Scientific Journals
of the Maritime University of Szczecin

Zeszyty Naukowe
Akademii Morskiej w Szczecinie

2016, 46 (118), 68–75
ISSN 1733-8670 (Printed)
ISSN 2392-0378 (Online)
DOI: 10.17402/120

Received: 31.08.2015
Accepted: 06.05.2016
Published: 27.06.2016

Internet application to support yacht navigation

Daniel Dakus, Artur Zacniewski✉

Polish Naval Academy, Faculty of Navigation and Naval Weapons
69 Śmidowicza St., 81-103 Gdynia, Poland
e-mails: dax0000@gmail.com, a.zacniewski@amw.gdynia.pl
✉ corresponding author

Key words: internet application, yacht navigation, sea monitoring, AIS, geolocation, ruby on rails

Abstract
The article presents the implementation of an Internet application that graphically depicts the current situation at sea, with the prospect of being used in maritime applications and the possibility to work on multiple hardware platforms. Modern techniques have been applied in order to allow permanent development of the application and ensure that it is maintenance-free and self-reliant, even in case of unexpected situations. This publication covers the procedure followed to build the application as well as its field and method of operation. Possible usages of the application and tests carried out on many platforms have been presented. Ideas for further development have also been introduced.

Introduction

Is there a web application that combines geolocation features, safety, and sea monitoring for assisted navigation?

Many web sites keep track of ships, monitor the weather, and share electronic maps. Unfortunately, most Internet services are unwilling to share data, and therefore only disclose a preview of the information or commercial material. Navigation requires precision and accuracy, thus demanding the use of more accurate, higher resolution, data. A review of the available Internet applications, which assist navigation and sea monitoring, has shown, among other things, a market gap in applications that combine multiple functions and layers of information, such as geolocation, weather conditions, ship tracking, and navigation warnings.

On the basis of the above considerations, the idea emerged of creating an application that gathers in one place data coming from a variety of sources and displays up-to-date information on a map. The application is intended to work on multitude of hardware platforms, including mobile platforms.

Materials and methods

Ruby on Rails

Rails is a web application development framework written in the Ruby language. It is designed to facilitate web application programming by making assumptions regarding the basic needs of a developer. It allows to write less code while accomplishing more than many other languages and frameworks (Ruby on Rails guide, 2015).

Ruby on Rails significantly accelerates the creation of Internet applications because it applies the scaffolding method. By merely providing short commands, it is possible to generate the whole skeleton of an application, creating the basis for further operation. *Ruby on Rails* includes everything needed to create database-backed web applications according to the Model-View-Controller (MVC) pattern (Ruby on Rails API, 2015).

Rails is equipped with multiple, well known, embedded libraries such as jQuery, which is a fast, small, and feature-rich JavaScript library. It makes HTML document traversal and renders manipulation,

event handling, animation, and Ajax much simpler. In addition, an easy-to-use API works across a multitude of browsers (jQuery, 2015).

The possibility of conducting unassisted updates of selected libraries, or installing new ones, is envisages. *Rails* also offers the opportunity to install many external GEMs, which significantly expand the framework capabilities and automate many creative processes. For example, Nokogiri GEM is an HTML, XML, SAX, and Reader parser. Nokogiri's many features include the ability to search documents via XPath or CSS3 selectors (Nokogiri, 2015).

Google Maps API

The Google Maps APIs provide several ways of embedding Google Maps into web pages, and allow for either simple use or extensive customization (Google developers, 2015).

Google Maps API uses several coordinate systems:

- Latitude and longitude values, which uniquely reference a point in the world (Google uses the World Geodetic System WGS84 standard).
- World coordinates, which uniquely reference a point on the map.
- Tile coordinates, which reference a specific tile on the map at the specific zoom level.

Whenever the Maps API needs to translate a location in the world to a location on a map (the screen), it needs to first translate latitude and longitude values into a "world" coordinate. This translation is accomplished using a map projection. Google Maps uses the Mercator projection for this purpose. World coordinates in Google Maps are measured from Mercator projection's origin (the northwest corner of the map at 180 degrees longitude and approximately 85 degrees latitude) and increase in the x direction towards the east (right) and in the y direction towards the south (down). Since the basic Mercator Google Maps tile is 256 x 256 pixels, the usable world coordinate space is {0–256}, {0–256} (Map types, 2015).

Tiles and pixels in Google Maps are numbered starting from the same origin. For Google's implementation of the Mercator projection, the origin tile is always at the northwest corner of the map, with x values increasing from west to east and y values increasing from north to south (Figure 1). Tiles are indexed using x, y coordinates from that origin. For example, at zoom level 2, when the Earth is divided up into 16 tiles, each tile can be referenced by a unique x, y pair (Map types, 2015).

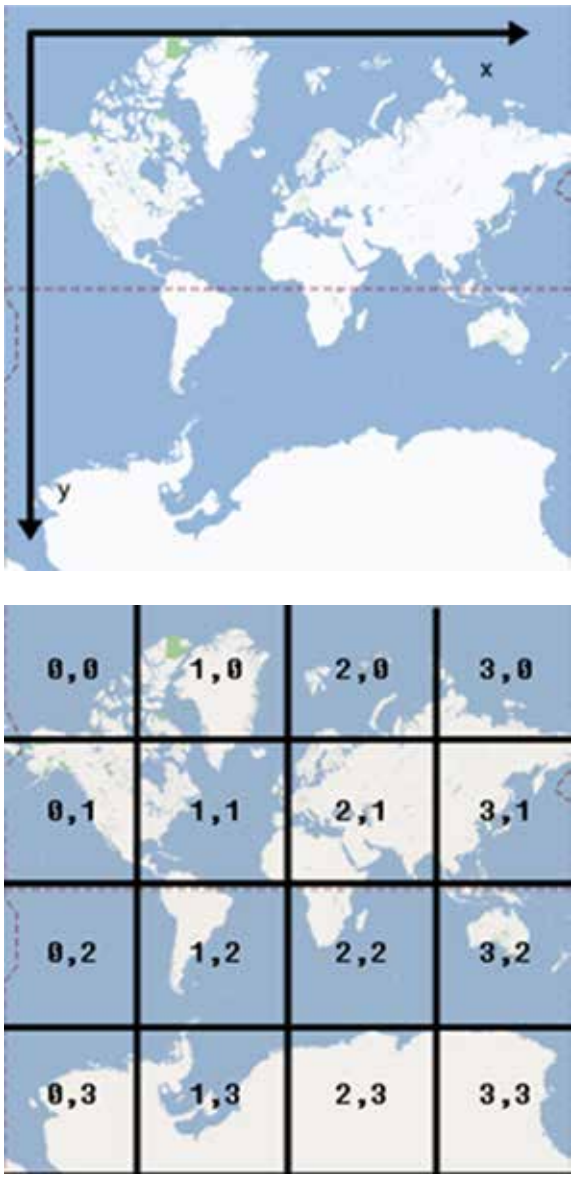

Figure 1. Google Maps Mercator projection measurement and tiles numbers (Map types, 2015)

HTML5 Geolocation API

The Geolocation API defines a high-level interface for location information, associated only with the device hosting the implementation, such as latitude and longitude. The API itself is agnostic of the underlying location information sources. Common sources of location information include Global Positioning System (GPS) and location inferred from network signals such as IP address, RFID, Wi-Fi and Bluetooth MAC addresses, and GSM/CDMA cell IDs, as well as user input (W3C, 2015). Example of using this API in JavaScript code is presented on Figure 2.

This specification is limited to providing a scripting API for retrieving geographic position information associated with a hosting device. The geographic

```
function scrollMap(position) {
  // Scrolls the map so that it is centered at (position.coords.latitude,
     position.coords.longitude).
}
// Request repeated updates.
var watchId = navigator.geolocation.watchPosition(scrollMap);

function buttonClickHandler() {
  // Cancel the updates when the user clicks a button.
  navigator.geolocation.clearWatch(watchId);
}
```

Figure 2. Example of requesting repeated position updates

position information is provided in terms of World Geodetic System coordinates WGS84 (W3C, 2015).

Data

Automatic identification systems

The Automatic Identification Systems (AIS) are designed to be capable of automatically providing information about the ship they are mounted on to other ships and to coastal authorities. AIS provide information – including the ship's identity, type, position, course, speed, navigational status, and other safety-related information – automatically to appropriately equipped shore stations, other ships and aircraft (ITU, 2015).

AIS Database Management System receives the NMEA O183 message, coded in accordance with ITU-R M.1371 sent via Saab R4S transponder, and then converts it to a readable XML file type format (Figure 3). Further, the XML file is made available to the Internet via a static HTTP server.

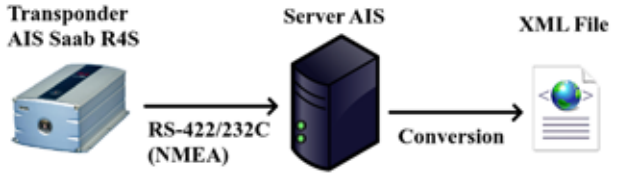

Figure 3. AIS data system architecture

Exemplary XML file with AIS data is presented on Figure 4.

```
<markers>
<marker lat="54.387687" lon="18.666317" name=
    "PIRAT USTKA " type="99" ais="261019170!0!
SPG2658!STATEK PASAZERSKI !29/07/2015
    11:40!4.6!216.0!8!20!14!3!3!0.4!1440767118"/>
<marker lat="54.530155" lon="18.560113" name=
    "LE QUY DON " type="36" ais="261031080!9728100!
SPG4220!GDYNIA !!5.4!352.8!0!33!34!4!6!4.3!1440767125"/>
<marker lat="54.530417" lon="18.535400" name=
    "BULKERS ALESSIA " type="70" ais="247280600!
9473573!IBYP !GDYNIA !25/08/2015
    00:30!0.0!196.0!5!150!30!10!20!9.3!1440767059"/>
</markers>
```

Figure 4. XML file fragment with AIS data

Meteorology / Weather

OpenWeatherMap API service provides many kinds of weather maps including Precipitations, Clouds, Pressure, Temperature, Wind, and many others. Moreover, the service connects them to mobile applications or web sites (OpenWeatherMap, 2015).

The weather data is collected from various weather stations, located all over the world, through the "protocol of weather station data transmission", which allows the transmission of one measurement. The data is transmitted by HTTP POST request (Table 1).

HTTP basic authentication is used (OpenWeatherMap, 2015).

Table 1. Parameters that can be transmitted in POST (if available)

Data – Name	Unit
wind_dir – wind direction	grad
wind_speed – wind speed	m/s
temp – temperature	grad C
humidity – relative humidity	%
pressure – atmosphere pressure	hPa
wind_gust – speed of wind gust	m/s
rain_24h – rain in recent 24 h	mm
rain_today – rain today	mm
lat – latitude	<lat>
long – longitude	<lon>
alt – altitude	m
name – station name	<text>

Next, the collected data can be obtained in JSON and XML formats. The search of weather data can be conducted in different ways:

- by city name (by geocoding system to find cities by name, country or zip-code);
- by geographic coordinates (The flexible algorithm, or estimation; of weather calculation allows to provide weather data not only for cities, but for any geographic coordinate and to receive weather forecast in any location on Earth);
- by city ID.

An examples of a request sent to http://api.openweathermap.org/data/2.5/weather?q=Gdynia,Poland &lang=en& units=metric is presented on Figure 5.

```
{"coord":{"lon":18.53,"lat":54.52},
"weather":[{"id":800,"main":"Clear","description":
    "clear sky","icon":"01d"}],
"base":"cmc stations",
"main":{"temp":24.44,"pressure":1016,"humidity":56,
    "temp_min":23, "temp_max":25.56},
"wind":{"speed":3.6,"deg":60},"clouds":{"all":0},"dt":1441020450,
"sys":{"type":1,"id":5349,"message":0.004,"country":
    "PL","sunrise":1440993078,"sunset":1441042787},
"id":3099424,"name":"Gdynia","cod":200}
```

Figure 5. Response API in JSON data format

Navigational Warnings

Navigation Warnings are regularly updated by the National Coordinator of Navigational Warnings operator on duty. They are available on the Hydrographic Office of the Polish Navy website in PDF format (Table 2). Up-to-date warnings are uploaded in separate panels, including: Local Warnings, Coastal Warnings, Subarea Warnings (BHMW, 2015).

Table 2. Data in PDF file (BHMW, 2015)

Identyfikator ostrzeżenia	GV_ON_0002/2015
Ważne od	2015-01-07 10:24:17
Ogólny obszar	ZATOKA GDAŃSKA
Dotyczy (treść)	W REJONIE PRZYBRZEŻNYM KĘPY OKSYWSKIEJ NA POZYCJACH: 1) 54-33'26,03361'N I 018-33'35,77877'E 2) 54-33'26,52619'N I 018-33'26,99430'E 3) 54-33'18,85022'N I 018-33'36,57699'E ZNAJDUJĄ SIĘ TRZY OBIEKTY POCHODZENIA MILITARNEGO. ZALECANA SZCZEGÓLNA OSTROŻNOŚĆ
Pozycja geograficzna	Φ=54°33'26"N Λ=018°33'35"E Φ=54°33'26"N Λ=018°33'27"E Φ=54°33'18"N Λ=018°33'36"E

In consideration of the fact that the file is in PDF format, as well as of the nature of the data that it includes, and the capabilities of API Google, it is currently not possible to transfer the data directly from the PDF file to the map.

It is possible to convert the PDF file into JSON or XML on the server with an appropriate script and to place it on the map, however considering the limited time span, the current data from the PDF file is placed on an external server, where it is properly prepared and saved in the .xml format beforehand. In the authors' opinion it would be much simpler to place the navigation warnings in the XML format directly on the HOPN (pol. BHMW) website.

XML data for one map marker is presented on Figure 6.

```
<markers>
<marker
lat="54.547231"
lon="18.559938"
id="GV_ON_0002/2015"
msg="W REJONIE PRAWOBRZEŻNYM KĘPY OKSYWSKIEJ
    NA POZYCJACH (THE COASTAL AREA KĘPA
    OKSYWSKA IN POSITION):<br>
1) 54-33'26,03361'N I 018-33'35,77877'E<br>
2) 54-33'26,52619'N I 018-33'26,99430'E<br>
3) 54-33'18,85022'N I 018-33'36,57699'E<br>
ZNAJDUJĄ SIĘ TRZY OBIEKTY POCHODZENIA
    MILITARNEGO (THERE ARE THREE OBJECTS OF
    MILITARY ORIGIN).<br>
ZALECANA SZCZEGÓLNA OSTROŻNOŚĆ (RECOMMENDED
    SPECIAL CARE)"/>
</markers>
```

Figure 6. Converted XML Data for one map marker

Results

With the use of API Google Maps; Ruby on Rails framework; and the collected data, namely AIS, Geolocation, Meteorology, and Navigation Warnings, the Internet application was created and made available at http://enavi.herokuapp.com/ address.

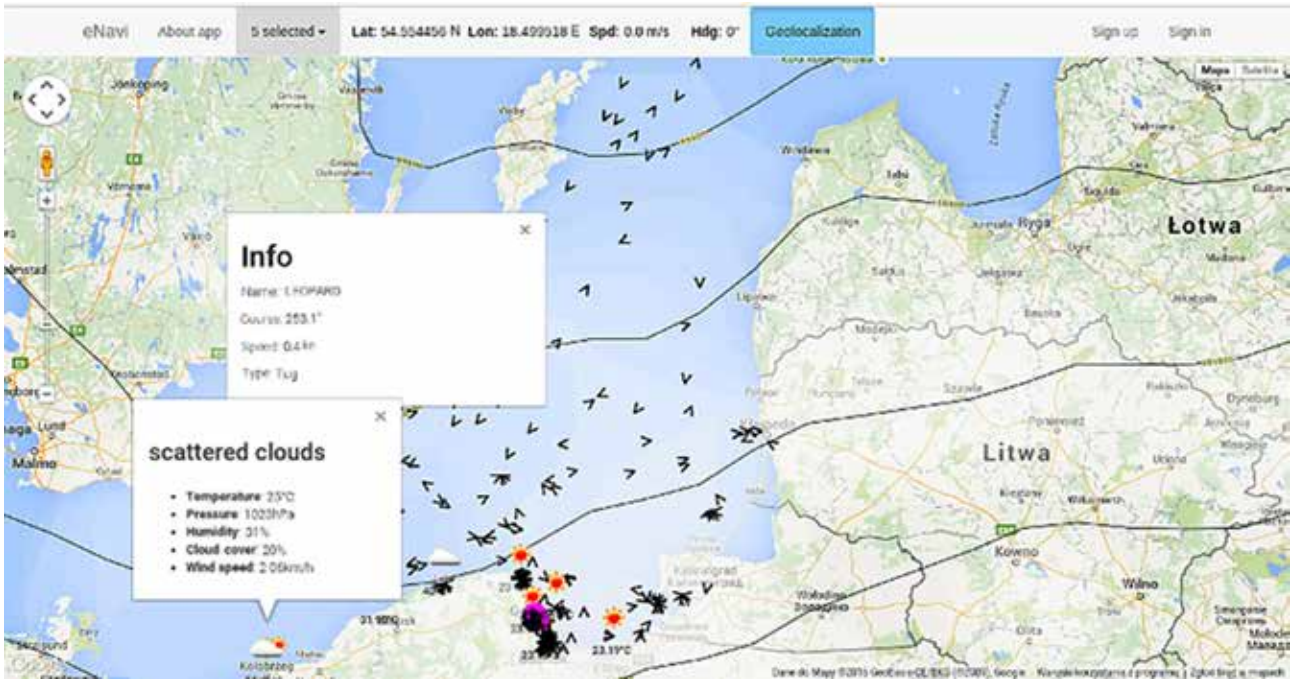

Figure 7. Application with selected layers

The application offers 5 layers, which enable the choice of required data. When selecting either a ship or the weather icon, detailed information is displayed.

The application with selected layers: ships, navigation warnings, current weather, pressure patterns, and clouds, as well as displayed info boxes concerning the selected ship is presented on Figure 7.

The innovativeness of the application lies in the fact that it is completely automatic and maintenance-free. This means that the operations of downloading data, converting it to the format required by API Google Maps, and representing it on the map, are conducted by the server. Data is updated without refreshing the page and live streamed. This was made possible by the use of AJAX, jQuery as well as Nokogiri methods. Below, the most essential code blocks and their functions are depicted and explained.

Ships (AIS data)

Code for analysis, parsing and rendering XML file is presented on Figure 8.

```
class ShipsController < ApplicationController
 def index
  begin
   doc = Nokogiri::XML(open('http://153.19.108.122/linia/GeoAIS.xml'))
  rescue
  @ships = Rails.cache.read('ships')
  else
   @ships = []
   markers = doc.xpath('//marker')
   markers.each do |marker|
    decoded_ship = decode_ship(marker)
    @ships.push(decoded_ship)
   end
   Rails.cache.write('ships', @ships)
  end
 render json: @ships
end
private
def decode_ship(marker)
 ais = marker.attributes['ais'].value.split('!')
 @hash = {
  lat: marker.attributes['lat'].value,
  lon: marker.attributes['lon'].value,
  name: marker.attributes['name'].value.strip,
  type: marker.attributes['type'].value,
```

Figure 8. A fragment of Ruby code: analysis, parsing and rendering of XML file

The process begins by the opening of GeoAIS.xml file, saved on the AMW server with the use of Nokogiri. Next, the *@ships* array is declared, to which the decoded information will be added in the definition of the *decode_ship(marker)* method, which was previously located with the use of XPath in the opened XML file. The process is finalised by rendering to the JSON form, which can be easily read by AJAX (Figure 9).


```
[{"lat":"54.448583","lon":"18.577737", "name":"PIRAT
USTKA","type":"99","imo":"0","call":"SPG2658","dest":
    "STATEK PASAZERSKI ","eta":"29/07/2015
11:40","speed":"0.0","course":"249.2","status":"8","length":
    "20","width":"14","draft":"3","mtime":"3","type
_text":"Other","info_window":"Info statku\u003e/Nazwa (Name):
    PIRAT USTKA\u003c/Kurs (Course):
249.2°\u003e/Prędkość (Speed): 0.0 kn\u003e/Typ (Type):
    Other\u003c/Cel (Destination): STATEK PASAZERSKI"}]
```


Figure 9. A fragment of ships.json file

Xpath is a pseudo-language that describes how to locate specific elements and attributes in an XML document, treating that document as a logically ordered tree (XPath, 2015).

The whole process is conducted in real time. Additionally, the XML file reading has been secured against exceptions, such as a missing file on the server or a writing error. Behind this are *Rails.cache* methods, as well as rescue exception handling, which continue coding with the data saved in cache after encountering an exception.

Displaying of the ships' markers is conducted with the use of jQuery, AJAX, and Java Script library of Google Maps API (Figure 10).

```
$(document).ready(function() {  var lat = 54.593714;  var lon = 18.892679;
 fitWindow();
 initialize(lat, lon);

 getShips();
 setInterval(function(){
  getShips();
}, 20000);
});

function getShips() {
 $.ajax({
  url: "/ships.json",
  async: false,
  data: {layers: JSON.stringify(selectedLayers)},
  method: 'GET'
 }).done(function(ships){
  var ships = ships;
  changeShips(ships);
 });
}
```

Figure 10. A fragment of JavaScript code that initiates the map and reading the parameters

The initializer starts the map with a predefined position, as required by the application. Next, the *getShips* function is applied, which uses jQuery to transfer an object to the AJAX method that contains the following parameters: path to *ships.json*, *async: false*, which means that AJAX will conduct the commands in a predetermined order to prevent readings of the map before ships markers have been read. Data is saved as *selectedLayers* in JSON notation for further use. The data fed is then placed on the map using the *changeShips* and *getShips* functions.

The data is continuously updated with the *setInterval* method, which reads and places the positions

on the map as well as ship parameters in the info window every 20 seconds.

Weather Data

The "Navigation warnings" layer works similarly to the code provided above, while weather, pressure patterns, and clouds are processed in a completely different manner, also due to the fact that data is presented in graphical form (Figure 11).

```
function showWeather() {
  var locations = ['Hel', 'Władysławowo', 'Gdynia', 'Leba',
  'Ustka', 'Kolobrzeg', 'Krynica Morska']
  for(var i=0; i<locations.length; i++)
  {
   loadWeather(locations[i]+',Poland');
  }
}
function hideWeather(){
  $.each( weatherMarkers, function( key, marker ) {
    marker.setMap(null);
  });
}
function showClouds(){
  var mapClouds = new google.maps.ImageMapType({
  getTileUrl: function (coordinates, zoom) {
   return "http://tile.openweathermap.org/map/clouds/" + zoom +
    + "/" + coordinates.x + "/" + coordinates.y + ".png"; },
```

Figure 11. A fragment of JavaScript code, displaying Weather and Clouds

After sending a query concerning the weather in iterator locations to OpenWeatherMap API, data is received in the JSON format, and is placed on the map using a function that allows adding new markers. The requests of pressure patterns and clouds are responded to directly by reading the current zoom level and position. The complete section of map is then downloaded in a .png graphic format and placed on the map.

Geolocation & GPS Info

The location is obtained through the use of the method depicted in the 'Materials and methods' section; however, to display additional information concerning geographic coordinates, speed, and heading it is necessary to add appropriate attributes to the method *navigator.geolocation.watchPosition(attribute)*.

The properties below are returned, if available (Table 3).

Table 3. Attributes for *watchPosition()* method

Attribute	Description
coords.latitude	The latitude as a decimal number
coords.longitude	The longitude as a decimal number
coords.heading	The heading as degrees clockwise from North
coords.speed	The speed in meters per second

The information from the attributes depicted above is updated on an ongoing basis (Figure 12).

The *watchPosition* method enables to establish a permanent location. Similarly to car navigation, the map is always centred on the current position, regardless of the map zoom level. This means that the map follows each movement and each change of position. Due to the impossibility of shifting the map in the geolocation mode during movement, a button with the *clearWatch* method, which stops the updating of positions and attributes, was introduced.

Discussion

The application has been tested using a desktop computer, laptop, tablet, smartphone and Internet access and has shown a satisfactory accuracy in obtaining position, both at home, via IP address, and outside, with the inbuilt GPS in mobile devices with Firefox browser for Android.

The application may have issues with the response speed of the navigation menu layout on mobile devices with resolution lower than XGA; however, the layout will be amended later on. Due to the limited possibility of obtaining data, the range of the application has been limited to the area of the Gdansk Bay. The comparison of the application with one of the major Internet services, marinetraffic.com, has shown significant discrepancies in terms of the precision of the position of the observed "Stena Vision" ship (Figure 13).

The comparison has been conducted at virtually the same time (5 seconds time lapse), refreshing, and zoom level 10 on both maps. The reference point was taken to be the Hel and Navigation Aid: BY HEL (Figure 14).

The conducted measurements have shown a discrepancy of around 1417 m = 0.77 nm, which

Figure 12. Result of the attributes

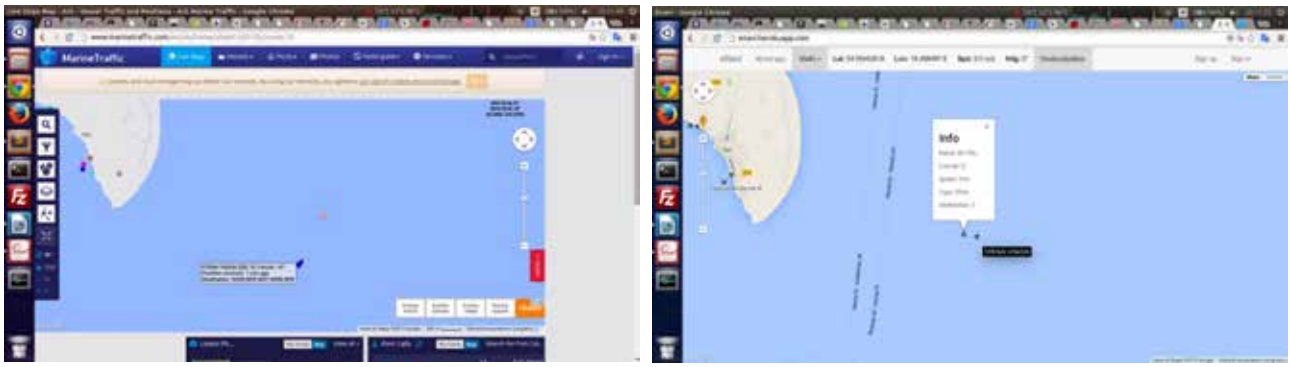

Figure 13. On the left – marinetraffic.com, on the right – enavi.herokuapp.com

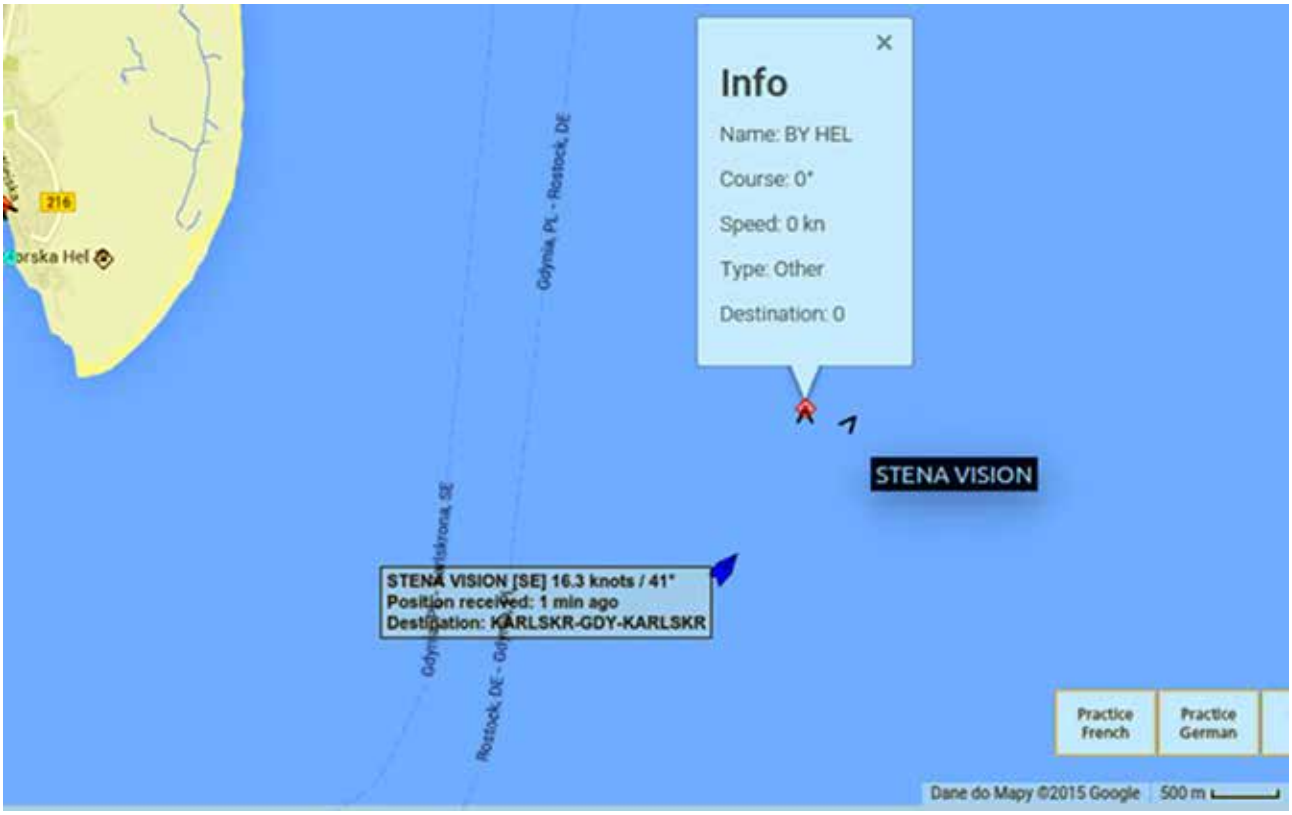

Figure 14. Comparison through merging images

constitutes an enormous difference. The measurements have also been conducted on vesselfinder.com, however the discrepancy was significantly lower, reaching 450 m = 0.24 nm. The live observation mode in case of both services is limited, and the markers have to be refreshed manually in order to obtain current data.

Conclusions

The application may be used to assist in amateur navigation and increase the safety in terms of sailing courses, water sports, kite-surfing and other sea activities. It may also be conditionally applied in emergencies, such as in case of damage to the deck GPS receiver, AIS-equipped system, and meteorological devices.

The starting development model allows modularity, which means that it is possible to easily add layers based on the existing solutions implemented in the application (like Customizing, Sign up, Sign in with user settings) as well as response speed for mobile applications.

The development model to be implemented is:

- Adding detailed navigation maps, or maps more oriented towards maritime navigation.

- Tracking and analysis/statistics of the travel routes saved with GPS module.
- Creating tracking history of ships within the range of the AIS system, movement analysis.
- The possibility to plan voyage, to draw routs and save them in the database.
- Adding more layers, mash-ups with convenient data/information.

Acknowledgments

We would like to thank Krzysztof Naus for sharing the data from the AIS server located in the Polish Naval Academy in Gdynia.

References

1. BHMW (2015) Official Polish Navigation Warnings page [Online] Available from: http://www.bhmw.mw.mil.pl/index.php?akcja=ostrzezenia [Accessed: May 16, 2016]
2. Google developers (2015) Official Google maps dev FAQ [Online] Available from: https://developers.google.com/maps/faq [Accessed: May 16, 2016]
3. ITU (2015) AIS technical *specification* [Online] Available from: https://www.itu.int/dms_pubrec/itu-r/rec/m/R-REC-M.1371-5-201402-I!!PDF-E.pdf [Accessed: May 16, 2016]
4. jQuery (2015), official page of jQuery library, http://api.jquery.com/ [Accessed: May 16, 2016]
5. Map types (2015) Google Maps API JavaScript documentation [Online] Available from: https://developers.google.com/maps/documentation/javascript/maptypes [Accessed: May 16, 2016]
6. Nokogiri (2015) Official Nokogiri's readme file [Online] Available from: http://www.rubydoc.info/github/sparklemotion/nokogiri [Accessed: May 16, 2016]
7. OpenWeatherMap (2015) Official OpenWeatherMap wiki [Online] Available from: http://bugs.openweathermap.org/projects/api/wiki/Upload_api [Accessed: May 16, 2016]
8. Ruby on Rails API (2015) Official page of Ruby on Rails framework [Online] Available from: http://api.rubyonrails.org/ [Accessed: May 16, 2016]
9. Ruby on Rails guide (2015) Official page of Ruby on Rails guide [Online] Available from: http://guides.rubyonrails.org/getting_started.html [Accessed: May 16, 2016]
10. W3C (2015) Geolocation API Specification [Online] Available from: http://dev.w3.org/geo/api/spec-source [Accessed: May 16, 2016]
11. XPath (2015), XPath wiki page [Online] Available from: https://en.wikipedia.org/wiki/XPath [Accessed: May 16, 2016]