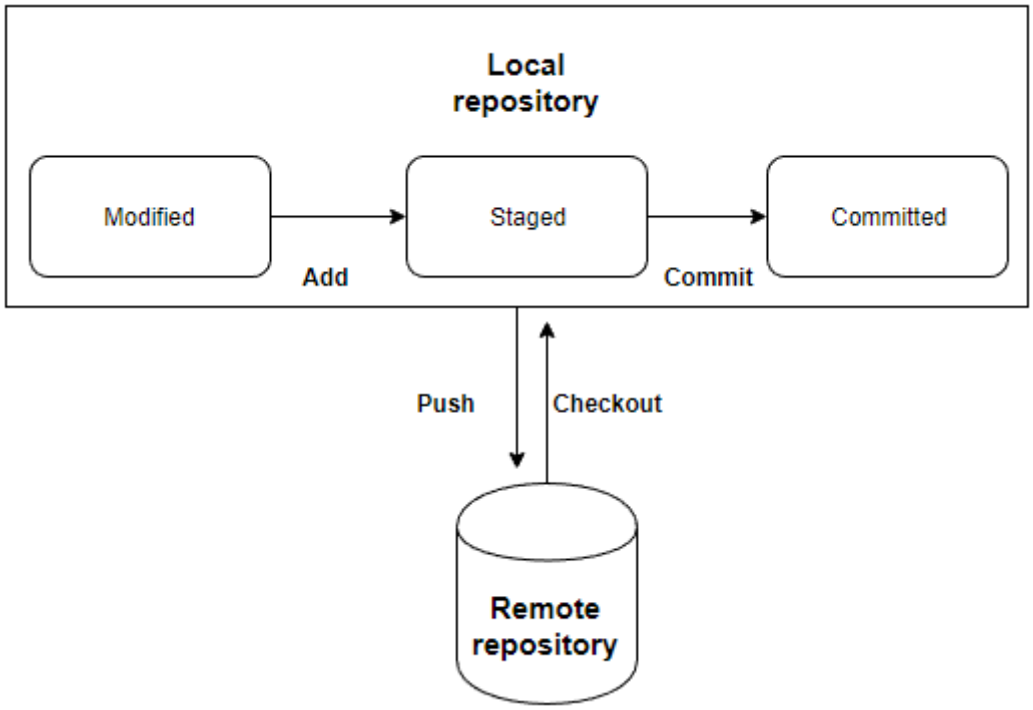
Vaikka AIS :n käyttöönotto onnistui menestyksekkäästi, maapallon kaarevuuden takia AIS -lähetysalue on rajoitettu noin 15-20 meripeninkulmaa useimmissa olosuhteissa. Lähetysalueen suuruus voi vaihdella riippuen topologiasta, alusten tiheydestä alueella ja ilmakehän olosuhteesta [4].

3 Käytetyt teknologiat

3.1 Git

Versionhallinta on työkalu, joka auttaa hallitsemaan koodin muutoksia ajan myöten. Versiohallinta tallentaa muutokset tiedostoon tai järjestelmään, josta sovelluskehittäjä voi hakea vanhat versiot tarvittaessa. Versionhallinta helpottaa ryhmässä työskentely seuraamalla yksittäisiä muutoksia ja estämällä ryhmän jäsenten samanaikaisia ristiriitaisia muutoksia.

Yksi nykyajan käytetyimmistä versionhallintasysteemeistä on Git, jonka Linus Torvalds on kehittänyt vuonna 2005. Gitin suurin ero muista versionhallintasysteemeistä on datan käsittelytapansa. Git ei säilytä muutokset tekemällä niistä tiedostonpohjaisia listoja. Sen sijaan jokaisen tallennuksen yhteydessä Git ottaa tiedostoista ”kuvankaappauksen”, miltä tiedostot näyttävät sillä hetkellä ja tallentaa sen jälkeen viitauksen kyseisen version kuvankaappaukseen. Jos tiedosto ei ole muutettu, Git luo vain linkin edelliseen identtiseen tiedostoon, joka on jo tallennettu Gittiin [5].



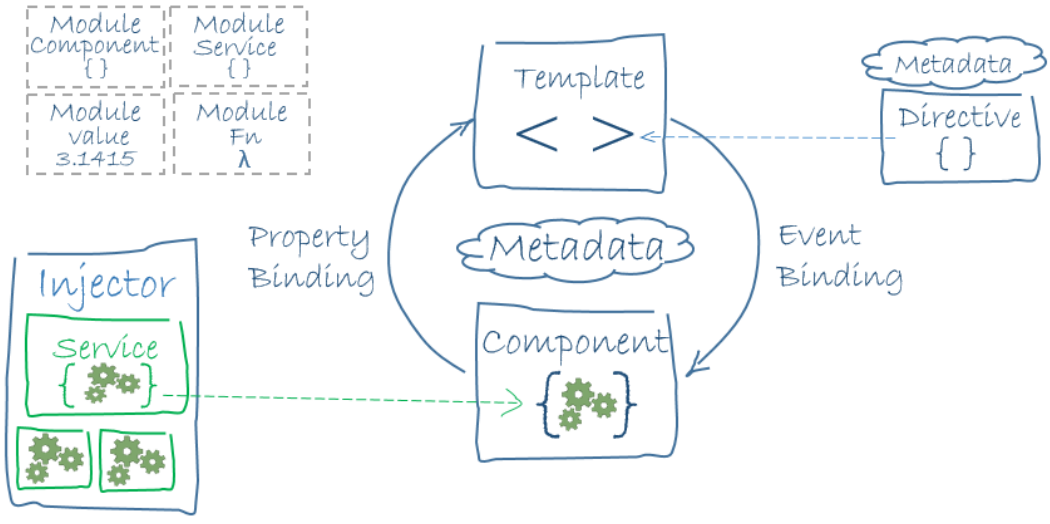
Kuva 2. Git -arkkitehtuuri

Gitilla on kolme keskeistä tilaa, joissa paikalliset tiedostot voivat sijaita: muokattu, lavastettu ja sidottu. Muokattu -tilassa Git tunnistaa, että tiedostoissa on tehty muutoksia, mutta niitä ei ole vielä tallennettu tietokantaan. Muokatut tiedostot lavastetaan komennolla ”Add”, jonka jälkeen tiedostot on lavastettu tilassa eli niitä merkataan nykyiseen versioon sidotettavaksi. Lopuksi turvallisesti sidotaan tiedostot ”commit”-komennolla paikalliseen tietokantaan (ks. Kuva 2).

Perinteisesti ryhmän työt edistetään siten, että jokainen ryhmän jäsen luo oman haaran, jossa tehdään omia muutoksia paikallisesti ja sitomisen jälkeen lähettää niitä etäpalveluun ”push”-komennolla. Uuden haaran tulee aina tehdä päähaaran viimeisimmästä versiosta. Etäpalvelussa voidaan yhdistää haarassa tehdyt muutokset päähaaraan tekemällä ”pull request” eli pyydetään ryhmän jäsenet testaamaan haaran toimivuutta ja hyväksyä yhdistämispyynnön. Ennen ”pull request”:in teko, Git tarkistaa päähaaran ja yhdistettävän haaran ristiriitaisia tilanteita ja pyytää käyttäjää ratkaisemaan niitä.

3.2 Angular 2

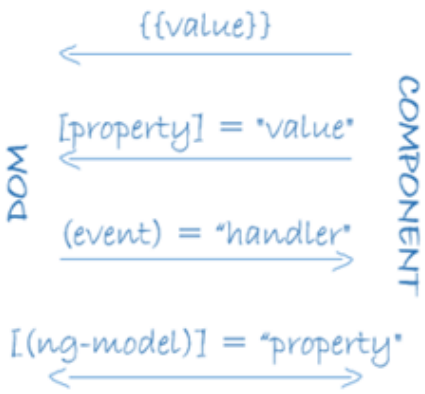
Angular 2 (ts. Angular) on Googlen ylläpitämä avoin lähdekoodiohjelmointikehys, joka on kokonaan uudelleen kirjoitettu edeltäjästään, AngularJS. Angularia käytetään websovelluksen kehittämiseen ja kehityskielenä toimii JavaScript, Dart tai TypeScript. Angular on tullut suosituksi ”data binding”-ominaisuutensa takia, mikä helpottaa sovelluksen ja käyttäjän vuorovaikutusta huomattavasti [6].



Kuva 3. Angular 2:n arkkitehtuuri

Angular-sovelluksen toiminta (ks. Kuva 3) alkaa kirjoittamalla HTML-malleja (template) Angularin merkeillä ja komponenttiluokkia (component) näiden mallien hallintaan, lisäämällä sovelluslogiikkaa palveluihin (service), lopuksi vie komponentit ja palvelut moduuleihin (module). Sovellusta ajetaan käynnistämällä juurimoduulia (bootstrapping), jonka jälkeen Angular ottaa sovelluksen haltuun ja esittää sen sisältöä selaimessa käyttäjille.

Jokaisella Angular-sovelluksella on vähintään yksi NgModule-luokka, juurimoduuli, yleisesti nimeltään AppModule. Kaikki moduuliluokat täytyy määrittää luokan alussa ”@NgModule”-sisustajalla. Moduuli on mekanismi, jolla Angular jäsentää komponentteja, ohjeita ja palveluita, jotka liittyvät toisiinsa siten, että niitä voidaan yhdistää muiden moduulien kanssa sovelluksen luomisen yhteydessä. Pienissä sovelluksissa on yleensä vain juurimoduuli, mutta isommissa sovelluksissa voi esiintyä monta ominaisuusmoduuliluokkia.



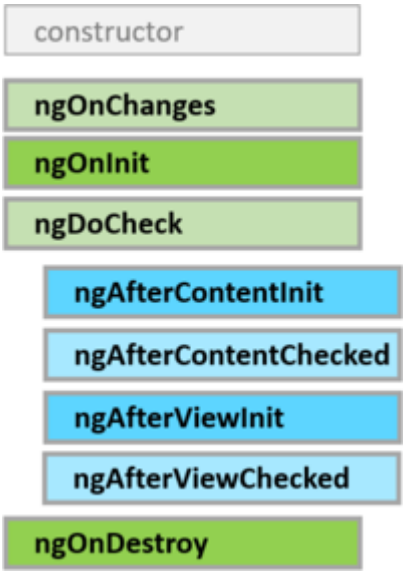
Kuva 4. Data-binding -syntaksien muotoja

Ilman kehityskehystä sovelluskehittäjä on vastuussa sovelluksen arvojen viemisessä HTML-malleihin ja käyttäjän vastausten muuntaminen toimintoihin ja arvopäivityksiin. Tällaiset push/pull-logiikkakoodit ovat vaikealukuisia, josta voi syntyä paljon virheitä. Angularin ”data-binding”-mekaniikalla pystyy yhdistämään HTML-mallin ja komponentin osia, lisäämällä HTML-malleihin sitomismerkkejä (ks. Kuva 4). Angularilla on neljä data-binding syntaksimuotoa, josta jokaisella on oma datan siirtosuuntansa (DOM:sta komponenttiin, komponentista DOM:in tai molempiin suuntiin).

Angular-komponentteja pyritään pitämään mahdollisimman yksinkertaisena, eli komponenteissa vältetään datan hakupalvelimelta, datan muodon käsittelyä ja lokitietojen kirjaamista konsolille. Tällaiset toiminnot kehotetaan suorittamaan Angular-palveluissa (service), joita injektoidaan komponentin käyttöön. Samaa palvelua voidaan injektoida

moniin komponentteihin, jolla vältetään koodin toistumista ja helpottaa koodin lukua. Kaikki Angular-komponentit noudattavat tiettyä elinkaarta (ks. Kuva 5). Angular luo ensin komponenttia, esittää sen, luo ja esittää komponentin lapsia, tarkistaa datan muutoksia ja tuhoa komponentin ennen sen poistoa DOM:sta.

Angular antaa kehittäjille kykyä vaikuttaa komponentin toimintaan tarjoamalla näkyvyyttä sen elinkaaren keskeisimpiin hetkiin elinkaarikoukkujen avulla. Koukut ovat valinnaisia, joita käytetään vain tarvittaessa. Näiden koukkujen avulla kehittäjät voivat halutessa ohjata komponentin toiminnan järjestystä halutulla tavalla.



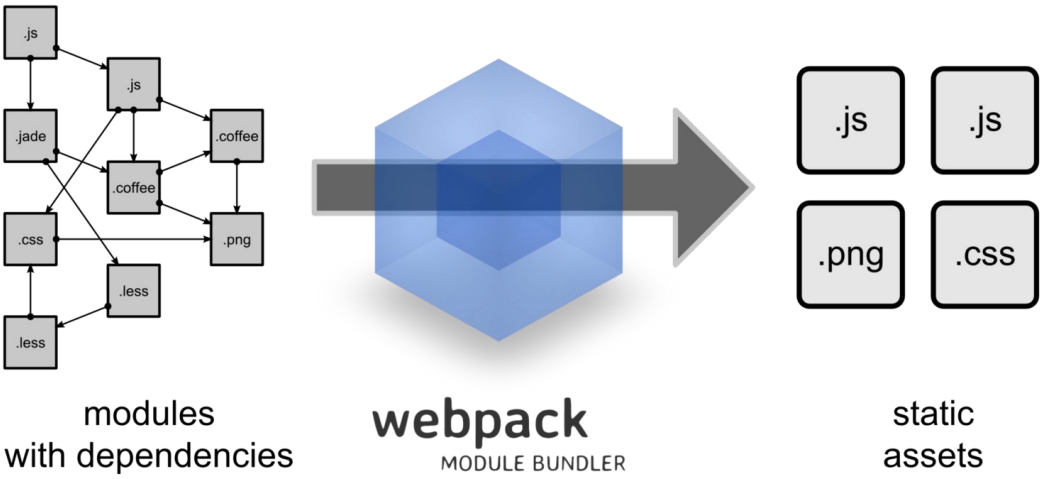
Kuva 5. Angular-komponentin elinkaaren koukut

Kaikilla Angularin tarjoamilla elinkaaren koukuilla (hooks) on etuliitteenä ”ng”, esim. OnInit-funktio, joka syntyy komponentin luomisen yhteydessä, on nimeltään ngOnInit. Näitä koukkuja voi hyödyntää haluttaessa ohjata sovelluksen suorittavan tiettyä koodia jonkin komponentin elämänvaiheen aikana [7].

3.3 Webpack

Webpack on moduulipaketti nykyaikaisille JavaScript-sovelluksille. Rakentaessa sovellusta webpack luo rekursiivisesti tiedostojen riippuvuussuunnitelman, joka sisältää sovelluksen kaikki tarvittavat moduulit. Webpack pakkaa suunnitelman mukaisesti kaikki

moduulit pienikokoisiin bundle.js-tiedostoihin (yleensä yhteen tiedostoon) selaimen ladattavaksi (ks. Kuva 6). [8]



Kuva 6. webpackin toimintaperiaate

Toinen tärkeä ominaisuus webpackissa on loaderit, jotka tarjoavat sovelluskehittäjille mahdollisuuden luoda omia tehtäviä, joita halutaan loaderin suorittavan webpackin yhdistäessä tiedostot yhteen. Loaderin avulla voidaan muuntaa eri kieliset tiedostot JavaScript -tiedostoksi. Loaderin avulla voi jopa tuoda CSS -tyylitiedostoja suoraan JavaScript -moduuleista (ks. Koodiesimerkki 1).

```
module: {
   rules: [
     {
       test: /\.css$/,
       use: [
         { loader: 'style-loader' },
         {
           loader: 'css-loader',
           options: {
             modules: true
           }
      ...
```

Koodiesimerkki 1. loaderin käyttö

3.4 Digitraffic

Digitraffic on Liikenneviraston palvelu, jonka kautta saadaan ajantasaista liikennetietoa Suomen tieverkolta, rautatie- ja vesiliikenteestä. Digitraffic käyttää tiedonlähteinä Liikenneviraston matka-aikatietopalvelua sekä liikenteen automaattisia mittauspisteitä (LAM), tiesääasemien, kelikamerakuvia sekä Tieliikennekeskuksen häiriötiedotteita. Vesiliikenteen reaaliaikaisen sijaintitietojen lähteenä on AIS-rajapintaa. Vesiliikenteestä on tarjolla reaaliaika sijaintitietojen lisäksi myös merivaroituksista, jonka tiedonlähteenä Pookin tietokanta [9].

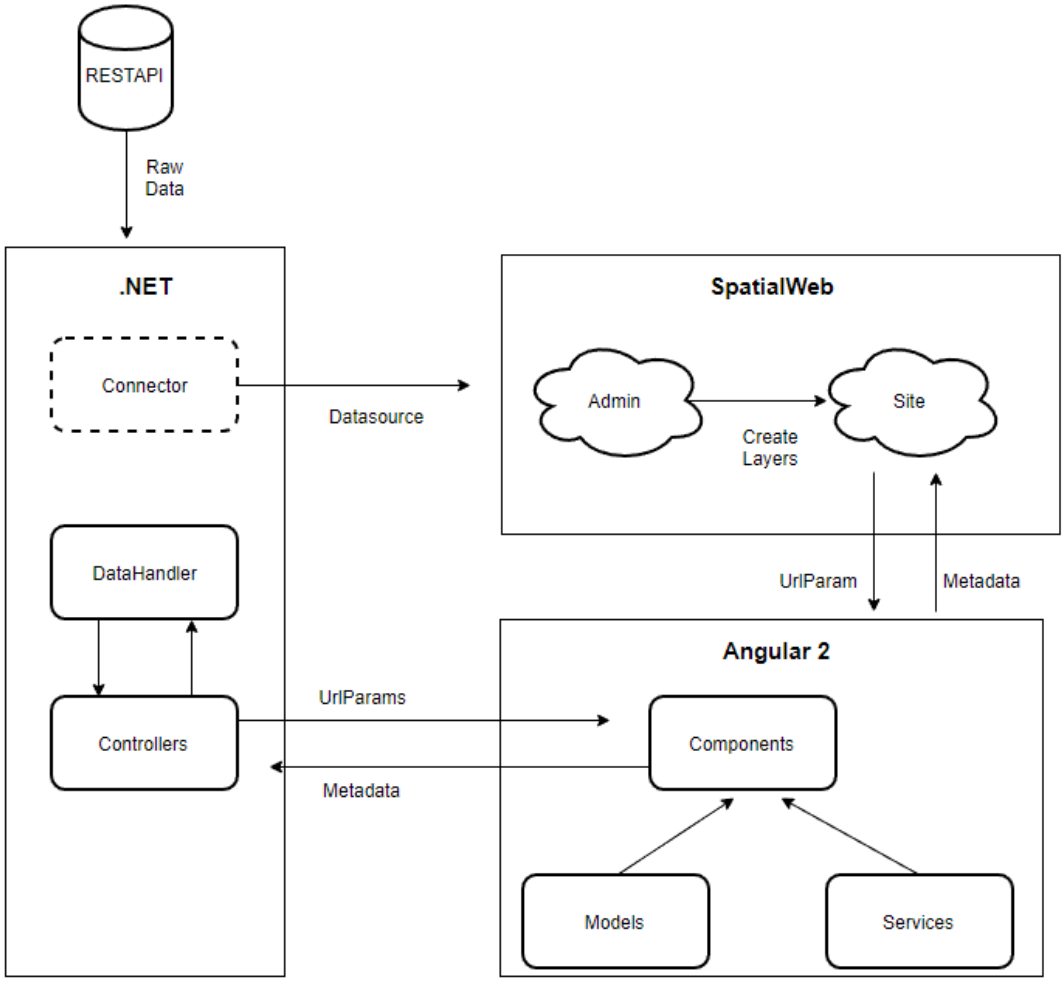
3.5 SpatialWeb

SpatialWebin käyttöliittymä koostuu kahdesta sivusta, admin ja site. Admin -käyttöliittymä on selainpohjainen työkalu SpatialWeb-karttasovellusten konfigurointia varten ja sillä on oma www-osoite. Kaikki SW7-palvelussa käytettävät aineistot sekä suuri osa palvelun ulkoasuun ja käyttäjiin liittyvistä asetuksista konfiguroidaan Admin-käyttöliittymällä. Admin-käyttöliittymä tallentaa tiedot SpatialWebin konfiguraatiotietokantaan. Lisäksi joitakin tietoja luetaan palvelimen hakemistosta. SpatialWebin site-käyttöliittymä on karttapalvelu, jonka päälle voidaan rakentaa erilaisia sovelluksia tasoina. Palvelussa voi liikkua vapaasti eri karttaikkunoiden ja hakutoimintojen välillä, vaihtaa kartta-aineistoa, vaikuttaa kartalla näkyviin tietoihin sekä käyttää erilaisia karttatoimintoja.

SpatialWeb lataa tasoja ja niiden dataa tiilenä, eli koko ruudun näkymä ei ole yksi iso karttatiili vaan sitä jaetaan moniin pienempiin tiiliin zoomiportaiden mukaisesti ja niitä ladataan yksi kerrallaan satunnaiselta vaikuttavassa järjestyksessä. Käytön mukaan SpatialWeb hakea ensisijaisesti selaimen välimuistista tarvittavat rasterikuvat ja puuttuvat kuvat pyydetään tietolähteestä. Tietolähteestä saadut rasterikuvat tallennetaan aina automaattisesti pyynnön jälkeen välimuistiin seuraavaa käyttökertaa varten. Käyttäjälle tietolähteen hitaus näkyy valkoisina tiilinä, jotka täyttyvät aikanaan. SpatialWeb ei kuitenkaan lataa koko kartan kerrallaan, vaan zoomiportaiden mukaan kartta lataa vain ne tiilet, jotka ovat kuvaruudussa näkyvillä.

4 Sovelluksen arkkitehtuuri

Sovelluksen Connectorissa voi hakea parametreista muodostetun URL-osoitteen avulla alusten sijaintitietoja reaaliaikaisesti. Connectorin dll-tiedoston avulla saadaan admin-sivulle luotu tietolähteen (Datasource), jota käytetään SpatialWebin tason luonnissa. Haettujen koordinaattien avulla SpatialWeb osaa sijoittaa tasolle määritetyt ikonit oikealle paikalle. Kun valitaan karttatason ikoneja, voidaan välittää Angularille valitun aluksen mmsi-numeron parametrina URL-osoitteen kautta ja hakea rajapinnalta sen metadataa tulostettavaksi pikatietoikkunaan.



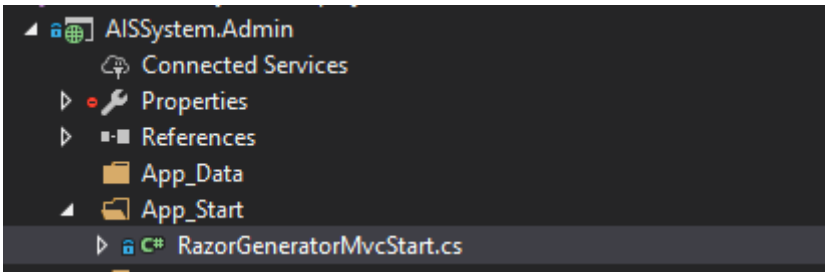
Kuva 7. Sovelluksen arkkitehtuuri

Sovellus on jaettu eri projekteihin MVC-mallin mukaisesti, joista jokainen projekti luo omasta osa-alueestaan dll-tiedostoja SpatialWebin käyttöliittymien varten. Sovellus on jaettu kahteen osaan, backend ja frontend. Backend-osan hoitavat Core- ja Connector-projekti, Site-projekti hoitaa frontend-osan. Projektien nimet eivät tarvitse olla täsmälleen

sama kuin yllä mainitut projektit, mutta on toivottavaa, että projektin nimi tulisi kuvata projektin toiminta tai osa-aluetta, esimerkiksi ”Admin” -projektissa tulee sisältää admin –käyttöliittymän liittyvät koodit ja ”Site” -projektissa site –käyttöliittymän.

4.1 AISSystem.Admin

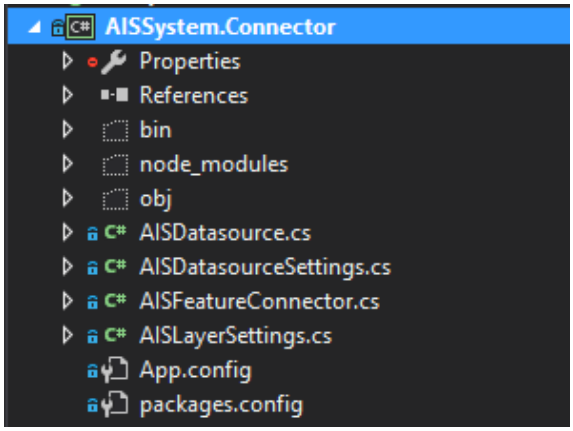
Admin-projekti luo MVC-mallin mukaisesti Razor-sivupohja, jonka avulla SpatialWeb osaa generoida admin –käyttöliittymälle tasojen luontivaiheessa tarvittavia lomakkeita. Admin-projekti ei varsinaisesti tarvita tässä sovelluksessa, mutta MVC-mallin noudattaen on hyvä luoda perussivupohjan tiedostolla $RazorGeneratorMvcStart.cs$.



Kuva 8. Admin-projektin tiedostorakenne

4.2 AISSystem.Connector

Connector-projekti sisältää sellaiset tiedostot, jotka tarjoavat admin-liittymälle tietolähteitä ja tason asetuksia.



Kuva 9. Connector-projektin tiedostot

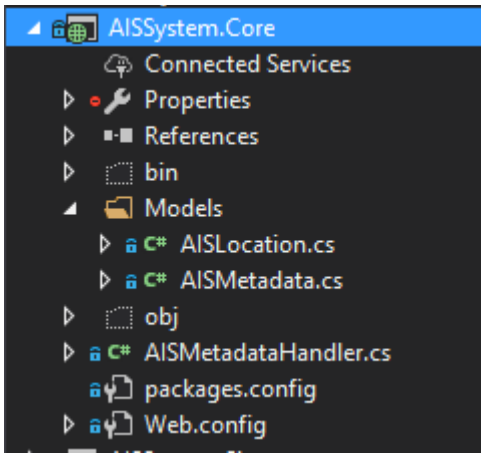
$Datasource.cs$ -tiedostossa määritetään tietolähteen perustietoja, kuten lähteen nimi, versio ja asetukset, jotka ovat selkeyden vuoksi jaettu omaan $DatasourceSettings$ -tiedostoon.

$FeatureConnector$ on yksi tärkeämmistä sovelluksen tiedostoista, joka sisältää kaikki tietolähteen toiminnat. Tässä sovelluksessa $FeatureConnector$ hoitaa sijaintidatan haku, käsittely ja sijoitus SpatialWebiin.

$LayerSettings$ -tiedostossa voi luoda mukautettuja asetuksia helpottaakseen tason luontivaihetta ja antaa pääkäyttäjälle enemmän vapauttaa tason määrittämiseen.

4.3 AISSystem.Core

Core-projektissa on sellaiset tiedostot, jotka liittyvät datan muotoihin ja sen käsittelyyn.



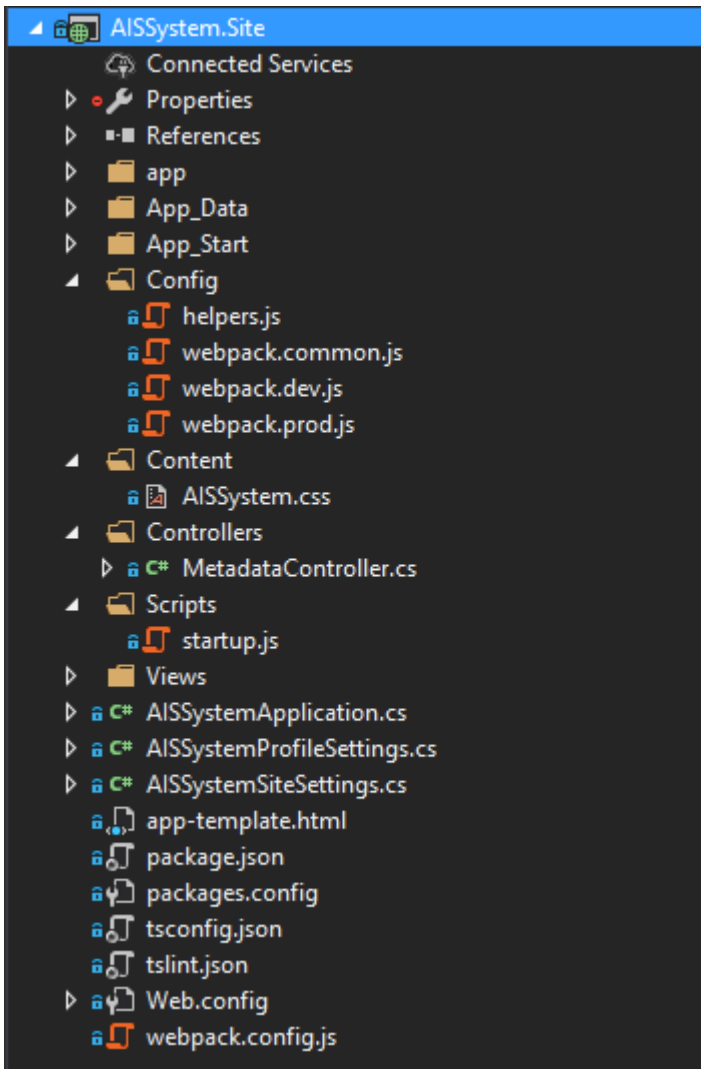
Kuva 10. Core-projektin tiedostorakenne

$Models$-kansiossa sijoitetaan sovelluksen backend datan mallit (ks. liite 4). Mallit ovat sovelluksen osia, joiden avulla pystytään käsittelemään rajapinnan ja käyttöliittymän välistä tietoja.

$MetadataHandler$ -tiedosto käsittelee alusten metadatojen haku, tallennus ja päivitystä tarvittaessa. Metadatan haku ei haluta sijoittaa samaan projektiin kuin sijaintidatojen haku, koska sijaintidataa haetaan jatkuvasti, mutta metadataa halutaan vain hakea tarvittaessa. Tiedostorakenteen selkeyden vuoksi on myös hyvä erotella näitä toisistaan.

4.4 AISSystem.Site

Site-projekti sisältää Angular-sovelluksen lisäksi kaikki sellaiset tiedostot, jotka ovat tekemisessä SpatialWebin site-käyttöliittymän kanssa.



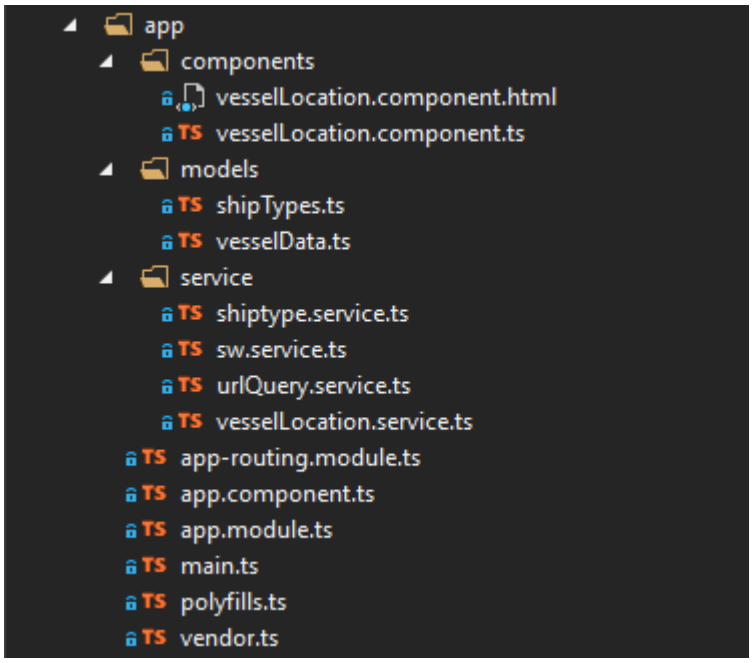
Kuva 11. Site-projektin tiedostorakenne

Tärkeämmät tiedostot ovat *app*-kansion tiedostot, *MetadataController.cs* ja *AISSystemApplication.cs*.

app-kansio sisältää kaikki Angular-sovelluksen tiedostot. Kontrolleritiedoston *MetadataController* avulla käsitellään backendin ja frontendin väliset datan siirrot. *AISSystemApplication* -tiedostossa haetaan mukautettuja asetuksia, jotka on määritetty admin-liittymässä käytettäväksi *startup.js*-tiedostossa.

4.4.1 Angular-sovellus

Angular-sovellus on pyritty toteuttaa ”hyvän käytännön” mukaisesti. Datan haku ja käsittely tehdään palveluissa, joita otetaan käyttöön komponenteissa injektoimalla (ts. kutsumalla palvelun metodia). Datoille luodaan omat mallit, jolla helpottaa datan käsittelyä ja esittämistä käyttöliittymällä.



Kuva 12. Site-projektin Angular-tiedostot

Komponentit hoitavat tällä tavalla vain palveluiden kutsut ja datan tulostus käyttöliittymälle. Komponentin HTML-mallille luodaan HTML-tiedoston, jonka URL-polun lisätään komponentin ominaisuuksiin käytettäväksi. HTML-mallissa määritetään datan tulostuksen muotoa.

Tiedostoon $app.module.ts$ tuodaan kaikki moduulit, joita tarvitaan sovelluksen toimintaan. Tiedostolla $vendor.ts$ tuodaan ulkoiset tiedostot (esim. tyylitiedostot) sovelluksen käyttöön.

5 Sovelluksen kehittäminen

5.1 Haun URL-osoitteen muodostaminen

REST-rajapinta palauttaa kehittäjille dataa, kun riittävät esitiedot toimitetaan haluttuun kyselyyn ts. kyselyn varten täytyy luoda rajapinnalle hakuosoite. Rajapinnalta halutaan hakea dataa tietyn väliajoin ja päivittää näin alusten sijainnit kartalla. Olisi hyvin raskasta muodostaa manuaalisesti URL-osoitteen jokaisen tiilen alueen haun varten. Ongelman ratkaisuksi haetaan tarvittavat tiedot SpatialWebin kautta ja muodostetaan niiden avulla dynaamisen URL-osoitteen, jota päivitetään automaattisesti ennen jokaista kyselyä. Osoitteen luomiseen tarvitaan seuraavat tiedot:

- alueen keskipisteen leveyspiiri
- alueen keskipisteen pituuspiiri
- haetun alueen säde
- aikaleima, mistä lähtien haetaan alusten tietoja.

SpatialWebin karttanäkymän lataustapaa voidaan käyttää hyväksi sijaintidatan hakuun ja päivittämiseen. Jokaisen kartan tiilen latauksen yhteydessä voi hakea rajapinnalta reaaliaikaiset sijaintidatat aluksilta, jotka ovat ilmoittaneet viimeisimmästä sijainnistaan viimeisen 24 tunnin aikana.

Haettavan alueen keskipisteen saa SpatialWebin tiilen datasta. Jokaisella tiilellä on $envelope$-tyyppinen muuttuja, joka sisältää tiilen omat sisällönkuvaustiedot. Muuntamalla kyseistä muuttujaa geometria-tyyppiseksi muuttujaksi saadaan selville tiilen keskipistettä muodossa (x^0, y^0) ja sen lisäksi myös tiilen $maxX$- ja $maxY$-arvo. Tarvittavat leveys- ja pituuspiirin arvot saadaan suoraan tiilen keskipisteen koodinaatista, koska sen koordinaatti ilmoitetaan maapallon asteina.

```
//Muuntaa envelope geometry:ksi
IGeometry geom = (new GeometryFactory()).ToGeometry(env);

//Haetaan keskipisteen x ja y arvot
double centroidX = geom.Centroid.X;
double centroidY = geom.Centroid.Y;
```

Koodiesimerkki 2. Keskipisteen määrittäminen

Rajapinta palauttaa kyselyssä alusten datat, jotka sijaitsevat annetusta pisteestä tietyn säteen sisällä. Koska karttatiili on neliön muotoinen, täytyy matemaattisesti laskea sellaisen ympyrän säteen, jonka pinta-ala peittäisi koko tiilen. Pienin mahdollinen ympyrän halkaisija, joka peittäisi koko neliön, on saman pituinen kuin sen peittämän neliön lävistäjän. Neliön lävistäjä d saadaan kaavasta:

$$d = \sqrt{a},$$

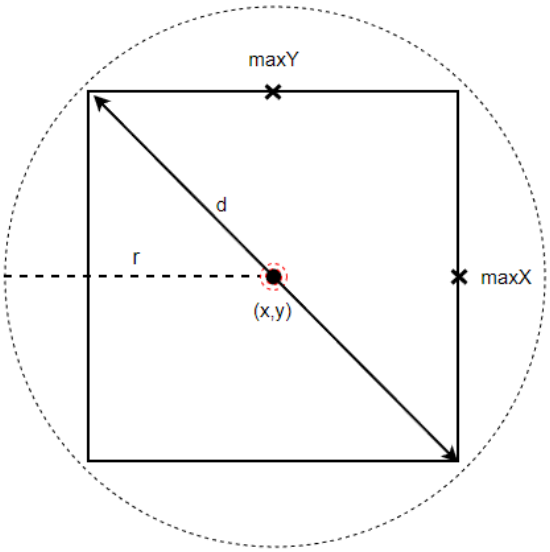
josta a on neliön sivun pituus. Säteen tarkkuudella ei ole sovelluksen kannalta iso merkitystä, joten ei käytetä C# tarjoamia matemaattisia neliöjuuri-funktiota, jotka saattavat hidastaa sovellusta, kun funktiota suoritetaan jokaisen tiilen latauksen yhteydessä. Neliöjuuri a on n. 1,41 kertaa a:ta. Sen voi pyöristää ylöspäin arvoon 1,5, koska tuloksia tullaan rajoittamaan myöhemmin välttääkseen duplikaattia. Hakulinkkiin tarvitaan ympyrän säde, joka saadaan suoraan puolikkaan neliön sivun pituuden neliöjuuresta. Tiilen puolikkaan sivun pituus saadaan maxY- ja $y0$-arvon erotuksesta (ks. Kuva 13).

$$r = (maxY - y0) * 1.5$$

Yleisesti geometria laskennassa ei ole merkitystä, minkä neliön sivuista valitaan laskemiseen, koska neliön sivut ovat kaikki samanpituisia. Maapallon piirien pituudet ilmoitetaan asteina (asteikolla 0º – 180º), sen pituuspiirien pituus (y-akseli) ovat kaikkialla samanpituisia, mutta leveyspiirien (x-akseli) pituus vaihtelee sijainnin mukaan. Tästä syystä on hyvä valita tiilen maxY-piste ja sen keskipiste niiden välisen matkan laskemiseen. Kaavasta saatu säde r on kuitenkin pituuspiirin asteen muodossa, jonka täytyy muuttaa kilometreiksi kertomalla 111,60 (1º ≈ 111,60 km).

```
double radius = Math.Abs((env.MaxY - centroidY) * 1.5 * 111.60);
```

Koodiesimerkki 3. Haettavan alueen säde



Kuva 13. Haettava alue säteellä r

Kyselyn varten järjestelmän nykyhetken UTC-aikaleimasta on vähennettävä 24 tuntia, koska halutaan rajoittaa tulokset vain niihin aluksiin, jotka ovat ilmoittaneet AIS-tietokantaan sijaintinsa viimeisen 24 tunnin aikana. Ohjelmoinnin näkökulmassa aikaleimaa vähennetään lisäämällä negatiivisia tunteja nykyiseen aikaleimaan, jonka jälkeen muunnetaan sopivaksi merkkijonoksi.

```
//Asetetaan aikaleimaa
string dateTime = (DateTime.UtcNow.AddHours(-24))
                .ToString("yyyy-MM-ddTHH:mm:ss.fffZ");
```

Koodiesimerkki 4. Aikaleima

Kun kaikille yllä mainituille muuttujille on määritetty arvot, voidaan muodostaa dynaamisen linkinsijoittamalla muuttujat merkkijonon oikealle paikalle. Osoite on oltava oikeassa muodossa, jotta rajapinta osaa hakea ja palauttaa oikeaa dataa. URL -osoitteen oikea muoto täytyy hakea itse rajapinnan dokumentaatiosta.

```
string url = "https://meri.digitraffic.fi/api/v1/locations/latitude/"
               + centroidY + "/longitude/" + centroidX + "/radius/" + radius
               + "/from/" + dateTime;
```

Koodiesimerkki 5. Linkin muodostaminen

5.2 Sijaintidatan pyyntö ja vastaanotto

Dynaamisen linkin avulla voidaan lähettää rajapinnalle GET- eli hakupyynnön, jonka jälkeen rajapinta palauttaa alusten raa'at sijaintidatat binääriketjuna (data stream). Raakatietoa ei voida käyttää suoraan sovellukseen, sitä täytyy ensin muuntaa sopivaksi muodoksi. Käydään läpi, miten datan pyyntö- ja vastaanottotapahtuma palvelimelta hoidetaan C#-koodissa.

Sovelluksen ja rajapinnan välille tulee luoda yhteyden WebRequest- ja WebResponse-luokan avulla. Yhteys luodaan HttpWebRequest-luokan avulla, jossa välitetään Create-funktiolle URL-osoitteen, josta dataa halutaan hakea (Tässä tapauksessa URL-osoitteena toimii luotu dynaaminen linkki). Rajapinnan vastausta pyydetään GetResponse-funktiolla, joka muodostaa protokollakohtaisen pyynnön WebRequest-ominaisuuksista, tekee TCP- (Transmission Control Protocol) tai UDP-liitännän (User Datagram Protocol) rajapintaan ja lähettää pyynnön. Kun rajapinnalta on saatu vastausta, kutsutaan vastauksen kautta GetResponseStream-funktiota, joka palauttaa rajapinnan resurssia binääriketjun muodossa.

Binääriketjun muotoista raakadataa täytyy käsitellä ennen kuin SpatialWeb ymmärtää, mitä data sisältää ja osaa siten sijoittaa alukset oikealle paikoilleen. Binääriketjua välitetään $ParseGeoJSONFeatureCollection$-funktioon käsiteltäväksi, missä dataa muunnetaan sopivaksi SpatialWebin käyttöön. Binääriketjun "responseStream" lisäksi, funktioon välitetään myös karttatiilen $FeatureDelegate$-niminen ominaisuuskäsittelijä, "$ref\ abort$"-niminen viittaus tiilen keskeytyksen totuusarvoon ja "$geom$"-niminen geometria-olio.

```
HttpWebRequest request =
      (HttpWebRequest)WebRequest.Create(url);

using (HttpWebResponse response = (
      HttpWebResponse)request.GetResponse())

using (Stream responseStream = response.GetResponseStream()){

          featuresReturned = ParseGeoJSONFeatureCollection(
               responseStream , FeatureDelegate, ref abort, geom);
            }
```

Koodiesimerkki 6. Datan pyynnön luonti ja vastauksen vastaanottaminen

5.3 Sijaintidatan käsittely

Määritetään ensin $Feature$-tyyppisen listan, johon tallennetaan kaikki rajapinnalta saadut features-ominaisuudet lisättäväksi tiilen tietoihin. Binääriketjun muotoiset datat on vaikea hallinnoida ja muokata, joten muunnetaan sitä JSON-muotoon. $JSON$ on tapa tallentaa tietoja järjestettyyn, helppokäyttöiseen tapaan. Toisin sanoen se antaa meille ihmisen luettavissa olevan tietomalliston, jota voimme käyttää todella loogisella tavalla (ks. liite 2).

Käyttämällä $StreamReader$- ja $JsonTextReader$-lukijaa voidaan helposti muuntaa binääriketjua JSON-muotoon. JSON-muotoisesta tekstistä valitaan vain geometry-kentän lisättäväksi features-listaan, koska tässä vaiheessa tarvitaan vain alusten koordinaatit sijoittaakseen niitä kartalle.

```
using (var sr = new StreamReader(stream))
      using (var jr = new JsonTextReader(sr)) {
          features = GeoJSONReader.ReadFeatureCollection(jr,
                        "geometry");
      }
```

Koodiesimerkki 7. JSON-muotoon muunnos

Haettu alue on suurempi kuin karttatiili, joten karttatiilen ulkopuolella olevat alueet ovat ylimääräisiä. Ylimääräisen alueen alusten sijaintitiedot ovat samat kuin viereisten tiilien ja aiheuttavat duplikaateitta. Tulokset pitäisivät siis rajoittaa karttatiilen alueen sisälle. Edellä mainittu features-listaan on lisätty kaikki haun alusten koordinaatit, sitä voidaan käyttää hyväkseen tulosten rajoituksessa. Käydään foreach-silmukassa listan jokaista koordinaattia läpi, ja lisätään SpatialWebiin vain ne koordinaatit, jotka sijoittuvat karttatiilen sisäpuolelle.

```
foreach (Feature feature in features) {
      if (feature.Geometry.Intersects(geom)) {
                  featureDelegate(feature, ref abort);
                  featureCount++;
              }
```

Koodiesimerkki 8. Alusten lisääminen SpatialWebin tasoon

5.4 Tason päivittäminen

Vaikka REST-rajapinnalta haetut datat ovat ajan tasalla, hakua pitää silti suorittaa tietyin väliajoin, jotta sovelluksen datat pysyvät myös ajan tasalla. Datanhakua suoritetaan aina tason päivityksen yhteydessä, joten jollain tavalla pitäisi pakottaa tasoa päivittämään itseään aina, kun uusia dataa on saatavilla. SpatialWebissa on olemassa tällainen automaattinen päivitys, mutta kyseinen päivitys päivittää kartan kaikki tasot riippumatta niiden olotilasta, mikä voi johtaa suoristuskyvyn ongelmiin, jos asennettujen tasojen määrä on suurin. Ratkaisuksi voidaan päivittää tietty ajoin vain ne tasot, jotka käyttävät AIS-sovelluksen tietolähdettä.

SpatialWebille voi määrittää sellaisen JavaScript-tiedoston, jolla suoritetaan omia mukautettuja toimintoja. Hyödyntämällä kyseistä tiedostoa voidaan pakottaa sovelluksen kutsua hakutapahtuma tietty väliajoin. $ApplicationBase$-pohjaisessa tiedostossa voi päällekirjoittaa olemassa olevaa funktiota $JSFiles$, jossa kerrotaan suoritettavan tiedoston sijainti ja $InitJSFunction$-funktiossa kerrotaan, minkä funktion JavaScript-tiedostossa suoritetaan.

```
public override string[] JSFiles{
     get { return new[ { "Applications/AISSitePlugin/Scripts/startup.js" }}
}
public override string InitJSFunction {
        get { return "AISstartup"; }
}
```

Koodiesimerkki 9. JavaScript-tiedoston määrittäminen

JavaScript-tiedossa voidaan pyytää manuaalisesti SpatialWebin päivittämään tietyn nimistä tasoa väliajoin. Tietoja ei kuitenkaan kannata kovakoodata, koska pääkäyttäjä joutuisi käyttää tason luontivaiheessa täsmälleen samaa nimeä kuin koodissa. Muuten ohjelma ei tunnistaisi päivitettävää tasoa. Parempi ratkaisu on antaa pääkäyttäjän määrittää itse asetukset, joita voidaan hakea tietolähteen kautta ja niiden avulla päivittää oikeaa tasoa. Näin pääkäyttäjä pääsee itse vaikuttamaan tason asetuksiin ilman minkälaista ennakkotietoja (ks. Kuva 14).

Kuva 14. Tason asetukset admin-käyttöliittymässä

Tietolähteen kautta haettujen asetuksien avulla päivitetään tasoa seuraavasti. Ensin luodaan $AISstartup$-funktion, kuten $InitJSFunction$-funktiossa on määritetty, jonka jälkeen määritetään funktiossa muuttujat, jotka ottavat vastaan asetusten arvoja (ks. Koodiesimerkki 10).

```
function AISstartup() {

    var app = SW.getAppById("AISSitePlugin");

    $.each(app.settings, function (idx, value) {

        var layer = SW.getLayerById(value.layerName);

        var frequency = value.frequency * 1000;

        var needForceUpdate = value.needForceUpdate;

       //setInterval()

    });

}
```

Koodiesimerkki 10. AISstartup-funktio

SetInterval-metodilla luodaan ikuisen silmukan, jossa metodi kutsuu toiminnon tai lausekkeen määrätyin väliajoin (frequency). Metodi jatkaa kutsumistoimintoa niin kauan kuin clearInterval-metodia kutsutaan tai sovellus suljetaan. Kuitenkaan ei haluta päivittää tasoa silloin, kun se ei ole site-käyttöliittymässä laitettu päälle. Suuren datamäärän vuoksi taustalla turhaan päivittämistä voi hidastaa site-käyttöliittymää. Voidaan aloittaa silmukkaa jo site-käyttöliittymän käynnistäessä, mutta $forceUpdate$-metodilla päivitetään AIS-tasoa vain silloin, kun se on päällä (ks. Koodiesimerkki 11).

```
setInterval(function () {
                if (layer.enabled) {
                    layer.forceUpdate(true);
                }}, frequency);
```

Koodiesimerkki 11. setInterval-metodi

5.5 Alusten metadata

AIS-alusten metadataa täytyy hakea rajapinnalta samalla tavalla kuin sijaintidata, eli luodaan dynaaminen linkki, pyytää datat rajapinnalta ja muokkaa saadut binääriketjut $JSON$-muotoon helpottaakseen datan käsittelyä. Alusten metadatat päivittyvät rajapinnalla n. 6 minuutin välein, mutta toisiin kuin sijaintidatat, alusten metadata (nimi, mmsi, jne.) ei todennäköisesti muutu usein, joten niitä ei tarvitse päivittää heti, kun uusimmat datat ovat saatavilla.

Metadata on hyvä tallentaa muistiin, ettei tarvitse jokaisen pikatietoikkunan kohdalla suorittaa uuden haun vain yhden aluksen takia. Helpointa tietotallennus tapaa C#-ohjelmoinnissa on kokoelma $Dictionary$, jossa tiedot tallennetaan avain-arvo parina (ks. Koodiesimerkki 12). Alusten MMSI (Maritime Mobile Service Identity) on laivankohtainen radio ainutlaatuinen tunnistenumero, jota voidaan käyttää kokoelman avaimena. Avaimen arvona säilytetään kyseisen aluksen kaikki metadatat, jonka vuoksi täytyy luoda metadatalle oma AISMetadata-mallin.

```
private Dictionary<string, Models.AISMetadata> metadata
            = new Dictionary<string, Models.AISMetadata>();
```

Koodiesimerkki 12. Dictionary-kokoelman luonti

Sen sijaan, että sijoitetaan datat SpatialWebin features-listaan kuten sijaintidata, metadataa voi muuntaa suoraan JSON-muotoisesta $JsonConvert$-luokan $DeserializeObject$-metodilla objektiksi, jonka jälkeen niitä voidaan lisätä $Dictionary$-kokoelmaan (ks. Koodiesimerkki 13).

```
public void AddData() {
       using (StreamReader sr = new StreamReader(responseStream)) {

                     string rawJson = sr.ReadToEnd();

                     var obj = JsonConvert.DeserializeObject

                         <List<Models.AISMetadata>>(rawJson);

       //Lisätään objektin sisällöt Dictionary-kokoelmaan

       }

}
```

Koodiesimerkki 13. Metadatan käsittely

Kun haetut metadatat on käsitelty, jokaisen aluksen metadata on tallennettu objektiin yhtenä esineenä. Objektin esineitä voi käydä läpi kuten samalla tavalla kuin listan sisällöt. Jokaisen esineen (ts. aluksen metadatan) kohdalla etsitään aluksen MMSI-arvo ja tallennetaan sen kokoelman avaimeksi, jonka arvona on aluksen metadatat uutena AIS-Metadata-mallina.

```
foreach (var item in obj) {
              lock (padLock) {
                     metadata.Add(item.MMSI.ToString(),
                                   new Models.AISMetadata {
                                          ...
                                  });
              }
}
```

Koodiesimerkki 14. Objektin esineiden lisäys kokoelmaan

Koska kyseessä on Dictionary-kokoelma, joka toimii asynkronisesti eli kokoelman toiminnot eivät ole ajallisesti toisistaan riippuvaisia, voi joskus syntyä päällekkäisyyksiä tai puutteellisia tietoja. Tällaisissa tapauksissa syntyy virheitä, koska kokoelmassa ei saa olla samanlaisia tai tyhjiä avaimia. Sen takia joudutaan käyttämään lukkoa, joka lukitaan aina ennen kuin suoritetaan jotakin kokoelman toimintoa ja vapautetaan lukon seuraa-

valle toiminnolle, kun edellinen on valmis (ks. Koodiesimerkki 15). Tällä tavalla varmistetaan, että objektin esineitä käydään läpi järjestyksessä ja seuraavaan mennään vasta, kun edellinen on suorittanut tehtäviensä.

```
private object padLock = new object();

      lock (padLock) {

            // Do something
}
```

Koodiesimerkki 15. Objektilukon luonti ja käyttö

Vaikka AIS-alusten metadata ei muutu yhtä usein kuin sijaintidata, niiden täytyy silti joskus päivittää, jotta tiedot ovat ajan tasalla. Tätä varten luodaan sellaisen funktion, joka tarkistaa jokaisen metadatan pyynnön yhteydessä, ovatko tiedot vanhat ja päivittää niitä tarvittaessa. Metadataa pyydetään aina, kun Angular-projekti käynnistyy ts. jokaisen aluksen pikatietoikkunan avautuessa.

Pyyntö tapahtuu $GetMetadata$-metodin kautta, jonne välitetään aluksen MMSI -numero. Funktion alussa tarkistetaan, onko metadata-kokoelma (Dictionary) tyhjä tai onko edellisestä päivityksestä kulunut 6 tuntia. Jos jompikumpi ehdosta on totta, kirjataan ensin päivityksen aikaleiman ylös, jonka jälkeen alustetaan kokoelma tyhjentämällä sen ja lisätään metadata kokoelmaan $AddData$-metodilla. Sen jälkeen, kun kokoelmaa on päivitetty, palautetaan kokoelmasta sellaisen AISMetadata-mallisen objektin, jonka avain-arvo on sama kuin funktioon välitetty MMSI-numero (ks. Koodiesimerkki 16).

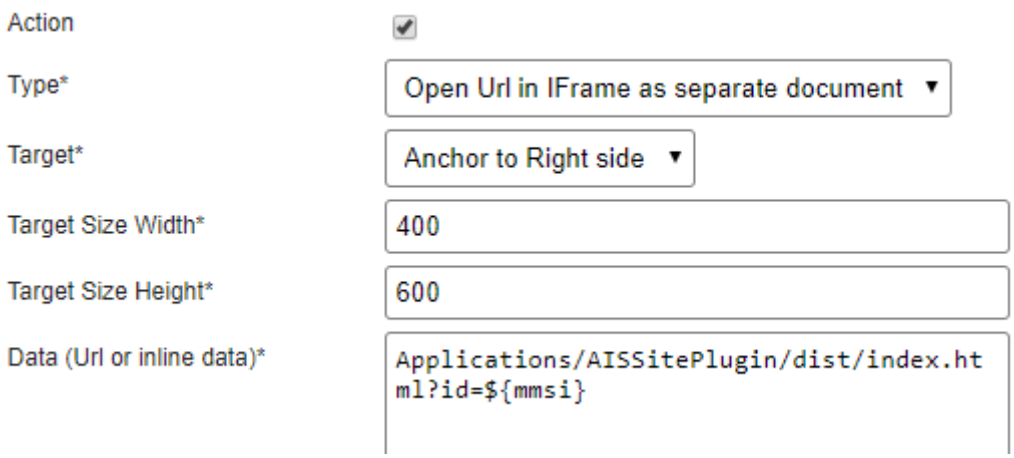
```
public Models.AISMetadata GetMetadata(string mmsi){

    lock (padLock) {

      if (metadata.Count <= 0 || lastUpdate.AddHours(6) <= DateTime.UtcNow){

                  lastUpdate = DateTime.UtcNow;

                   ClearData();

                   AddData();
            }
            return metadata[mmsi];
       }
}
```

Koodiesimerkki 16. Alusten metadatan haku

Toinen syy Dictionary-kokoelman valintaan on se, että kokoelmasta objektin etsiminen avaimen perusteella on hyvin helppo. Kokoelman ei tarvitse käydä läpi tietyn avaimen arvon etsimiseen.

5.6 Metadatan esittäminen Angularilla

Alusten pikatietoikkuna aukaistaan valitsemalla kartalla oleva alus ikoniin. Tätä valintaprosessia voi käyttää hyväkseen aluksen MMSI-numeron välittämiseen metadatan hakuun backendissa. SpatialWebin admin-liittymässä voidaan määrittää, mitä halutaan suorittaa, kun tason ikonia valitaan. Pikatietoikkuna on määritetty (ks. Kuva 15)iframe-kehykseen, jossa näytetään Angularin pääsivua. Linkin loppuosaan lisätään kysymysmerkin, jonka jälkeen voidaan sijoittaa haluamamme arvot parametreina käytettäväksi Angularissa. Tässä tapauksessa id-parametrina toimii aluksen MMSI-numeron, jonka saadaan aluksen sijaintidatan kautta.



Kuva 15. SpatialWeb admin-liittymän Action-kentät

Kun backend -haku ja päivitys ovat kunnossa, halutaan näyttää haetut metadata pikatieto-ikkunassa. Backend -koodit ei kuitenkaan voi käyttää suoraan frontendissa. Datan siirron varten täytyy luoda sisäisen API -kontrolleriin, jonka kautta tiedot kulkevat kahden puolen välillä. Tässä tapauksessa on kyseessä datan haku backendista, joten käytetään [$HttpGet$], jolle pitää määrittää reitin ([$Route$]). Reitin avulla voidaan pyytää frontendissa Http-kontekstiin tallennetut vastaukset.

API-funktion sisältö (ks. Koodiesimerkki 17) on melko simppeli. Funktiossa kutsutaan metodia $GetMetadata()$, jonne välitetään API reitissä oleva {mmsi}-arvo. Metodi $GetMetadata()$ palauttaa haettavan aluksen metadata, jota tallennetaan ensin muuttujaan $meta$. Sen jälkeen alustetaan Http-kontekstin vastaus JSON-muotoiseksi ja tallennetaan muuttujan $meta$ sisällöt vastaukseen frontendin käytettäväksi.

```
[Route("vessels/{mmsi}")]
[HttpGet]
        public ActionResult Vessels(string mmsi){
            meta = handler.GetMetadata(mmsi, true);
            HttpContext.Response.ContentType = "application/json";
            HttpContext.Response.Write(JsonConvert.SerializeObject(meta));
            HttpContext.Response.Flush();
            return null;
}
```

Koodiesimerkki 17. Sisäinen API

Angular -reititin $router$ voi tulkita URL-osoitetta käskynä ohjata haluttuun näkymään. Se voi siirtää myös valinnaisia parametreja komponenttiin, joka auttaa sitä päättämään, mitä erityistä sisältöä tulee esille. Tätä ominaisuutta hyödyntäen voidaan ottaa aluksen MMSI-arvon talteen osoitteen parametrin kautta ja luoda uuden URL-osoitteen Angularin kontrollerin varten.

```
public getParsedUrl(): string {
        this.router.events.subscribe((event: Event) => {
            if (event instanceof RoutesRecognized) {
                if (event.state.root.firstChild.queryParams['id']) {
                    this.url = "../vessels/" +
                        (event.state.root.firstChild.queryParams['id']
                            .toString()); }}});
       return this.url;
}
```

Koodiesimerkki 18. MMSI-numeron vastaanotto parametrina

Muodostetulla URL-osoitteella voidaan hakea kontrollerin sisäisen API kautta metadata kutsumalla Angularin http-palvelun $get()$-metodia. Samalla pysäytetään sovellusta $catch()$-metodilla virhetilanteiden syntyessä. Tällaiset haut yleensä suoritetaan Angularin palvelussa, joka on helposti injektoida moniin eri komponentteihin.

```
getVesselLocation(): Observable<VesselData> {
        return this.http.get(this.urlQueryService.getParsedUrl())
            .catch(errors);
    }
```

Koodiesimerkki 19. Metadatan pyyntö Angularissa

Sovelluksen komponentissa voidaan kutsua palvelun metodia ja tehdä tilaus, jossa tallennetaan metadatat komponentin muuttujan tulostettavaksi pikatieto-ikkunaan. Vaikka backendissa on tehty haetuille metadatalle oma malli, sen muoto pysyy vain backendin käsittelyssä. Tässä ollaan siirretty dataa backendista frontendiin JSON-muodossa, jolloin täytyy muodostaa frontendin oma malli, jonka mukaan tallennetaan metadatat käsiteltäväksi frontendissa.

```
public vesselData: VesselData;
ngOnInit() {
        this.vesselLocationService.getVesselLocation()
            .subscribe(data => this.vesselData = data);
}
```

Koodiesimerkki 20. Datan tilaaminen HttpGET API:lta

Metadata on nyt käytettävissä Angularissa, joten voidaan tulostaa datat taulukkoon näkyviin komponentin html-sivulla (ks. liite 3). Angulariin voi myös lisätä bootstrap, jolla parannetaan sovelluksen ulkoasua.

6 Yhteenveto

Sovellus on toteutettu SpatialWebin karttatasona. Alusten sijainnit päivittyvät automaattisesti valittu väliajoin. Päivitysfrekvenssi voi määrittää sovelluksen Admin -asetuksissa. Rajapinnasta saatujen sijaintitietojen (geometry) avulla voidaan näyttää alukset SW-kartan tasolla. Ohjelma hakee rajapinnasta tietoja frekvenssin verran väliajoin (esim. 10 sekunnin välein) ja samalla päivittää myös SW-tasoa, jolloin alusten sijainnit myös päivittyvät kartalla.

Kartalla olevat aluksille voidaan määrittää ikonit niiden nykyisen statuksen mukaan. Viemällä hiiren kartalla olevan ikonin päälle kohteesta näytetään tooltip-tieto (info-ikkuna), jossa on esimeriksi aluksen nimi, nopeus tai muita tietoja. Painamalla ikonia kartan vasempaan reunaan ilmestyy pikatietoikkuna, jossa on kohteen tarkempia tietoja.

Sovellukseen ei kehitetty testitapauksia, koska niitä ei ole vaadittu määritelmässä.

Jatkosuunnitelmana on kehittää alusten sijaintihistorian tallennus tietokantaan, josta voidaan hakea ja piirtää kartalle reitti, jota pitkin alus on kulkenut edellisen 24 tunnin aikana. Tätä suunnitelmaa toteutetaan vain, jos asiakas haluaa historia-toiminnon.

Jatkosuunnitelmaa toteuttaessa myös metadata voidaan tallentaa tietokantaan, toisiin kuin sovelluksen muistiin tällä hetkellä. Rajapinnalta tulevat alusten sijaintitiedot tallennetaan tietokantaan ja päivitetään niitä tarvittaessa. Koska kartalla on suuria määriä aluksia, ei voida näyttää kaikkien sijaintihistorioita samaan aikaan. Tästä syystä sijaintihistoria todennäköisesti tulee näkyviin pikatieto-ikkunan napin kautta, jolloin kartalta piilotetaan muut alukset ja piirretään valitun aluksen sijaintihistoria reitin kartalle.

Tietokantaan tallennetaan n. tunnin välein kaikkien alusten sijainnit ja samalla poistetaan yli 24 tunnin vanhat sijaintitiedot. Tällä tavalla vältetään tietokannan koon kasvamista liian suuriksi. Tallennetuista sijainnista muodostetaan lista, jossa on sijainnin koordinaatti ja sen aikaleima. Koordinaattien mukaan sijoitetaan kartalle pisteet ja niitä yhdistetään aikaleiman järjestyksessä.

Lähteet

1 Satellite AIS - Introduction to Satellite AIS. 2015. Verkkodokumentti <https://cdn2.hubspot.net/hubfs/183611/Landing_Page_Documents/Satellite_AIS_White_Paper_Final-1.pdf>.

2 Marine Traffic - What is the Automatic Identification System (AIS)? 2012. Verkkodokumentti <https://help.marinetraffic.com/hc/en-us/articles/204581828-Automatic-Identification-System-AIS>.

3 All About AIS - How AIS works. 2012. Verkkodokumentti <http://www.allaboutais.com/index.php/en/aisbasics1/how-ais-works>.

4 Navigation Center - Types of AIS. 2017. Verkkodokumentti <https://www.navcen.uscg.gov/?pageName=typesAIS>.

5 Scott Chason & Ben Straub. 2009. Pro Git, 2nd Edition. Apress

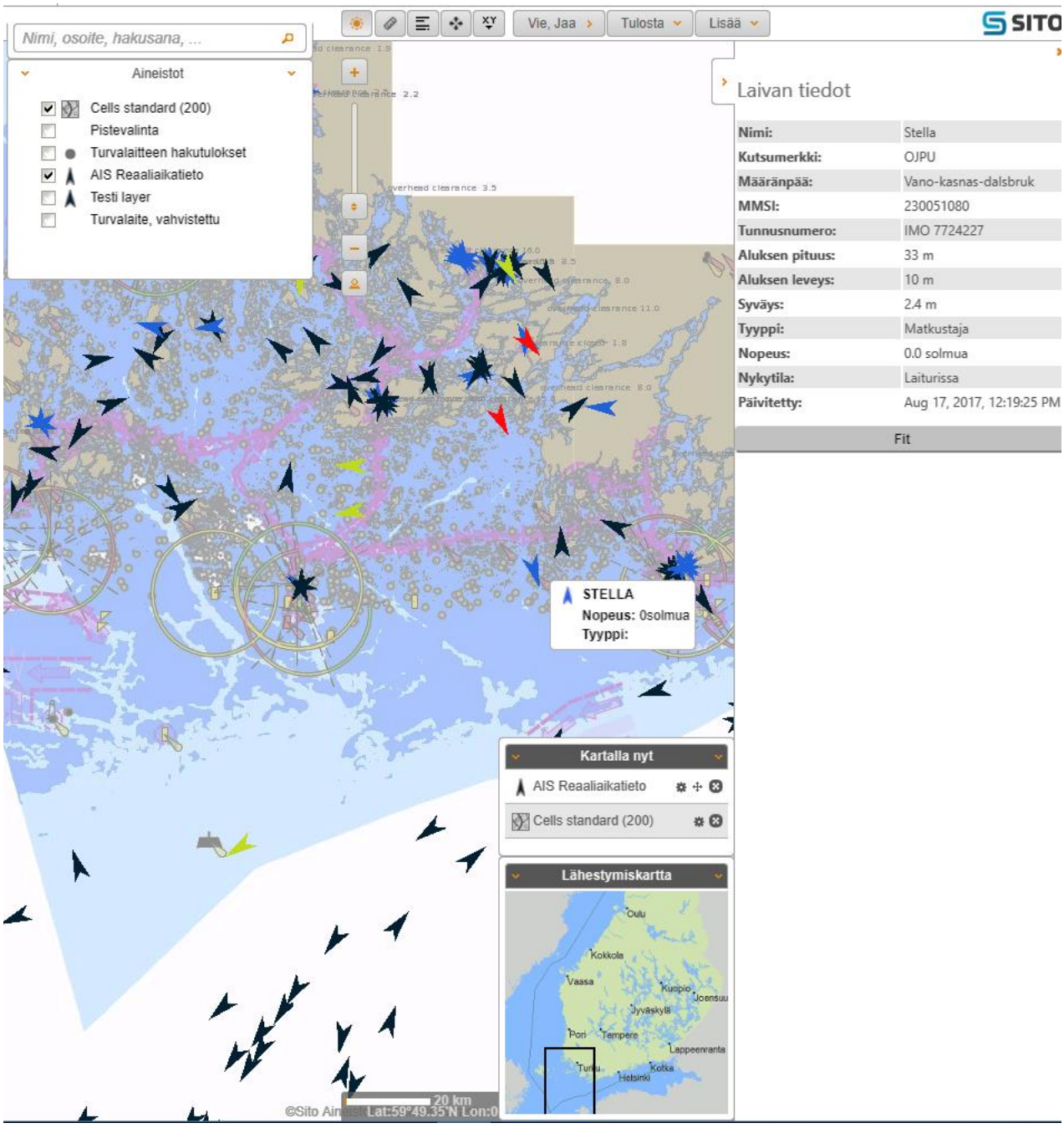
6 Fain, Y. & Moiseev A. 2017. Angular 2 Development with TypeScript. Shelter Island: Manning Publications Co.

7 Angular 2 Documents – Fundamentals. 2017. Verkkodokumentti https://angular.io/docs.

8 Webpack documentation - Core Concepts. Verkkodokumentti https://webpack.js.org/concepts/.

9 Liikennevirasto – Digitraffic. 2017. Verkkodokumentti. https://www.liikennevirasto.fi/avoindata/digitraffic#.WfX4T2i0OUk.

Lopputulos



Kuvio 1. AIS-alukset esitetään kartan tasolla

Rajapinnan data JSON-muodossa


```
{
  "features": [
    {
      "geometry": {
        "coordinates": [
          {}
        ],
        "type": "string"
      },
      "mmsi": 0,
      "properties": {
        "cog": 0,
        "heading": 0,
        "navStat": 0,
        "posAcc": false,
        "raim": false,
        "rot": 0,
        "sog": 0,
        "timestamp": 0,
        "timestampExternal": 0
      },
      "type": "Feature"
    }
  ],
  "type": "FeatureCollection"
}
```

Rajapinnan Response-vastauksen malli

```
VesselLocationFeatureCollection {
       features (Array[VesselLocationFeature]),

       type (string) = ['FeatureCollection']

}

VesselLocationFeature {

       geometry (Geometry, optional): GeoJSON Geometry object ,

       mmsi (integer): Maritime Mobile Service Identity ,

       properties (VesselLocationProperties): GeoJSON Properties object ,

       type (string) = ['Feature']

}

Geometry {

       coordinates (Array[Inline Model 1]): WGS84 coordinates in decimal
       degrees. [LONGITUDE, LATITUDE, ALTITUDE] ,

       type (string): Type of GeoJSON object

}

VesselLocationProperties {

       cog (number): Course over ground in 1/10 = (0-3599). 3600 = not
       available ,

       heading (integer, optional): Degrees (0-359) (511 indicates not
       available = default) ,

       navStat (integer): Navigational status ,

       posAcc (boolean): Position accuracy, 1 = high, 0 = low ,

       raim (boolean): Receiver autonomous integrity monitoring (RAIM) flag of
       electronic position fixing device ,

       rot (integer): Rate of turn ,

       sog (number): Speed over ground in 1/10 knot steps ,

       timestamp (integer): UTC second when the report was generated by the
       electronic position system ,

       timestampExternal (integer): Location record timestamp in milliseconds
       from Unix epoch.

}
```

vesselComponent.html

```
<div *ngIf="vesselData">
    <table class="striped-table">
        <caption>Laivan tiedot</caption>
        <tbody>
            <tr>
                <td>Nimi: </td>
                <td>{{this.capFirst(vesselData.name)}}</td>
            </tr>
            <tr>
                <td>MMSI: </td>
                <td>{{vesselData.mmsi}}</td>
            </tr>
            <tr>
                <td>Tunnusnumero: </td>
                <td *ngIf="vesselData.imo !== 0">IMO {{vesselData.imo}}</td>
                <td *ngIf="vesselData.imo === 0">Unavailable</td>
            </tr>
            <tr>
                <td>Syväys: </td>
                <td>{{vesselData.draught/10}} m</td>
            </tr>
            <tr>
                <td>Tyyppi: </td>
                <td>{{returnShipType(vesselData.shipType)}}</td>
            </tr>
         </tbody>
    </table>
</div>
```

C# malli

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace AISSystem.Core.Models
{
    public class AISMetadata
    {
        public long Timestamp { get; set; }
        public int Eta { get; set; }
        public int ShipType { get; set; }
        public int MMSI { get; set; }
        public string Destination { get; set; }
        public int Draught { get; set; }
        public int IMO { get; set; }
        public string Callsign { get; set; }
    }
}
```

Angular malli

```
export class VesselData {


    public callSign: string;

    public destination: string;

    public draught: number;

    public eta: number;

    public imo: number;

    public mmsi: number;

    public name: string;


    constructor(

        callSign: string,

        destination: string,

        draught: number,

        eta: number,

        imo: number,

        mmsi: number,

        name: string{


        this.callSign = callSign;

        this.destination = destination;

        this.draught = draught;

        this.eta = eta;

        this.imo = imo;

        this.mmsi = mmsi;

        this.name = name;

    }

}
```