

# Institut für Mathematik und Informatik

Fernuniversität Hagen

Vergleichende Implementierung und Evaluierung einer  
ereignisgesteuerten, nicht blockierenden I/O Lösung für eine  
datenintensive Real-Time Webanwendung in Javascript und Dart

Bachelorarbeit

Barbara Drücke

Matrikel-Nummer 7397860

<b>Betreuer</b>	Dr. Jörg Brunsmann
<b>Erstprüfer</b>	Prof. Hemmje
<b>Zweitprüfer</b>	Dr. Jörg Brunsmann

# Inhaltsverzeichnis

Abbildungsverzeichnis	IV
Tabellenverzeichnis	V
1 Einleitung	1
1.1 Motivation für diese Arbeit . . . . .	2
1.2 Aufbau der Arbeit . . . . .	3
2 Realtime-Schiffsverfolgung per AIS-Daten-Strom	4
2.1 Anwendungsfälle . . . . .	4
2.2 Beschreibung der Anforderungen . . . . .	4
2.3 Grobentwurf der Anwendung . . . . .	5
3 Grundlagen	7
3.1 Automatisches Informationssystem . . . . .	7
3.2 OpenStreetMap . . . . .	9
3.3 Leaflet . . . . .	9
3.4 Bidirektionale Kommunikation über HTML5 Websockets . . . . .	9
3.5 Node.js . . . . .	10
3.6 Google Dart . . . . .	11
4 Implementierungen	13
4.1 Strategie bei der Vorgehensweise . . . . .	13
4.2 Notwendige Strategie-Korrekturen . . . . .	13
4.3 Das Problem der Vergleichbarkeit . . . . .	14
4.4 Beschreibung der ausgeführten Implementierungen . . . . .	15
4.4.1 socket.io-Server . . . . .	15
4.4.2 socket.io-Client . . . . .	18
4.4.3 HTML5-Server . . . . .	21
4.4.4 HTML5-Client in Javascript . . . . .	21
4.4.5 HTML5-Client in Google Dart . . . . .	22
5 Ergebnisse	26
5.1 Evaluation der Anwendung . . . . .	26
5.2 Vergleichende Evaluation der Javascript- und der Dart-Anwendung . . . . .	28
5.2.1 Node.js - socket.io-Socket-Server vs. node.js-Websocket-Server . . . . .	29
5.2.2 Latenzzeit . . . . .	29
5.2.3 Browserunterstützung . . . . .	30
5.3 Javascript-Client vs. Dart-Client . . . . .	30
5.3.1 Implementierungsaufwand . . . . .	30

5.3.2	Performance . . . . .	30
5.3.3	Browserunterstützung . . . . .	33
6	Fazit . . . . .	34
6.1	Ergebnisse . . . . .	34
6.1.1	Die Realtime-Applikation . . . . .	34
6.1.2	Vergleich Javascript und Google Dart . . . . .	34
6.2	Ausblick . . . . .	34

## Abbildungsverzeichnis

1.1	Vesseltracker_Webapplikation . . . . .	1
1.2	Cockpit_Elbe . . . . .	2
2.1	Architektur-Entwurf der Realtime Webapplikation . . . . .	6
4.1	Übersicht Javascript-Files . . . . .	18
4.2	Übersicht Dart-Files . . . . .	22
5.1	Anzeige aller Schiffe im Hamburger Hafen . . . . .	26
5.2	Auf schnell fahrende Schiffe reduzierte Anzeige am Beispiel der Nordsee . .	27
5.3	Schleppmanöver . . . . .	27
5.4	Anwortzeiten des socket.io-Websocket-Servers in Abhängigkeit von der Anzahl verbundener Clients . . . . .	28
5.5	socket.io-Websocket-Server: Latenzzeit der Positionsmeldungen und Anzahl empfangener Schiffe . . . . .	29
5.6	HTML5-Websocket-Server: Latenzzeit der Positionsmeldungen und Anzahl empfangener Schiffe . . . . .	30
5.7	Dauer des Renders in Dartium . . . . .	32
5.8	Dauer des Renders in Chrome . . . . .	32
5.9	Dauer des Renders in Firefox . . . . .	32

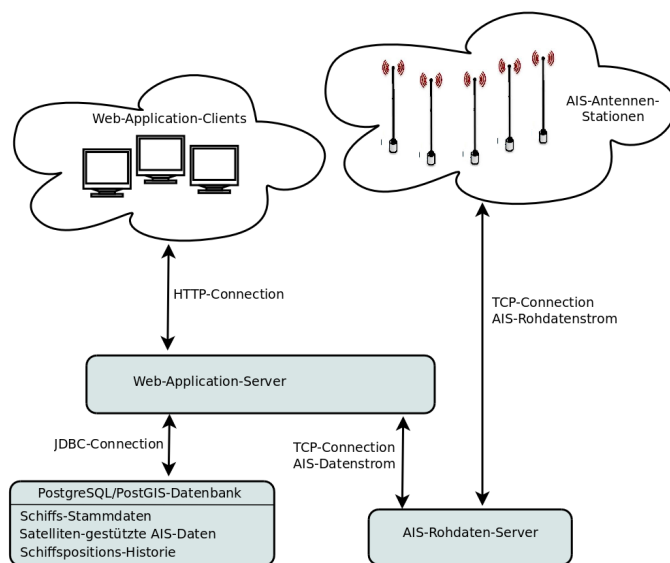
## Tabellenverzeichnis

3.1	Intervalle, in denen ein Schiff AIS-Nachrichten sendet . . . . .	8
4.1	Übersicht über Server-und Clientimplementierungen in dieser Arbeit . . . . .	14
5.1	Latenzzeit von Positionsmeldungen . . . . .	27

# 1 Einleitung

Die Vesseltracker.com GmbH ist ein Schiffsmonitoring und -reporting-Dienstleister. Der kostenpflichtige Dienst stellt den Kunden umfangreiche Informationen zu Schiffen weltweit zur Verfügung. Dabei handelt es sich einerseits um Schiffs-Stammdaten und andererseits um Schiffs-Positionsdaten. Die Positionsdaten sind AIS (Automatic Identification System) -Daten, wie sie von allen Schiffen über Funk regelmäßig zu senden sind.

Vesseltracker.com unterhält ein Netzwerk von ca. 800 terrestrischen AIS-Antennen, mit denen küstennahe AIS-Meldungen empfangen und via Internet an einen zentralen Rohdatenserver geschickt werden. Der Rohdatenserver verarbeitet die Meldungen und gibt sie umgewandelt und gefiltert an die Anwendungen des Unternehmens weiter. Zusätzlich erhält das Unternehmen AIS-Daten via Satellit über einen Kooperationspartner. Damit werden die küstenfernen Meeresgebiete und Gegenden, in denen Vesseltracker.com keine AIS-Antenne betreibt, abgedeckt. Die Kernanwendung des Unternehmens ist eine Webanwendung, die die terrestrischen AIS-Daten in einer Geodatenbank speichert und sie mit den Schiffs-Stammdaten und Satelliten-AIS-Daten in Beziehung setzt.



**Abbildung 1.1:** Vesseltracker\_Webanwendung

Für eine geographische Visualisierung der Schiffspositionen existiert das sogenannte 'Cockpit', wo die Schiffe als Icons auf Openstreetmap-Karten dargestellt werden. Diese Karte zeigt jeweils alle Schiffe an, die sich in dem frei wählbaren Kartenausschnitt zu der Zeit befinden. Aktualisiert werden die Positionsinformationen jeweils bei Änderung des betrachteten Bereichs oder einmal pro Minute. Detailinformationen erhält der Nutzer durch ein Click-Popup über das Icon des Schiffes. Darüber kann er sich auch die gefahrene Route der letzten 24 h anzeigen lassen.

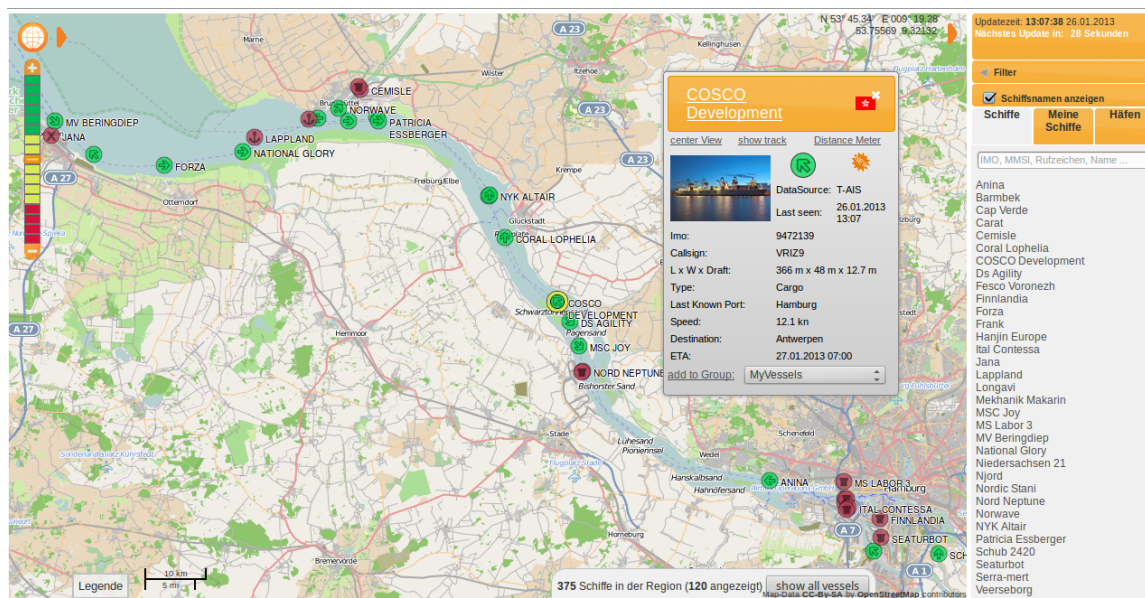


Abbildung 1.2: Cockpit\_Elbe

## 1.1 Motivation für diese Arbeit

Aus mehreren Gründen erscheint es angebracht, die geographischen Schiffspositionen nicht nur über die Cockpit-Anwendung anzubieten, sondern alternativ als real-time-Darstellung.

- Aufgrund der herausragenden Qualität des vesseltracker.com Antennen-Netzwerks sind die verfügbaren AIS-Daten aktuell, aktualisieren sich kontinuierlich und erreichen eine hohe weltweite Abdeckung. Damit ist es möglich, die Schiffsverkehrslage beliebiger Häfen, Wasserstraßen, Küstengebiete weltweit und sekundengenau zu präsentieren.
- Ein Phänomen in der menschlichen Wahrnehmung lässt die geplante Anwendung sehr viel zweckmäßiger erscheinen als die bisherige Cockpit-Anwendung. Aufgrund der sogenannten Veränderungsblindheit oder "Change Blindness" werden Veränderungen an einem Objekt (in diesem Fall die Position eines Schiffs-Icons auf der Karte) in der Wahrnehmung überdeckt, wenn im selben Augenblick Veränderungen an der Gesamt-sicht vonstatten gehen. Im Cockpit werden nach dem Laden neuer Positionsdaten alle Schiffsicons neu gerendert und unter Umständen Namens-Fähnchen gelöscht oder hinzugefügt, was zu einem kurzen "Flackern" führt. Dadurch ist es dem Betrachter nahezu unmöglich, die Positionsänderung eines Schiffes auf der Karte nachzuvollziehen.
- Real-time-Anwendungen gewinnen zunehmend an Bedeutung. Ihre Verbreitung wird durch den Fortschritt der verfügbaren Webtechnologien auf breiter Basis unterstützt. Mitbewerber auf dem Markt für AIS-Daten (z.B. Fleetmon.com) bieten bereits Echtzeit-Darstellungen ihrer AIS-Daten an. Um in diesem Geschäftsfeld weiterhin eine Spitzenposition innezuhaben, ist eine Realtime zwingend erforderlich, in einem nächsten Schritt sicher auch als Anwendung für Mobile Devices.

## 1.2 Aufbau der Arbeit

Im Kapitel 2 werden mögliche Anwendungs-Szenarien genauer beleuchtet und die funktionalen und nicht funktionalen Anforderungen an die geplante Anwendung herausgestellt. Anschließend wird die Systemarchitektur der geplanten Anwendung grob entworfen. In Kapitel 3 wird kurz auf die Grundlagen der verwendeten Technologien eingegangen: die AIS-Technologie, die zur Gewinnung des Datenmaterials verwendet wird und damit für Art und Format der Daten verantwortlich ist; node.js und Google Dart, die bei der Verarbeitung der Daten zum Einsatz kommen; HTML5-Websockets, weil sie für die Verteilung, MongoDB und Redis, weil sie für die Zwischenspeicherung der Daten eingesetzt werden. Im Zusammenhang mit der Datenpräsentation wird kurz das OpenStreetMap-Projekt vorgestellt als Lieferant des Kartenmaterials, sowie die Leaflet-Bibliotheken, mit deren Hilfe die Schiffsobjekte letztlich in Echtzeit auf der Karte angezeigt werden.

In Kapitel 4 werden zunächst die Gründe für die Auswahl an Implementierungen dargelegt. Anschließend wird die Vorgehensweise bei der Implementierung erläutert und zwar einmal ausführlich für die jeweils erste Server- bzw. Client-Implementierung (socket.io-Server und socket.io-Client) und anschließend nur dort, wo sich die Implementierungen im Kern unterscheiden. Die ausgearbeiteten Implementierungen werden dann in Kapitel 5 getestet und nach verschiedenen Aspekten verglichen. Kapitel 6 fasst die Ergebnisse zusammen und gibt einen Ausblick auf mögliche Weiterentwicklungen.



## **2 Realtime-Schiffsverfolgung per AIS-Daten-Strom**

### **2.1 Anwendungsfälle**

Hafendienstleister wie Schlepper, Lotsen oder Festmacher verschaffen sich über einen Monitor einen Überblick über die Arbeitsvorgänge in ihrem jeweiligen Heimathafen, z.B. welche Schlepper welches Schiff schleppen, wo Lotsen an oder von Bord gehen, von welchen Tankern Schiffe betankt werden. Sie kontrollieren die Ausführung der eigenen Aufträge oder auch die der Mitbewerber. Die Anwendung läuft hierbei eher statisch, das heißt, Zoomstufe und Kartenausschnitt ändern sich nur selten. Es ist also notwendig, dass die Anwendung sich unabhängig von Interaktionen regelmäßig aktualisiert.

Reedereien beobachten das Einlaufen, Anlegen, Festmachen oder Ablegen und Auslaufen ihrer Schiffe in entfernten Häfen, wo es keine Unternehmensniederlassung gibt. Zum Beispiel kontrollieren sie, wann, an welchen Liegeplätzen ein Schiff wie lange festmacht. Dazu ist es zum einen notwendig, jederzeit auf eine geringe Zoomstufe heraus- und auf einen anderen Hafen wieder hineinzoomen zu können. Zum anderen soll die Anwendung Schnittstellen bieten, damit zusätzliche Informationen aus dem vesseltracker.com Datenpool (in diesem Fall Liegeplatzinformationen) von der Anwendung abgerufen werden können.

Weitere Anwendungsfälle sind Nutzer aus der Passagierschiffahrt, Schiffsfotografen oder Sicherheitsorgane (z.B. die Wasserschutzpolizei), bei denen die Beobachtung / Überwachung bestimmter Wasserverkehrswege oder Häfen von besonderem Interesse ist.

Die vesseltracker.com GmbH nutzt die Realime-Anwendung, um die vom Unternehmen angebotenen Daten zu präsentieren und zu bewerben. Dabei ist es wichtig, dass die Anwendung gesendete AIS-Signale im Schnitt in weniger als einer Sekunde auf dem Monitor als Position oder Positionsänderung darstellen kann und dass die Schiffsbewegungen fließend ohne das in der Einleitung beschriebene "Flackern" dargestellt werden. Damit kann vesseltracker.com die höhere Genauigkeit und Aktualität der eigenen Daten gegenüber denen anderer Anbieter herausstellen.

Die Anwendungsfälle verdeutlichen noch einmal, dass der zusätzliche Nutzen der Realtimeanwendung gegenüber der Cockpitanwendung nicht ausschließlich im Informationsgehalt liegt. Die Daten im Cockpit sind ja ebenfalls im Minutenbereich aktuell. Der Vorteil liegt vielmehr in der Lebendigkeit der Darstellung. Bewegte Darstellungen binden stärker und für einen längeren Zeitraum die Aufmerksamkeit des Betrachters.

### **2.2 Beschreibung der Anforderungen**

Die funktionalen Anforderungen sind:

- als Datenquelle sollen ausschließlich die vom Rohdatenserver als JSON-Datenstrom zur Verfügung gestellten AIS-Informationen dienen
- Schiffe sollen an ihrer aktuellen (realtime) Position auf einer Karte im Browser dargestellt werden
- Positionsänderungen einzelner Schiffe sollen ad hoc sichtbar gemacht werden
- die Schiffsbewegungen auf der Karten sollen nicht sprunghaft, sondern fließend erscheinen (Animation der Schiffsbewegungen in dem Zeitraum zwischen zwei Positionsmeldungen)
- die Karte soll in 16 Zoomstufen die Maßstäbe von 1:2000 bis 1: 200 Mio abdecken
- Schiffe sollen auf der Karte als Icons dargestellt werden, die den Navigationsstatus und gegebenenfalls den Kurs widerspiegeln
- bei hoher Auflösung und ausreichend statischen AIS-Informationen soll ein Schiff als Polygon in die Karte eingezeichnet werden.
- bei geringer Auflösung ist ein Überblick über die Verteilung der empfangenen Schiffe zu vermitteln
- Detail-Informationen zu jedem Schiff sollen als Popups über das Icon abrufbar sein

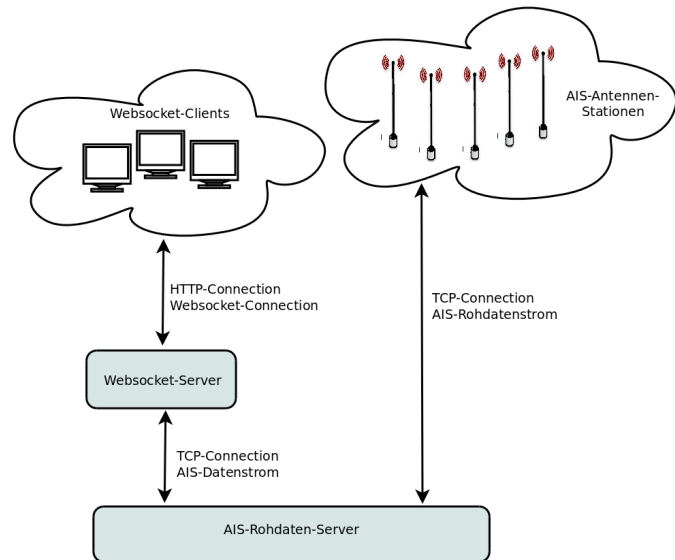
Nicht funktionale Anforderungen sind:

- die von den Antennen empfangenen AIS-Daten sind mit minimaler Verzögerung (< 500 msec) auf der Karte darzustellen
- die Anwendung sollte ca. 300 Verbindungen gleichzeitig erlauben und skalierbar sein
- als Clients der Anwendung sollten die gängigsten Browser unterstützt werden (IE, Chrome, Firefox, Safari, Opera)
- die Implementierungen werden auf Github als privates repository gehalten
- als Kartenmaterial sind die von vesseltracker gehosteten OpenstreetMap-Karten zu verwenden
- verwendete Software-Module sollten frei zugänglich sein (open source)
- ein Prototyp der Anwendung soll schnell zur Verfügung stehen. Dieser Prototyp soll Mitarbeitern und Partnern ermöglichen, ihre Anforderungen genauer zu spezifizieren oder sogar neue Anforderungen zu formulieren.

## 2.3 Grobentwurf der Anwendung

Die eingehende Schnittstelle der zu erstellenden Anwendung ist die Verbindung zum Rohdatenserver, die als TCP-Verbindung ausgeführt ist und einen JSON-Datenstrom liefert. Die ausgehende Schnittstelle ist der HTTP-Client (Browser). Zu erstellen ist also eine Client-Server-Anwendung, in der der Server zweifaches zu leisten hat, nämlich

1. eine tcp-socket-Verbindung zum Rohdatenserver zu unterhalten und
2. eine bidirektionale Verbindung zum HTTP-Client zu halten, in der der Client jederzeit Änderungen des betrachteten Kartenausschnittes an den Server senden und der Server jederzeit den Client über relevante, aus dem JSON-Datenstrom ausgelesene, Schiffsbewegungen im betrachteten Kartenausschnitt informieren kann.



**Abbildung 2.1:** Architektur-Entwurf der Realtime Webapplikation

## 3 Grundlagen

### 3.1 Automatisches Informationssystem

Das Automatic Identification System (AIS) ist ein UKW-Funksystem im Schiffsverkehr, das seit 2004 für alle Berufsschiffe über 300 BRZ in internationaler Fahrt und seit 2008 auch für solche über 500 BRZ in nationaler Fahrt verpflichtend eingeführt worden ist. Es soll dabei helfen, Kollisionen zwischen Schiffen zu verhüten und die landseitige Überwachung und Lenkung des Schiffsverkehrs zu erleichtern. Außerdem verbessert AIS die Planung an Bord, weil nicht nur Position, Kurs und Geschwindigkeit der umgebenden Schiffe übertragen werden, sondern auch Schiffsdaten (Schiffsname, MMSI-Nummer, Funkrufzeichen, etc.). AIS ist mit UKW-Signalen unabhängig von optischer Sicht und Radarwellenausbreitung [Wik13]. Für die Nutzung von AIS ist ein aktives, technisch funktionsfähiges Gerät an Bord Voraussetzung, das sowohl Daten empfängt als auch Daten sendet. Für Schiffe der Berufsschiffahrt sind Klasse-A-Transceiver an Bord vorgesehen, für nicht ausrüstungspflichtige Schiffe genügen Klasse-B-Transceiver, die mit niedriger VHF-Signalstärke und weniger häufig senden. Die dynamischen Schiffsdaten (LAT, LON, COG, SOG, UTC) erhält der AIS-Transceiver vom integrierten GPS-Empfänger, bei Klasse A auch von der Navigationsanlage des Schiffes. Die Kursrichtung (Heading = HDG) kann über eine NMEA-183-Schnittstelle vom Kompass eingespeist werden.

Die AIS-Einheit sendet schiffsspezifische Daten, die von jedem AIS-Empfangsgerät in Reichweite empfangen und ausgewertet werden können:

#### Statische Schiffsdaten

- IMO-Nummer<sup>1</sup>
- Schiffsname
- Rufzeichen<sup>2</sup>
- MMSI-Nummer<sup>3</sup>
- Schiffstyp (Frachter, Tanker, Schlepper, Passagierschiff, SAR, Sportboot u. a.)
- Abmessungen des Schiffes (Abstand der GPS-Antenne von Bug, Heck, Backbord- und Steuerbordseite)

---

1 <http://de.wikipedia.org/wiki/Schiffsnummer#IMO-Nummer>

2 <http://de.wikipedia.org/wiki/Schiffsnummer#Unterscheidungssignal>

3 <http://de.wikipedia.org/wiki/Schiffsnummer#MMSI>

## Dynamische Schiffsdaten

- Navigationsstatus (unter Maschine, unter Segeln, vor Anker, festgemacht, manövrierunfähig u. a.)
- Schiffsposition (LAT, LON, in WGS 84)
- Zeit der Schiffsposition (nur Sekunden)
- Kurs über Grund (COG)
- Geschwindigkeit über Grund (SOG)
- Vorausrichtung (HDG)
- Kursänderungsrate (ROT)

## Reisedaten

- aktueller maximaler statischer Tiefgang in dm
- Gefahrgutklasse der Ladung (IMO)
- Reiseziel (UN/LOCODE)<sup>4</sup>
- geschätzte Ankunftszeit (estimated Time of Arrival = ETA)
- Personen an Bord

Der Navigationsstatus und die Reisedaten müssen vom Wachoffizier manuell aktualisiert werden. Gesendet werden die AIS-Signale auf zwei UKW-Seefunkkanälen (Frequenzen 161,975 MHz und 162,025 MHz), wobei die Sendeintervalle abhängig sind von der Klasse, dem Manöverstatus und der Geschwindigkeit.

Klasse	Manöver-Status	Geschwindigkeit	Sendeintervall
Class A	geankert/festgemacht	<3kn	3 min
Class A	geankert/festgemacht	>3kn	10 sec
Class A	in Fahrt	0-14kn	10 sec
Class A	in Fahrt, Kursänderung	0-14	3 1/3 sec
Class A	in Fahrt	14-23kn	6 sec
Class A	in Fahrt, Kursänderung	14-23	2 sec
Class A	in Fahrt	>23kn	2 sec
Class B		<2 kn	3 min
Class B		>2 kn	30 sec

**Tabelle 3.1:** Intervalle, in denen ein Schiff AIS-Nachrichten sendet

Für AIS-Daten sind 22 standardisierte Nachrichtentypen bzw. Telegramme festgelegt. Für diese Arbeit interessieren nur die regulären Positionsmeldungen (dynamische Schiffsdaten) der Klasse-A-Transceiver (Typ 1-, 2- und 3-Messages) und die regulären Meldungen von (statischen) Schiffs- und Reisedaten der Klasse-A-Transceiver (Typ 5-Messages).

<sup>4</sup> <http://www.unece.org/cefact/locode/service/location.html>

### 3.2 OpenStreetMap

OpenStreetMap <sup>5</sup> ist eine freie, editierbare Karte der gesamten Welt auf der Basis von Daten, die von einer breiten Nutzergemeinde zusammengetragen werden. Inzwischen kann die Qualität der Karten mit denen proprietärer Angebote mithalten und übertrifft sie sogar in manchen Bereichen. Die Daten können gemäß der entsprechenden Freien Lizenz frei heruntergeladen und genutzt werden unter der Bedingung, dass sie nur unter der “Creative Commons Attribution-Share-Alike” (CC-BY-SA) -Lizenz weitergegeben werden.

### 3.3 Leaflet

Leaflet ist eine open-source JavaScript-Bibliothek <sup>6</sup> für interaktive Karten, die von einer Gruppe um Vladimir Agafonkin geschrieben wurde. Die Bibliothek zeichnet sich im Vergleich zu OpenLayers durch ein klares, schlankes Design aus und überzeugt durch gute Performance. Es unterstützt alle verbreiteten Plattformen auch im mobilen Bereich mithilfe von HTML5 und CSS3. Es ist hinreichend dokumentiert und verfügt über gut lesbaren Quellcode.

### 3.4 Bidirektionale Kommunikation über HTML5 Websockets

In der Entwicklung der Kommunikationstechnologien im Internet galt lange Zeit das request/response Paradigma, nach dem Anfragen vom Client vom Server beantwortet werden. Dieses Paradigma wird Stück für Stück aufgebrochen durch kontinuierliche Weiterentwicklungen in Richtung einer bidirektionalen Kommunikation zwischen Server und Client. Schon seit HTTP Long Polling, HTTP Streaming und Ajax on demand ist es für Serveranwendungen möglich nach einem initialen Verbindungsaufbau durch den Client, beim serverseitigen Eintreffen neuer Daten scheinbar selbständig einen Datenaustausch zum Client zu initiieren. Dabei handelt es sich eigentlich nur um einen aufgeschobenen response auf einen zuvor gestellten client-Request.

Der Nachteil dieser Technologien liegt darin, dass sie, weil sie Nachrichten über das HTTP-Protokoll austauschen, einen großen Überhang an Header-Informationen mitzusenden gezwungen sind, der sich in Summe negativ auf die Latenzzeit auswirkt [Var11]. Damit sind diese Technologien für zeitkritische (realtime) Anwendungen nicht unbedingt geeignet.

Das 2011 eingeführte WebSocket-Protokoll dagegen spezifiziert eine API (HTML5-WebSocket API-Spezifikation <sup>7</sup>), die eine echte bidirektionale Socket-Verbindung zwischen Server und Client ermöglicht, in der beide Seiten jederzeit Daten schicken können. Dieser Socket wird im Anschluss an einen initialen HTTP-handshake aufgebaut, indem Server und Client einen Upgrade der Verbindung auf das WebSocket-Protokoll aushandeln [MU12].

---

<sup>5</sup> <http://www.openstreetmap.org/>

<sup>6</sup> <http://leafletjs.com/reference.html>

<sup>7</sup> <http://www.w3.org/TR/2011/WD-websockets-20110419/>

### 3.5 Node.js

Node.js<sup>8</sup> ist ein Framework zur Entwicklung serverseitiger Webanwendungen in Javascript. Es wurde 2009 von Ryan Dahl veröffentlicht und hat seitdem viel Aufmerksamkeit erregt, weil Anwendungen in node.js

- hoch performant
- skalierbar
- und echtzeitfähig sind.

Diese Eigenschaften sind größtenteils dem Konzept des asynchronen, nicht blockierenden I/O von javascript im Allgemeinen und node.js im Besonderen geschuldet. Javascript ist von Anfang an asynchron konzipiert für die Verwendung im Webbrowser, wo synchrone Verarbeitung wegen der Verzögerung der Seitendarstellung nicht in Frage kommt. Den gleichen Ansatz übernimmt node.js für die Serverseite.

Node.js arbeitet single-threaded und eventbasiert. Die zentrale Kontrollstruktur, die den Programmablauf steuert, ist der Event-Loop. Er empfängt Events, die von Programm- oder Nutzeraktionen ausgelöst werden und setzt sie in Callback-Funktionen um. Kommt es im Programmablauf zur Interaktion mit einer externen Ressource, wird diese Interaktion in einen neuen Prozess ausgelagert und mit einer Callback-Methode versehen. Anschließend kann der Event Loop weitere aufgelaufene Events verarbeiten. Ist die Interaktion abgeschlossen, bekommt der Event Loop ein Signal und setzt beizeiten die Verarbeitung mit der entsprechenden Callback-Methode fort [Tei12].

Node.js bringt als Laufzeitumgebung die V8-Javascript-Engine mit, die die Ausführung von javascript-code durch Just-In-Time-Kompilierung optimiert. Außerdem bietet node.js eine direkte Unterstützung für das HTTP-Protokoll Websockets. Mit der Unterstützung des JSON-Datenformats sind alle notwendigen Bausteine zusammen für skalierbare, echtzeitfähige Serveranwendungen. Außerdem lassen sich mit dem Node Package Manager npm jederzeit weitere Pakete aus dem wachsenden Angebot nachinstallieren und verwalten.

Als konkrete Pakete für Websockets standen innerhalb von node.js zum Zeitpunkt der Implementierung (November 2012) die Bibliotheken websocket<sup>9</sup> und socket.io<sup>10</sup> zur Verfügung. Die Bibliothek websocket genügt der HTML5-Websocket-API-Spezifikation (s.o.). Socket.io erweitert die Funktionalität des websockets um heartbeats, timeouts und disconnection support. Außerdem kapselt socket.io die Details des Nachrichtenaustauschs: Bei Browsern, die Websockets nicht unterstützen, handelt socket.io die bestmögliche Verbindungsalternative aus in der Reihenfolge: → WebSocket → Adobe® Flash® Socket → AJAX long polling → AJAX multipart streaming → Forever Iframe → JSONP Polling<sup>11</sup>.

---

<sup>8</sup> <http://www.nodejs.org/>

<sup>9</sup> <https://github.com/Worlize/WebSocket-Node>

<sup>10</sup> <http://socket.io>

<sup>11</sup> <http://socket.io/#browser-support>

## 3.6 Google Dart

### Motivation für Dart

Dart ist eine von der Firma Google als OpenSource Projekt seit ca. 2 Jahren explizit für Webanwendungen entwickelte Programmiersprache. Das Ziel ist es, eine Sprache zu entwickeln, die komplexe Webanwendungen besser unterstützt als Javascript mit seinen historisch bedingten Ungereimtheiten und Schwächen. Das Entwicklerteam definiert die Design-Ziele folgendermaßen<sup>12</sup>: Dart soll

- eine sowohl strukturierte als auch flexible Web-Programmiersprache sein
- sich für Programmierer vertraut anfühlen und intuitiv erlernbar sein
- mit seinen Sprachkonstrukten performant sein und schnell zur Ausführung kommen
- auf allen Webdevices wie Mobiles, Tablets, Laptops und Servern gleichermaßen lauffähig sein
- alle gängigen Browser unterstützen.

### Spracheigenschaften von Dart

- Dart arbeitet **ereignisbasiert** und **asynchron** und in einem einzigen Thread ganz nach dem Vorbild von node.js.
- Dart läuft nativ in der **Dart-Virtual-machine**, kann aber auch nach Javascript kompiliert werden.
- **Klassen** sind ein wohlbekanntes Sprachkonzept zur Kapselung und Wiederverwendung von Methoden und Daten. Jede Klassen definiert implizit ein Interface.
- **Optionale Typisierung**: Die Typisierung in Dart ist optional, das heißt sie führt nicht zu Laufzeitfehlern. Sie ist als Werkzeug für den Entwickler gedacht, zur besseren Verständlichkeit des Codes und als Hilfe beim Debuggen.
- Die **Gültigkeitsbereiche** von Variablen in Dart gehorchen einfachen, intuitiv nachvollziehbaren Regeln: Variablen sind gültig in dem Block (...), in dem sie definiert sind.
- Zur **Parallelverarbeitung** nutzt Dart das Konzept von Isolates (übernommen von ER-LANG), eine Art Lightweight Processes. Isolates greifen nicht auf einen gemeinsamen Speicherbereich zu teilen nicht denselben Prozessor-Thread. Isolates kommunizieren miteinander ausschließlich über Nachrichten (über SendPort und ReceivePort). Sie werden gesteuert von einem übergeordneten Event Loop.
- Der **DartEditor** ist eine Entwicklungsumgebung für die Entwicklung von Dart Web- und Serverapplikationen. Sie beinhaltet das **Dart SDK** und den **Dartium Browser** mit der **Dart VM**.

---

<sup>12</sup> <http://www.dartlang.org/docs/technical-overview/#goals>



- Der **dart2js Compiler** ist ebenfalls im DartEditor enthalten und kompiliert Dart-Code zu Javascript-Code, der für die Chrome V8 Javascript engine optimiert ist.
- Mit **Pub** verfügt Dart über einen Package Manager vergleichbar dem Node Package Manager npm.

[Mü12][Lad12]

### Einbindung von Javascript-Bibliotheken in Dart mit js-interop

Für die Verwendung von Javascript-Bibliotheken in Dart-Code existiert die Dart-Bibliothek **js-interop**<sup>13</sup>. Damit können Dart-Anwendungen Javascript-Bibliotheken verwenden und zwar sowohl in nativem Dart code, der in der Dart-Virtual-Machine ausgeführt wird als auch in mit dart2js zu Javascript kompilierten Dart-Code.

Nachdem die Bibliothek in eine Dart-Anwendung eingebunden worden ist, kann ein sogenannter **Proxy** zum javascript-Kontext der Seite erstellt werden. Referenzen an diesen Proxy werden automatisch zu Javascript umgeleitet. Auf oberster Ebene lassen sich damit Javascript-Arrays und -Maps generieren, die mit den entsprechenden Objekten in Dart korrespondieren. Über diesen Proxy können aber auch Proxies zu beliebigen Javascript-Objekten erstellt werden, deren Eigenschaften und Methoden im Javascript-Scope zur Verfügung stehen.

Um Dart-Funktionen aus dem javascript-Scope heraus aufzurufen, wird die entsprechende Funktion in ein **Callback-Objekt** umgewandelt, das entweder ein einziges Mal oder mehrmals aufrufbar ist. Um die Lebensdauer dieser Proxies und Callback-Objekte zu verwalten benutzt Dart das Scope-Konzept: Per default haben alle proxies nur lokale Gültigkeit. Sollen sie den Ausführungszeitraum des Scopes überdauern, können sie ausdrücklich aufbewahrt werden, müssen dann aber zu Vermeidung von memory leaks auch explizit wieder freigegeben werden. Dasselbe gilt für Callback-Objekte, die mehrmals aufrufbar sind [Men12].

### Dart-Websockets

Für serverseitiges Dart, das auf der serverseitigen Dart-VM läuft existiert das Paket Dart:io. Es ermöglicht Zugriff auf das Dateisystem und auf Prozesse. In Dart:io existiert auch eine WebSocket-Implementierung,<sup>14</sup> mit der bereits einfache WebSocket-Server geschrieben werden können<sup>15</sup>.

<sup>13</sup> <https://github.com/dart-lang/js-interop>

<sup>14</sup> [http://api.dartlang.org/docs/releases/latest/dart\\_io/WebSocket.html](http://api.dartlang.org/docs/releases/latest/dart_io/WebSocket.html)

<sup>15</sup> <http://www.dartlang.org/docs/dart-up-and-running/contents/ch05.html>

## 4 Implementierungen

### 4.1 Strategie bei der Vorgehensweise

Zunächst wird eine Implementierung gewählt, die die besten Chancen hat, alle funktionalen und nicht funktionalen Anforderungen zu erfüllen. Dies ist eine Lösung in Javascript mit dem node.js-Framework und dem socket.io-Websocket (Abschnitt 4.4.1 und 4.4.2). Node.js-Serveranwendungen werden schon länger mit guten Ergebnissen in Netzwerken eingesetzt, besonders für Realtime-Anwendungen mit vielen gleichzeitig verbundenen Clients. Das socket.io-Paket wird genutzt, weil durch die Kapselung der verschiedenen Transportmechanismen die Bedienung einer maximalen Anzahl verschiedener Browser, auch in älteren Versionen möglich ist, ohne den Implementierungsaufwand unverhältnismäßig zu erhöhen. Die Implementierung dieser Lösung steht im Zeitplan ganz vorne, um der Anforderung von Unternehmensseite nach einer zeitnahen Umsetzung und Auslieferung zu entsprechen. In einem zweiten Schritt wird eine vergleichbare Implementierung in Google Dart ausgeführt. Die Entwicklung von Dart befindet sich noch in der Beta-Phase. Der zweite Beta-Release fand im Dezember 2012 statt. Ein dritter Beta-Release ist angekündigt. Ein zeitnaher ausschließlicher Einsatz von Dart im Produktivsystem wird also in absehbarer Zeit nicht möglich sein, weshalb diese Lösung als Investition in die Zukunft anzusehen ist. Der Vergleich beider Implementierungen (Javascript vs. Dart) ist deshalb nicht weniger interessant.

### 4.2 Notwendige Strategie-Korrekturen

Der ursprüngliche Plan, sowohl einen Server als auch einen Client in Dart zu schreiben, musste korrigiert werden, weil mit dem Dart-Websocket-Server einige der grundlegenden Anforderungen nicht umzusetzen waren. Zum einen unterstützt Dart keine JSON-over-TCP -Kommunikation, wie sie für die Abfrage des JSON-Datenstroms vom Rohdatenserver erforderlich ist. Und zum anderen gab es noch keinen Redis-Client für Dart. Der publish/-subscribe Mechanismus der Redis-Datenbank wird für die Verteilung der Positionsupdates benötigt. Deshalb wird nur die Clientanwendung in Dart implementiert (in Abschnitt 4.4.5). Dadurch ergibt sich ein weiteres Problem: Dart unterstützt Websockets nach der HTML5-Websocket-API-Spezifikation clientseitig mit dem Paket dart:html. Der socket.io-Websocket-Server entspricht nicht der HTML5-Websocket-API-Spezifikation und benötigt deshalb auf Clientseite zusätzliche Bibliotheken. Diese socket.io-client-Bibliotheken stehen in Dart nicht zur Verfügung. Also wird außer dem socket.io-Server ein zweiter Server implementiert, der eine Websocket-Verbindung nach der HTML5-Websocket-API-Spezifikation aufbaut (in Abschnitt 4.4.3). Die Umsetzung in node.js mit dem websocket-Modul <sup>1</sup> ist recht einfach möglich.

---

<sup>1</sup> <https://github.com/Worlize/WebSocket-Node/wiki/Documentation>

### 4.3 Das Problem der Vergleichbarkeit

An dieser Stelle stellt sich die Frage, ob beide Serverlösungen vergleichbare Ergebnisse liefern. Denn Unterschiede zwischen den node.js-Servern (socket.io vs. HTML5) würden in das Ergebnis des Vergleichs zwischen den Clients (Dart vs Javascript) einfließen. Deshalb wird für den HTML5-kompatiblen Websocket-Server noch ein Javascript-Client geschrieben, der dann direkt vergleichbar ist mit dem in Dart geschriebenen Client. Es bleibt noch zu beurteilen, inwieweit die Javascript-Implementierung durch den Verzicht auf das socket.io-Framework, das in der ersten Lösung verwendet wird, ausgebremst wird. Das geschieht über einen Performance-Vergleich in Abschnitt 5.2.1. Es werden also zwei Vergleiche durchgeführt:

- In Javascript wird der socket.io-Websocket gegen den HTML5-Websocket getestet.
- Unter Verwendung des HTML5-Websockets wird der Javascript-Client gegen den Dart-Client getestet

Tabelle 4.1 zeigt eine Übersicht über die Implementierungen. Auf diese Weise wird die präferierte Lösung in node.js mit socket.io nicht unmittelbar sondern mittelbar über die Javascript-Lösung mit HTML5-Websocket gegen die Implementierung mit einem Google Dart Client getestet.

		HTTP-Client	
		Javascript	Google Dart
HTTP-Server	socket.io-Websocket-Server (4.4.1)	socket.io-Client (4.4.2)	✗
	HTML5-Websocket-Server (4.4.3)	HTML5-Client (4.4.4)	HTML5-Client (4.4.5)

**Tabelle 4.1:** Übersicht über Server-und Clientimplementierungen in dieser Arbeit

## 4.4 Beschreibung der ausgeführten Implementierungen

In der ersten Implementierung werden Lösungen entwickelt für die in den Anforderungen beschriebenen Aufgaben. In allen weiteren Implementierungen werden diese Lösungen möglichst übernommen und andernfalls eine Alternative entwickelt.

### 4.4.1 socket.io-Server

Die zu entwickelnde Serveranwendung hat grundsätzlich zwei Aufgaben:

1. Daten über eine JSON-over-TCP-Verbindung vom Rohdatenserver abzurufen und
2. einen Websocket zu betreiben, der die Daten an Websocket-Clients weitergibt

Weil Node.js singlethreaded ist (vgl. 3.5), würde der Server beide Aufgaben in einem einzigen Prozess bearbeiten. Um das Potential an Parallelverarbeitung eines Dualcore oder Multicore-Servers zu nutzen, ist es daher sinnvoll, mindestens zwei Prozesse zu generieren. Dazu wurde das node.js-Modul `child_process` genutzt. Die Start-Datei `master.js` generiert damit zuerst einen Prozess, der den AIS-Daten-Client (`aisData-client.js`) abzweigt, um Daten vom Rohdaten-Server abzufragen und anschließend einen worker-Prozess (`worker.js`), um einen Websocket -Server für Client-Verbindungen zur Verfügung zu stellen (siehe Listing 4.1).

```
/* AIS-Client - Process */
child.fork(path.join(__dirname, 'aisData-client.js'));

/*worker- Process*/
child.fork(path.join(__dirname, 'worker.js'));
```

**Listing 4.1:** Generierung von Kindprozessen in `master.js`

Bei der Weitergabe der Daten durch den worker-Prozess sind zwei Fälle zu unterscheiden:

- ein Client verbindet sich neu oder ändert den Kartenausschnitt. In diesem Fall (Vessel-in-Bounds Request) sind die Schiffs-und Positionsdaten aller im Bereich (Bounds) befindlichen Schiffe an den Client zu senden (Kapitel 4.4.1).
- ein Schiff sendet ein Positions-Update, das an alle Clients verteilt werden muss, deren Kartenausschnitt die betreffende Schiffsposition enthält. Dieses Ereignis wird im Folgenden Vessel-Position-Update genannt (Kapitel 4.4.1).

### Vessels-in-Bounds-Request

Der Vessels-in-Bounds-Request macht eine Zwischenspeicherung der Daten unumgänglich. Wegen der großen Anzahl gleichzeitig empfangener Schiffe (weltweit ca. 60.000) und der Notwendigkeit, einen geographischen Index zu verwenden, wird einer persistenten gegenüber einer transienten Speicherung der Vorzug gegeben.

Für die Persistierung wird hier MongoDB verwendet, weil MongoDB als NoSQL-Datenbank mit geringem Overhead schnelle Antwortzeiten und außerdem einen geographischen Index bietet. Der Serverprozess in `aisData-client.js` schreibt die Daten (siehe listing 4.2) in eine

MongoDB-Collection namens 'vesselsCollection'. Die MMSI eines Schiffes ist eindeutig und wird als unique key verwendet (siehe Abschnitt 3.1). Über die Option `upsert:true` wird der Mongo Datenbank mitgeteilt, dass entweder ein `insert`-Befehl oder, falls die `mmsi` bereits in der MongoDB-Collection vorhanden ist, ein `update`-Befehl auf das entsprechende set auszuführen ist.

```
vesselsCollection.update(
  { mmsi: obj.mmsi },
  { $set: obj },
  { safe: false, upsert: true }
);
```

**Listing 4.2:** Schreiben in die Datenbank in `aisData-client.js`

In Listing 4.3 ist zu sehen, wie über das Schlüsselwort "2d" der MongoDB-Geo-Index auf das Feld 'pos' mit den Koordinaten des Schiffes gesetzt wird. Damit ist garantiert, dass nicht jede Anfrage des Servers an die Datenbank sämtliche Datensätze durchlaufen muss, um die Schiffe in einem bestimmten geographischen Ausschnitt zu finden. Aufbau und Unterhalt des Geo-Indexes findet im `aisData-client`-Prozess statt, der schreibend auf die Datenbank zugreift.

```
vesselsCollection.ensureIndex({ pos: "2d", sog: 1, time_received: 1 },
  function(err, result) {... });
```

**Listing 4.3:** Aufbau des Geo-Indexes in `aisData-client.js`

Dabei handelt es sich um einen zusammengesetzten Index, weil neben der Geo-Position auch die Geschwindigkeit und der Zeitpunkt des Empfangs der Nachricht Filterkriterien sind, wenn der zweite Prozess (`worker.js`) lesend auf die Datenbank zugreift. In Listing 4.4 ist zu sehen, wie der Prozess `worker.js` mit den vom Client in einem `Vessels-in-Bounds-Request` übermittelten Geo-Daten die `MongoDb` anfragt.

```
var vesselCursor = vesselsCollection.find({
  pos: { $within: { $box: [ [bounds._southWest.lng,bounds._southWest.lat],
    [bounds._northEast.lng,bounds._northEast.lat] ] } },
  time_received: { $gt: (new Date() - 10 * 60 * 1000) },
  sog: { $exists:true },
  sog: { $gt: zoomSpeedArray[zoom]},
  sog: {$ne: 102.3}
});
vesselCursor.toArray(function(err, vesselData) {
  client.sendUTF(JSON.stringify({ type: 'vesselsInBoundsEvent', vessels:
    vesselData}));
});
```

**Listing 4.4:** Vessel-in-Bounds-query in `worker.js`

## Vessel-Position-Update

Für die Kommunikation eines Vessel-Position-Updates (AIS-Nachrichtentyp 1-3) zwischen dem `aisData-client.js`-Prozess und dem `worker.js`-Prozess wird der `publish/subscribe`-Mechanismus einer Redis-Datenbank genutzt. Der `aisData-client.js`-Prozess publiziert jedes Positions-Update auf dem Kanal 'vessel-Pos' der Redis-Datenbank. Der `worker.js`-Prozess meldet

sich als subscriber am Kanal 'vessel-Pos' der Redis-Datenbank an und wird so bei jedem Positions-Update benachrichtigt. Um diese Nachricht an die betroffenen Websocket-Clients weiterzuleiten, ist eine serverseitige Verwaltung der Clients notwendig. Die Serveranwendung muss bei jeder Positionsmeldung wissen, welche Clients benachrichtigt werden müssen. Die Client-Verwaltung ist ein Feature des socket.io-Paketes. Für jeden Client wird bei der Registrierung zusätzlich das Zoomlevel der Karte und die Koordinaten gespeichert.

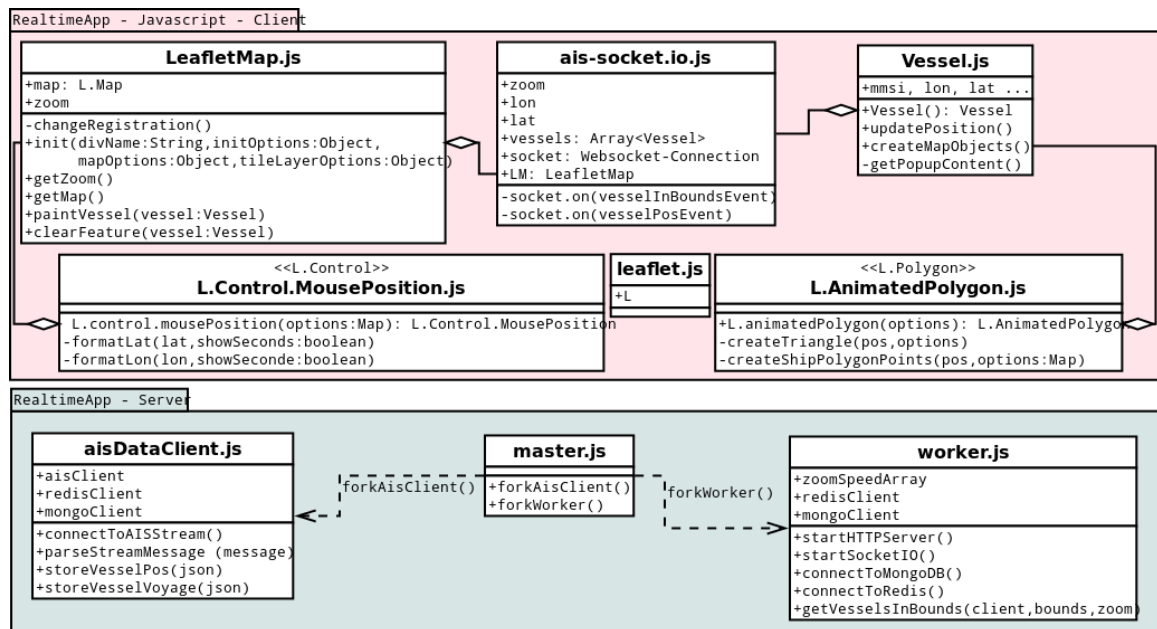
```
io.sockets.on('connection', function(client) {
  log(' Connection from client accepted. ');
  client.on('register', function(bounds, zoom) {
    client.set('zoom', zoom);
    client.set('bounds', bounds, function() {
      getVesselsInBounds(client, bounds, zoom);
    });
  });
  client.on('unregister', function() {
    client.del('bounds');
    client.del('zoom');
  });
});
```

**Listing 4.5:** Speichern der übermittelten Client-Daten in worker.js

Bei jedem Vessel-Position-Update, das der worker.js-Prozess empfängt, geht er die Liste der Clients durch und benachrichtigt diejenigen, in deren Bereich das Positions-Update fällt.

```
redisClient.on('message', function(channel, message) {
  if (channel == 'vesselPos') {
    ...
    var json = JSON.parse(message);
    ...
    var clients = io.sockets.clients();
    var lon = json.pos[0];
    var lat = json.pos[1];
    var sog = json.sog/10;
    var cog = json.cog/10;
    clients.forEach(function(client) {
      client.get('bounds', function(err, bounds) {
        if (bounds != null && lon != null && lat != null)
        {
          /* check, if Client-Connection is affected by Vessel-Position-
             Update */
          if (positionInBounds(lon, lat, bounds))
          {
            client.get('zoom', function(err, zoom)
            {
              if(sog !=null && sog > (zoomSpeedArray[zoom]) && sog !=
                102.3)
              {
                client.emit('vesselPosEvent', message);
              }
            }
          }
          ...
        }
      });
    });
  }
});
```

**Listing 4.6:** Weiterleitung von Positions-Updates an Websocket-Clients in worker.js



**Abbildung 4.1:** Schematische Übersicht der implementierten Javascript-Dateien und -Objekte. In der Darstellung wird die Notation des UML-Klassendiagramms verwendet, was natürlich (Klassen existieren nicht in Javascript) nicht als solches gelesen werden kann. Weil aber die Anwendung obektorientiert konzipiert ist, war dieser Ansatz sowohl während der Entwicklung als auch bei der Beschreibung der Anwendung sehr nützlich. Die Attribute und Methoden sind nicht vollständig dargestellt, sondern so ausgewählt, dass Zusammenhänge und Abhängigkeiten erkennbar werden. Dasselbe gilt für Abbildung 4.2, die den Dart-Client schematisch darstellt.

#### 4.4.2 socket.io-Client

Der socket.io-Client hat folgende Aufgaben.

1. Es ist eine html-Seite zu erstellen, die die benötigten Source-Dateien lädt und eine HTML-Struktur aufbaut mit den DOM-Elementen für die Karte.
2. URL-Parameter sollen optional übergeben werden können.
3. Optionen sollen zentral an einer Stelle der Anwendung geändert werden können.
4. Eine Karte auf Basis des auf dem unternehmenseigenen Server gehosteten Kartenmaterials mit Navigations- und Infobereichen ist in den Kartenbereich zu rendern.
5. Zum Socket.io-Websocket-Server soll eine Verbindung aufgebaut werden, die
  - 5.1 Vessel-In-Bounds- und Vessel-Position-Events empfängt und
  - 5.2 bei Positionsänderungen auf der Karte eine register-Nachricht sendet
6. Aus den empfangenen JSON-Daten sind geeignete Objekte zu erstellen und zu speichern.
7. Die Objekte sind als Features auf die Karte zu rendern
8. Objekte, die den Status 'Moving' haben, sind entsprechend ihrer Geschwindigkeit zu animieren.

Punkt 1 geschieht in der Datei `ais-socket.io.html`, die zum Start der Anwendung vom Browser-Client aufgerufen wird. Dort werden die benötigten javascript- und css-Dateien eingebunden, inklusive der JQuery- und der Leaflet-Bibliothek. Nach dem Laden (`on document.ready()`) führt der Browser die Datei `ais-socket.io.js` aus. Falls Url-Parameter übergeben worden sind für initialen Zoomlevel und Kartenausschnitt (Punkt 2) werden sie nun mit der Funktion `getParam(name)` aufgegriffen, sonst wird ein Defaultwert für `zoom` und `bounds` benutzt. Wie unter Punkt 3 gefordert, kann dieser Defaultwert und alle weiteren Einstellungen (z.B. Server-IP, Server-Port, Map-Server-Url (Punkt 4)) in dieser Datei zentral angepasst werden. Als Javascript-Bibliothek zur Darstellung der Schiffe auf der Karte wurde die Leaflet-Bibliothek (siehe Abschnitt 3.3) ausgewählt. Die Cockpit-Anwendung der Vesseltracker-GmbH nutzt die älteren OpenLayers-Bibliotheken, diese sind jedoch im Vergleich sehr viel sperriger in der Nutzung und werden inzwischen weniger aktiv von der Community weiterentwickelt und verbessert. Mithilfe dieser Bibliothek wird die Karte als Javascript-Objekt in der Datei `LeafletMap.js` realisiert. Dazu wird das ‘Revealing Module Pattern’ genutzt, mit dem sich die API der Karte von ihrer internen Implementierung trennen lässt. Nur die in der `return`-Klausel zurückgegebenen Funktionen bilden die öffentliche Schnittstelle (Punkt 4).

```
var LMap = function(){
    var map, featureLayer, tileLayer, zoom, socket, boundsTimeout,
        boundsTimeoutTimer;
    function init(elementid, initOptions, mapOptions, tileLayerOptions) { ... }
    function changeRegistration() { ... }
    function getMap(){ ... }
    function getZoom(){ ... }
    function addToMap(feature, animation, popupContent){ ... }
    function removeFeatures(vessel){ ... }
    return {
        init: init,
        getMap: getMap,
        getZoom: getZoom,
        addToMap: addToMap,
        removeFeatures: removeFeatures }
}();
```

**Listing 4.7:** ‘Revealing Module Pattern’ in `LeafletMap.js`

Nach dem Initialisieren der Karte wird die Websocket-Verbindung (Punkt 5) hergestellt (siehe listing 4.8). Für den Empfang der Nachrichten des Websocket-Servers (Punkt 5a, siehe auch Kapitel 4.4.1) genügen dazu zwei Listener:

```
var socket = io.connect('http://' + WEBSOCKET_SERVER_LOCATION + ':' +
    WEBSOCKET_SERVER_PORT);
socket.on('vesselsInBoundsEvent', function (data) {...}
socket.on('vesselPosEvent', function (data) {...}
```

**Listing 4.8:** Client-Seite der `socket.io`-Websocket-Verbindung in `ais-socket.io.js`

Um eine Liste aller im Kartenbereich befindlichen Schiffe vom Server zu bekommen, muss der Client eine ‘register’-Nachricht mit den aktuellen Bounds an den Server senden (Punkt 5b, siehe dazu 4.5). Dies geschieht einmal nach dem Initialisieren der Karte und soll anschließend durch den von der Leaflet-Map nach jeder Änderung des Kartenausschnitts getriggerten



moveend-Event ausgelöst werden, bzw. spätestens nach der als `BOUNDS_TIMEOUT` übergebenen Zeitspanne. Weil dieser Event innerhalb des `LeafletMap`-Objektes auftritt, wird dem `LeafletMap`-Objekt bei der Initialisierung eine Referenz auf die Websocket-Connection übergeben. Diese Lösung ist unproblematisch, weil beim Verlust der socket-Verbindung ohnehin ein Reload der Seite stattfinden muss.

Um geeignete Objekte aus den AIS-Messages zu erstellen (Punkt 6), wird in der Datei `Vessel.js` eine Konstruktor-Funktion zur Verfügung gestellt, mit der Instanzen von `Vessel`-Objekten generiert werden können. Diese Instanzen werden in einem assoziativen Array, also einem Objekt, namens `'vessels'` in `ais-socket.io.js` gespeichert. Beim Empfang eines `Vessel-Position-Events` kann mit `vessels[mmsi]` nach dem passenden `vessel`-Objekt zum Update gesucht werden. Schließlich sind die `Vessel`-Objekte auf die Karte zu rendern (Punkt 7). Dazu wird in `Vessel.js` eine asynchrone Funktion genutzt (siehe Listing ??), mit der zuerst je nach Status (`moving` / `not moving`) und `zoomlevel` unterschiedliche Features für ein Schiff erstellt werden (`Polygon`, `Triangle`, `Speedvector`, `Circle`). Anschließend werden die Map-Features des `vessel`-Objekts auf die Karte gerendert und zusammen mit dem Objekt gespeichert. Das Speichern ist notwendig, um die Features bei `Vessel-Position-Updates` von der Karte zu entfernen, bevor sie an eine neue Position gerendert werden.

```
vessel.paintToMap(LMap.getZoom(), function(){
    vessels[vessel.mmsi] = vessel;
});
```

**Listing 4.9:** Aufruf der public function `paintToMap` des `Vessel`-Objekts in `ais-socket.io.js`

Für die letzte Aufgabe, die Animation (Punkt 8), wird das `Polygon`-Objekt des `Leaflet`-Frameworks erweitert zur Klasse `L.AnimatedPolygon`. Ein `Polygon` in `Leaflet` ist eine `Polyline`, die mehrere Punkte auf der Karte verbindet. Die Animation eines Punktes entlang einer Linie mit einer bestimmten Geschwindigkeit wurde aus dem `Leaflet`-Plugin `L.AnimatedMarker`<sup>2</sup> übernommen. Die dort präferierte `css3`-Transition zur Animation konnte aber nicht verwendet werden, weil dazu ein Objekt im DOM-Baum identifiziert werden muss. Das von `Leaflet` für ein `Polygon` erstellte DOM-Objekt (`svg-Graphik`) ist aber durch die `Leaflet`-Bibliothek gekapselt. Wohl deswegen ist der Versuch gescheitert.

Ein `Schiffspolygon` wird berechnet aus der `Schiffsposition` (Positionsangabe in der AIS-Nachricht) und aus der relativen Position des AIS-Transceivers an Bord (Abstand zum Bug, zum Heck, nach Backbord und nach Steuerbord). Um zu wissen, in welche Richtung der Bug eines Schiffes zeigt, wird die AIS-Angabe zur Fahrtrichtung verwendet. Dies ist entweder `cog` (= `Course over Ground`) bei fahrenden Schiffen oder `true_heading` bei liegenden Schiffen. Aus dieser Richtungsangabe und der übermittelten Geschwindigkeit (`sog` = `Speed over Ground`) wird ein `Speedvector` berechnet, der als Linie auf die Karte gezeichnet wird. Er ist genauso lang wie die Strecke, die das Schiff bei kontinuierlicher Fahrt in den nächsten 30 Sekunden zurücklegen wird. Bei der Erstellung des `AnimatedPolygon`-Objektes wird dieser `Speedvector` übergeben, damit das `Polygon` entlang dieser Linie verschoben werden kann. Aus den Optionen `distance` und `interval` wird bestimmt, in wieviele Teilschritte der Vektor unterteilt wird und wieviel Zeit zwischen zwei Animationsschritten liegt. Nach jedem Animationsschritt wird das `Polygon` neu berechnet, von der Karte gelöscht und neu gezeichnet.

<sup>2</sup> <https://github.com/openplans/Leaflet.AnimatedMarker>

### 4.4.3 HTML5-Server

Diese Server-Implementierung soll genau dieselbe Funktionalität besitzen wie die socket.io-Server-Implementierung (4.4.1). Lediglich der Websocket socket.io wird durch einen node.js-Websocket nach der HTML5-Websocket-Spezifikation ausgetauscht<sup>3</sup>. Dazu wird in der Datei worker.js das entsprechende Paket ('websocket') eingebunden. Einige Features des socket.io-Paketes müssen jetzt selbst organisiert werden:

- die Clientverwaltung erfolgt in einem Array 'clients', in dem zu jeder Zeit alle verbundenen Clients mit ihren Attributen stehen.
- in der Websocket können keine eigenen Events definiert werden (siehe API-Dokumentation<sup>4</sup>). Deshalb wird der message-Event genutzt und der Name der aufzurufende Funktion wird innerhalb der message als "type" übermittelt. Unten ist eine identische Nachricht in den beiden unterschiedlichen Formaten zu sehen.

```
message origin=ws://127.0.0.1:8090, data={"type":"vesselsInBoundsEvent",
  "vessels":[{"_id":"50d9fdb4bcc2e678a9278c18","aisclient_id":57,"callsign":"
  OVYC2 ", "cog":285,"dest":"HAMBURG ", "dim_bow":70,"dim_port":10, "
  dim_starboard":4,"dim_stern":30,"draught":54,"imo":"9363170","mmsi"
  :220515000,"msgid":1,"name":"RIKKE THERESA ", "nav_status":0,"pos"
  :[9.8375,53.54885], "rot":0,"ship_type":80,"sog":10.3,"time_captured"
  :1366733896000,"time_received":1366733855248,"true_heading":286}]}
```

**Listing 4.10:** vom Websocket-Server gesendete message

```
[{"_id":"50d9fdb4bcc2e678a9278c18","aisclient_id":57,"callsign":"OVYC2 ", "
  cog":285,"dest":"HAMBURG ", "dim_bow":70,"dim_port":10, "
  dim_starboard":4,"dim_stern":30,"draught":54,"imo":"9363170","mmsi"
  :220515000,"msgid":1,"name":"RIKKE THERESA ", "nav_status":0,"pos"
  :[9.8375,53.54885], "rot":0,"ship_type":80,"sog":10.3,"time_captured"
  :1366734056000,"time_received":1366733855248,"true_heading":286}]}
```

**Listing 4.11:** vom socket.io-Server gesendete message

### 4.4.4 HTML5-Client in Javascript

Die socket.io-Client-Implementierung und die HTML5-Client bieten die gleiche Funktionalität und arbeiten nur geringfügig unterschiedlich.

- wie in Listing 4.10 zu sehen, muss der HTML5-Client zuerst den message-type abfragen, um die Daten korrekt zuzuordnen.
- weil dem Leaflet-Map-Objekt die Websocket-Connection als Parameter übergeben wird, muss der Aufbau der Websocket-Connection abgewartet werden. Deswegen wird die Map-Initialisierung erst durch den onopen-Event des Websockets angestoßen.

<sup>3</sup> <https://github.com/Worlize/WebSocket-Node>

<sup>4</sup> <https://github.com/Worlize/WebSocket-Node/wiki/Documentation>

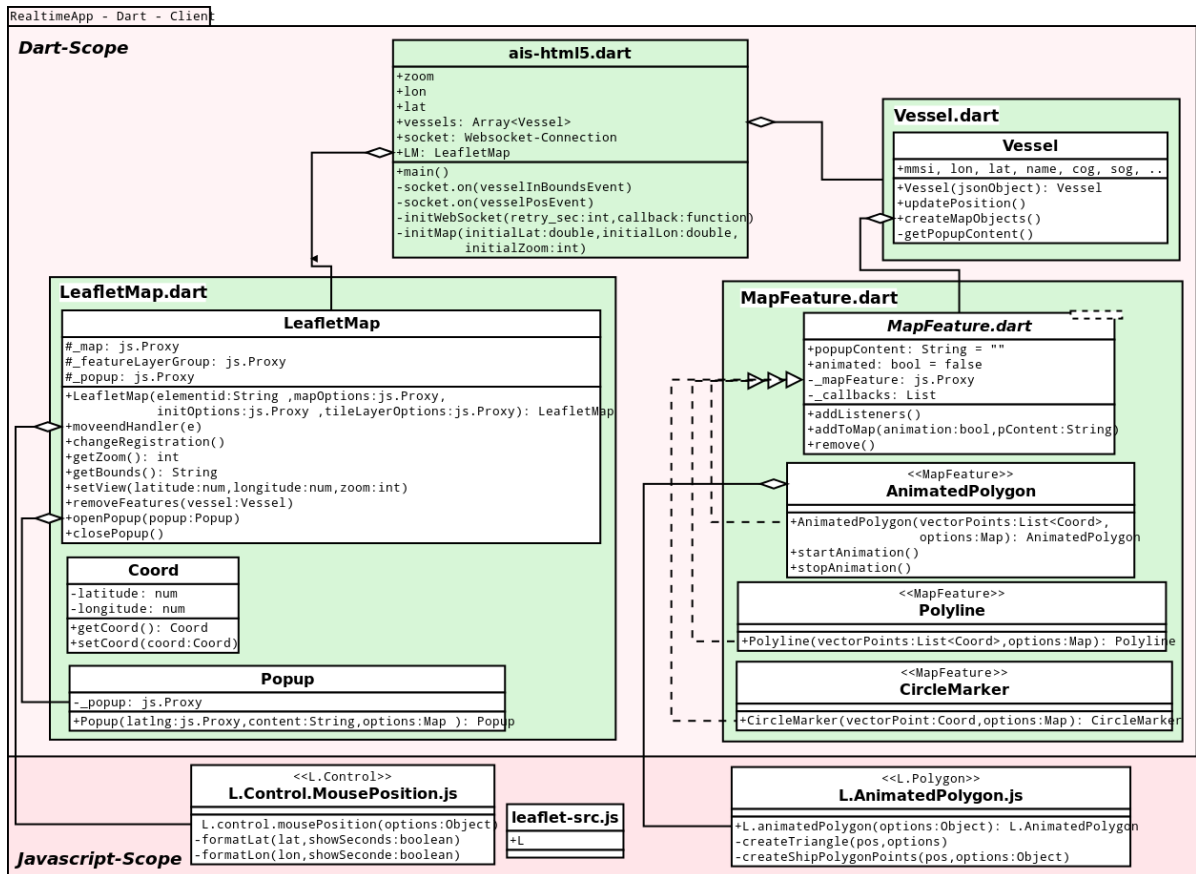


Abbildung 4.2: Schematische Übersicht der Dart-Dateien und -Klassen, sowie der verwendeten Javascript-Dateien, siehe auch Anmerkungen zu Abb. 4.1

#### 4.4.5 HTML5-Client in Google Dart

Die Implementierung des HTML5-Websocket-Clients in Dart orientiert sich an der Implementierung des HTML5-Websocket-Clients in Javascript. Oberstes Ziel dabei ist es, eine mindestens gleichwertige Funktionalität zu erreichen unter Ausnutzung der sprachspezifischen Vorteile von Google Dart.

Die Modularisierung, also die Verteilung der Objekte und Funktionalitäten auf mehrere Dateien, ist in der Dart-Client-Implementierung ähnlich gestaltet wie in Javascript (siehe Abbildungen 4.1 und 4.2). Die verwendeten Javascript-Dateien (leaflet-src.js, L.AnimatedPolygon.js und L.Control.Mouseposition.js) sind identisch mit denen, die der Javascript-Client (Abschnitt 4.4.4) verwendet. Um sie aus Dart heraus nutzen zu können, wird das Dart-Paket js-interop (siehe Abschnitt 3.6) eingebunden. Eine Dart-Anwendung wird über die main-Funktion (in ais-html5.dart) gestartet. Die main-Funktion initialisiert den HTML5-Websocket-Client. Im Falle eines erfolgreichen Verbindungsaufbaus mit dem HTML5-Websocket-Server wird die Callback-Funktion ausgeführt, die das LeafletMap-Objekt erstellt. Beide Objekte sind Singletons und bleiben über die Laufzeit der Anwendung erhalten. Das LeafletMap-Objekt kapselt für die Anwendung den Zugriff auf das Javascript L.Map-Objekt der leaflet.js-Bibliothek (siehe Listing 4.12). Dafür wechselt es in den Javascript-Scope der Anwendung und initialisiert in diesem ein L.Map-Objekt und ein L.LayerGroup-Objekt. Beide Objekte werden mit der Anweisung js.retain(...) im Javascript-Scope als globale Variable eingeführt, so dass sie für jede folgende Funktion, die den Javascript-Scope betritt, zur Verfügung stehen.

In der Gegenrichtung muss eine Möglichkeit existieren, vom Javascript-Scope der Anwendung auf den Dart-Code zuzugreifen. Ein Beispiel dafür ist der ‘moveend’-Listener, der Teil der leaflet.js-Bibliothek ist. Für den Javascript-Listener wird ein Dart-Callback.many-Objekt erstellt (moveendHandler(e)), das die changeRegistration-Funktion aus dem Dart-Scope ausführt. Das Callback.many-Objekt in Dart kann im Unterschied zum Callback.once-Objekt mehrmals aufgerufen werden. Das heißt, jedes Mal, wenn im Javascript-Scope der ‘moveend’-Listener einen ‘moveend’-Event registriert, ruft er über den Handler die Dart-Funktion changeRegistration() auf. Diese Funktion sendet an den HTML5-Websocket eine Nachricht vom Typ ‘register’ mit den aktuellen Bounds der Karte.

```
LeafletMap(String elementid, js.Proxy mapOptions, js.Proxy initOptions, js.
Proxy tileLayerOptions){
  boundsTimeout = initOptions['boundsTimeout']*1000;
  js.scoped(() {
    map = new js.Proxy(js.context.L.Map, elementid, mapOptions);
    featureLayerGroup = new js.Proxy(js.context.L.LayerGroup);
    map.addLayer(featureLayerGroup);
    js.retain(featureLayerGroup);
    var tileLayer = new js.Proxy(js.context.L.TileLayer, tileLayerOptions[
      'tileURL'], tileLayerOptions);
    map.addLayer(tileLayer);
    var mouseOptions = initOptions['mousePosition'];
    if( mouseOptions != false)
    {
      var mousePosition = new js.Proxy(js.context.L.Control.MousePosition
        , mouseOptions);
      mousePosition.addTo(map);
    }
    map.setView(new js.Proxy(js.context.L.LatLng, initOptions['lat'],
      initOptions['lon']), initOptions['zoom']);
    js.retain(map);
    map.on('moveend', new js.Callback.many(moveendHandler));
  });
  changeRegistration();
}
```

**Listing 4.12:** Konstruktor des LeafletMap-Objektes mit Zugriff auf den Javascript-Scope

Der Websocket-Server antwortet auf diese Nachricht mit einer Nachricht vom Typ “VesselInBoundsEvent” (siehe Listing 4.10).

Diese Nachricht und die “VesselPositionEvent”-Nachricht empfängt und verarbeitet der Dart-Client genauso wie der Javascript-Client in Listing 4.9. Bei der Ausführung der paintToMap-Methode des Vessel-Objektes allerdings besteht ein wichtiger Unterschied darin, dass in Javascript direkt Map-Feature-Objekte der Leaflet.js Bibliothek (L.Polyline, L.AnimatedPolygon und L.CircleMarker) erzeugt werden, wohingegen in Dart der Konstruktor einer Unterklasse (Polyline, AnimatedPolygon oder CircleMarker) von MapFeature aufgerufen wird. Die Konstrukturen aller Unterklassen von MapFeature wechseln in den Javascript-Scope, erstellen dort über einen Proxy ein entsprechendes Objekt aus der Leaflet-Bibliothek und stellen dieses als Attribut des MapFeature-Objektes im Dart-Scope zur Verfügung (siehe Beispiel Klasse Polyline in Listing 4.13).

```
class Polyline extends MapFeature{
  Polyline(List<Coord> vectorPoints, Map options) {
    js.scoped(() {
```

```

var latlng = js.context.L.LatLng;
var points = js.array([]);
for(var x = 0; x < vectorPoints.length; x++)
{
    var lat = vectorPoints[x].latitude;
    var lng = vectorPoints[x].longitude;
    points.push(new js.Proxy(latlng, lat, lng));
}
var lineOptions = js.map(options);
_mapFeature = new js.Proxy(js.context.L.Polyline, points, lineOptions);
js.retain(_mapFeature);
});
}
}

```

Listing 4.13: Konstruktor des Dart-Objekts Polyline

Auf diese Weise erhält jedes Vessel-Objekt im Array ‘Vessels’, der alle auf der Karte dargestellten Schiffe enthält, als Attribut `.feature` und/oder `.polygon` eine Assoziation zu einer Dart-Klasse, die über einen `js.Proxy` die Verwaltung des dazugehörigen Javascript-Objektes im DOM-Baum für die Anwendung kapselt.

Nach demselben Prinzip funktionieren Popups in der Anwendung.

Um Popups auf den MapFeature-Objekten über MouseEvents öffnen und schließen zu können, werden auf den `mouseover` und den `mouseout`-EventListener des Javascript-Proxies eines jeden MapFeature-Objektes EventHandler als Dart-Callback.many-Funktionen registriert (siehe Listing 4.14). Diese Callback.many-Funktionen führen im Dart-Scope alle notwendigen Anweisungen aus. Zum Beispiel erstellt der MouseoverHandler ein Dart-Popup-Objekt, das im Konstruktor einen Proxy zu einem `L.Popup`-Objekt erstellt und als Instanzvariable speichert. Operationen auf dem `L.Map`-Objekt müssen über das Dart LeafletMap-Objekt ausgeführt werden, das den Javascript-Proxy zum `L.Map`-Objekt verwaltet. Im Falle des `onMouseoverHandler`, wird das Dart-Popup-Objekt der Funktion `LMap.openPopup(popup)` übergeben, die Funktion wechselt in den Javascript-Scope und öffnet auf dem `L.Map`-Objekt den Javascript-Proxy des Popup-Objektes (siehe Listing 4.15)

```

void addListeners(){
    onmouseoutHandler(e){
        LMap.closePopup();
    }
    onMouseoverHandler(e){
        var ll = e.latlng;
        ll = new js.Proxy(js.context.L.LatLng, ll.lat, ll.lng);
        var popupOptions = {'closeButton': false,
                           'autoPan': false,
                           'maxWidth': 150,
                           'offset': [50, -50]};
        var popup = new Popup(ll, popupContent, popupOptions);
        LMap.closePopup();
        LMap.openPopup(popup);
    }

    _callbacks.add(new js.Callback.many(onMouseoverHandler));
    _callbacks.add(new js.Callback.many(onMouseoutHandler));
}

```

```

    _mapFeature.on('mouseover', _callbacks[0]);
    _mapFeature.on('mouseout', _callbacks[1]);
}

```

**Listing 4.14:** EventHandling mithilfe von Callback-Funktionen

```

class LeafletMap {
  js.Proxy map;
  ...
  closePopup(){
    js.scoped(){
      map.closePopup();
    };
  }

  openPopup(Popup popup){
    js.scoped(){
      map.openPopup(popup._popup);
    };
  }
  ...
}

```

**Listing 4.15:** Zugriff auf das L.Map-Objekt (im Javascript-Scope) über das LeafletMap-Objekt (im Dart-Scope) in LeafletMap.dart



## 5 Ergebnisse

### 5.1 Evaluation der Anwendung

Alle funktionalen Anforderungen sind im Prototyp der Anwendung in Javascript mit dem node.js-Framework und dem socket.io-Websocket (Abschnitt 4.4.1 und 4.4.2) vollständig umgesetzt. Im Screenshot des Hamburger Hafens (Abb. 5.1) kann man erkennen, dass liegende Schiffe als Kreise dargestellt werden und fahrende Schiffe als Richtungsdreiecke. Wurden Masse übermittelt (AIS-Nachricht vom Typ 4), sind maßstabsgetreue Polygone eingezeichnet, deren Farbe den Schiffstyp kennzeichnet. Ein Popup mit Detailinformationen zum dem Schiff links daneben ist geöffnet. In dieser Zoomstufe ist die Animation bereits aktiv, das heißt für den Betrachter, dass alle angezeigten Richtungsdreiecke ununterbrochen in Bewegung sind.

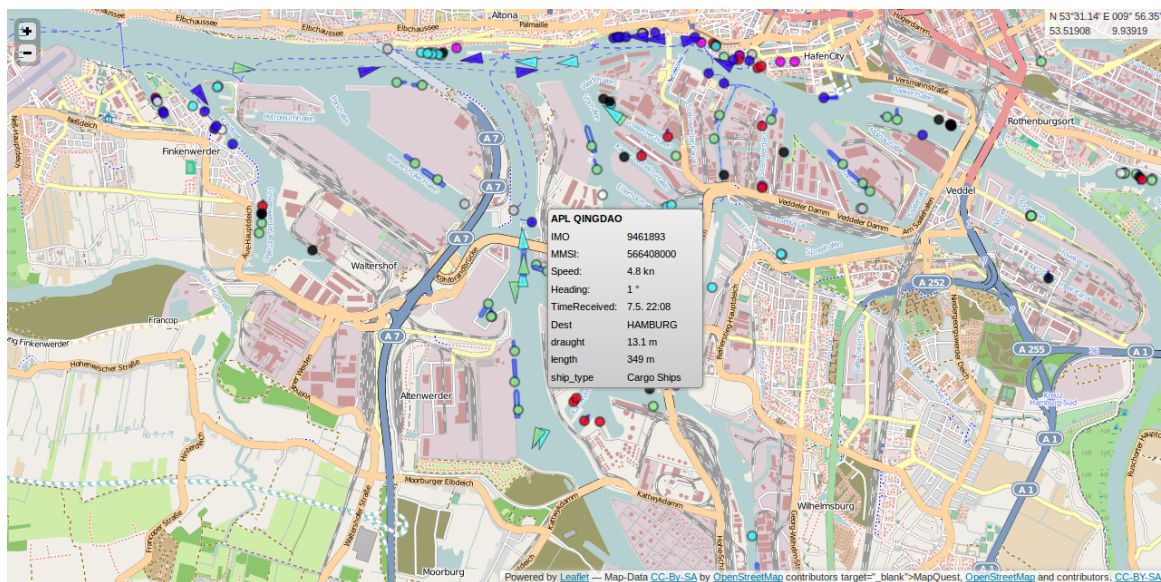
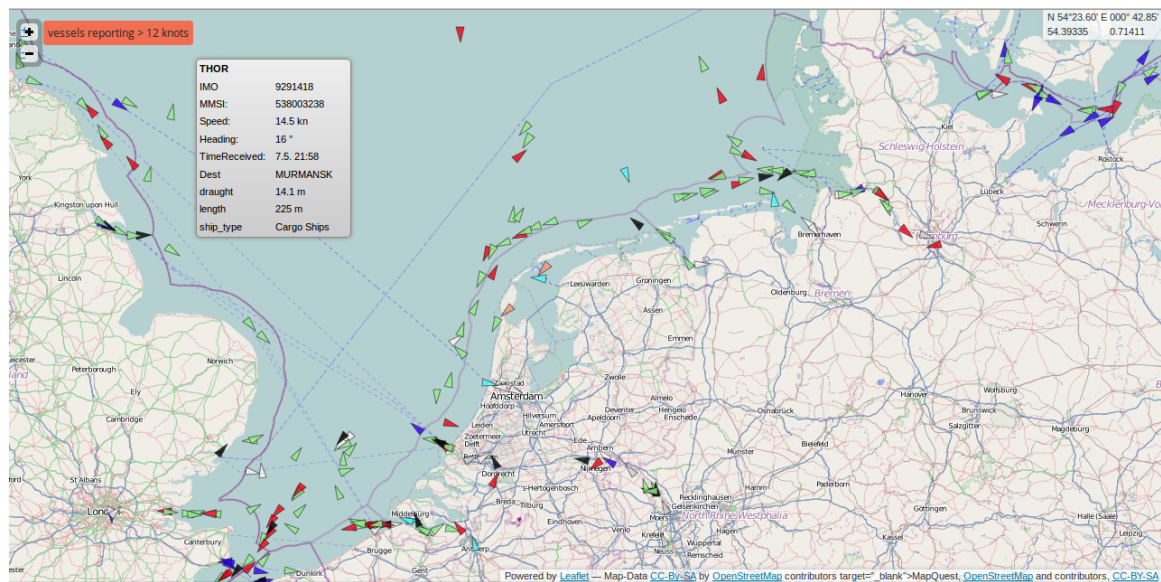


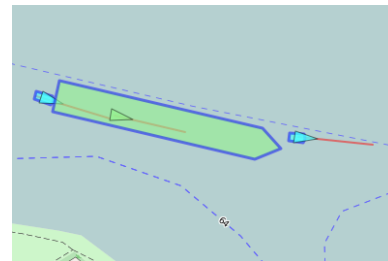
Abbildung 5.1: Anzeige aller Schiffe im Hamburger Hafen

Das zweite Beispiel zeigt die Situation bei niedriger Zoomstufe (Abb. 5.2). Oben links in der Karte ist ein Hinweis eingeblendet, dass nur Schiffe mit einer Geschwindigkeit über 12 Knoten angezeigt werden. Die Verteilung des Schiffsverkehrs lässt sich gut erkennen, während die große Zahl hafenliegender Schiffe ausgespart bleibt, um die Performance des Browser-Clients zu erhalten. Aus demselben Grund ist die Animation ausgeschaltet. Durch die große Anzahl empfangener Positionsmeldungen, ist die Darstellung trotzdem bewegt.



**Abbildung 5.2:** Auf schnell fahrende Schiffe reduzierte Anzeige am Beispiel der Nordsee

Schließlich ist in hohen Zoomstufen eine Beobachtung von aktuellen Schiffsmanövern möglich, wie im Beispiel 5.1 das Schleppen eines Frachtschiffes durch zwei Schleppschiffe. Durch die hohe Frequenz der Positions-Meldungen bei fahrenden Schiffe und mithilfe der eingebauten Animation erlebt der Betrachter die Szene wie in einem Animationsfilm.



**Abbildung 5.3:** Schleppmanöver

Mit dieser Implementierung wurde auch die wichtige nicht funktionale Anforderung nach einer zeitnahen Umsetzung erfüllt. Der Prototyp wird auf github als privates Repository gehostet und konnte nach der Übergabe (Ende Januar) vom Unternehmen vesseltracker für Weiterentwicklungen der Webanwendung und Kundenprojekte verwendet werden. Desweiteren ist die gesamte Anwendung mit open source Produkten entwickelt worden und verwendet das von vom Unternehmen gehostete Kartenmaterial.

Die Latenzzeit zwischen dem Empfang der AIS-Positions-Meldung durch den Rohdatenserver und der Propagierung derselben Position auf der Karte sollte unterhalb 500 msec liegen. Untersucht wurde die Latenzzeit anhand der 'time\_received', die Bestandteil jeder AIS-JSON-Message ist. Dieser Zeitstempel wird vom Rohdatenserver jeder AIS-Message hinzugefügt. Auf dem Rohdatenserver läuft ein ntp-Daemon zur Zeitsynchronisation. Ebenso ein Daemon wurde auf dem Client, auf dem der Browser läuft gestartet und ein Zeitstempel genommen, nachdem das Schiff mit der neuen Position auf die Karte gerendert wurde. Weil die Latenzzeit direkt abhängig

	[msec]
Mittelwert	237
Maximum	701
Minimum	23

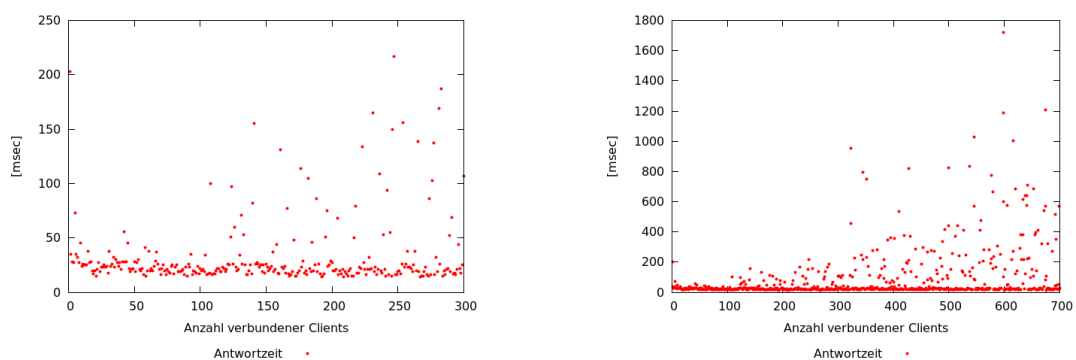
**Tabelle 5.1:** Latenzzeit von Positionsmeldungen



ist von der Anzahl der Meldungen, die pro Zeit vom Client empfangen werden, wurde als Referenz eine Ansicht des Hamburger Hafens gewählt in Zoomstufe 12, der niedrigsten Zoomstufe, in der noch Schiffe jeder Geschwindigkeit angezeigt werden. Die Ergebnisse (Tabelle ??) liegen mit dem Mittelwert gut innerhalb der geforderten Geschwindigkeit.

Die Anzahl der Verbindungen, die der Server gleichzeitig bedienen kann, ist mit einem node.js-Script getestet worden, das alle 500 ms eine neue Clientverbindung erstellt, bis 750 Clients verbunden sind. Der Aufbau der Verbindung geschah auch mit steigender Verbindungsanzahl zuverlässig, jedoch ist in der Abbildung erkennbar, dass die Anzahl der Fälle zunimmt, in denen ein Client lange auf Antwort warten muss.

Eine Skalierung der Serveranwendung ist seitens des socket.io-Servers kein Problem. Statt einen einzigen Worker-Prozess zu generieren, können auch mehrere Worker-Prozesse parallel gestartet werden, die alle dieselbe mongo-Datenbank-Collection abfragen und sich bei derselben redis-Datenbank im Channel 'positionUpdate' registrieren können. Allerdings muss sichergestellt sein, z.B. über unterschiedliche Ports oder unterschiedliche (virtuelle) Server, dass ein verbundener Client mit jedem neuen Request auf demselben Worker-Prozess landet.



**Abbildung 5.4:** Antwortzeiten des socket.io-Websocket-Servers in Abhängigkeit von der Anzahl verbundener Clients

## 5.2 Vergleichende Evaluation der Javascript- und der Dart-Anwendung

Die in Google Dart geschriebene Client-Anwendung soll nun mit dem in Javascript programmierten Client verglichen werden. Die in den Abschnitten 4.2 und 4.3 begründete zusätzliche Implementierung eines HTML5-kompatiblen node.js-Websocket-Servers muss nun in einem ersten Schritt mit dem node.js-socket.io-Server verglichen werden. Für diesen und alle folgenden Tests wurde auf dem Rohdatenserver ein Port eingerichtet, der nur die Daten von drei AIS-Antennen (Hamburg, Wedel, Geesthacht) ausgibt. Dies war notwendig, um den als Server verwendeten Arbeitsplatzrechner (MHz und 1 GB Arbeitsspeicher) nicht zu überlasten. Dadurch entstünde eine zusätzliche Latenzzeit wegen des kontinuierlichen Anwachsens der Messagequeues auf dem Server, die die Ergebnisse verzerrt.

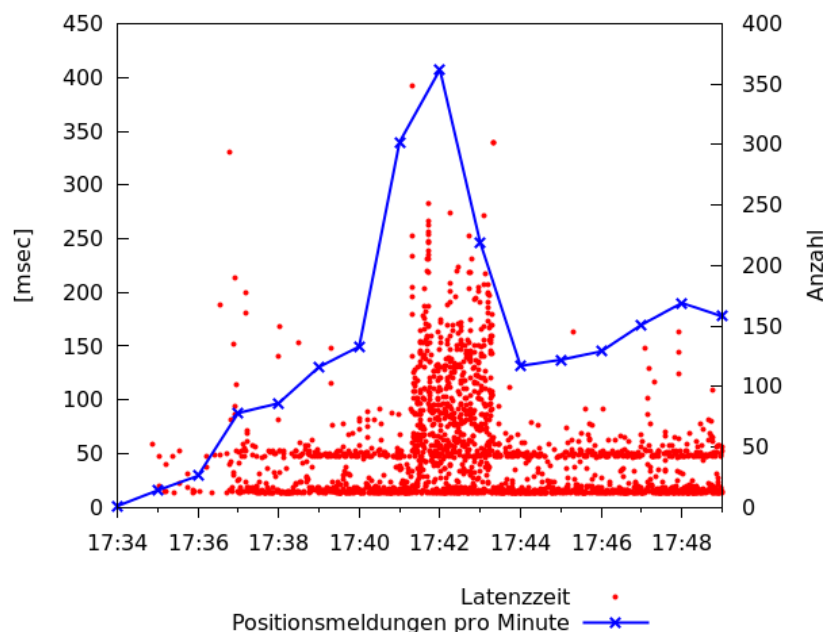
### 5.2.1 Node.js - socket.io-Socket-Server vs. node.js-Websocket-Server

#### Implementierungsaufwand

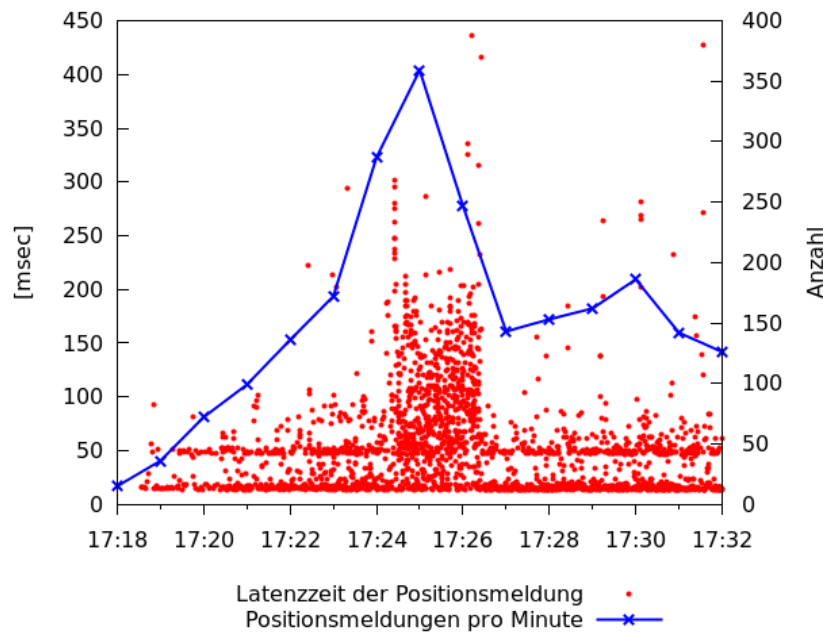
Im Implementierungsaufwand unterscheiden sich beide Server- und Client-Anwendungen kaum (Anzahl zeilen code). Einige Features der socket.io-Bibliotheken (z.B. die Clientverwaltung, Parameter für 'Production' und 'Development'-Umgebung bezüglich Logleveln, Client-minification oder Client-Zip) sind praktisch und müssten in der Alternativimplementierung für den Einsatz in einer produktiven Umgebung anderweitig gelöst werden. Durch die in socket.io eingeführten Events vermindert sich der Kommunikationsaufwand zwischen Server und Client geringfügig, was in diesem Fall einer datenintensiven Anwendung mit großen Datenmengen pro Nachricht wenig zu Buche schlägt.

#### 5.2.2 Latenzzeit

Die Leistungsfähigkeit beider Implementierungen wird verglichen, indem wieder wie in Tabelle ?? die Zeit gemessen wird, die eine Positionsmeldung braucht für den Weg vom Rohdatenserver bis zur Präsentation auf der Karte. Um eine ähnliche Situation in beiden Szenarien abzubilden, wurde jeweils eine bestimmte Position in Hafen Hamburg angesteuert und ein Timer in die Client-Anwendung integriert, der jeweils nach einer Minute um eine Stufe herauszoomt. Da ab Zoomlevel 11 und kleiner nur noch Schiffe mit einer jeweils definierten Mindestgeschwindigkeit angezeigt werden, nahm die Anzahl empfangener Schiffe von dieser Zoomstufe an wieder ab. Zur Auswertung wird auf dem Client ein LogFile geschrieben, das pro Nachricht, deren 'time\_received' und einen aktuellen Zeitstempel schreibt. Die Differenz wird als Latenzzeit interpretiert. Anschließend wird über das Logfile berechnet, wieviele Positionsmeldungen in einer Minute an den Client gesendet wurden. Darüber ist es möglich, die Latenzzeit gegen die Anzahl empfangener Positionsmeldungen pro Minute darzustellen, wie in Abbildung 5.5 und 5.6 zu sehen.



**Abbildung 5.5:** socket.io-Websocket-Server: Latenzzeit der Positionsmeldungen und Anzahl empfangener Schiffe



**Abbildung 5.6:** HTML5-Websocket-Server: Latenzzeit der Positionsmeldungen und Anzahl empfangener Schiffe

Es ist offensichtlich, dass die Geschwindigkeit in der Darstellung von der Anzahl empfangener Nachrichten linear abhängt. Darüber hinaus ist zu erkennen, dass beide Implementierungen ihre Aufgabe in ähnlicher Geschwindigkeit erledigen.

### 5.2.3 Browserunterstützung

Firefox, Chrome, IE, Safari

## 5.3 Javascript-Client vs. Dart-Client

### 5.3.1 Implementierungsaufwand

In der Dart-Implementierung verlangt das Arbeiten mit zwei unterschiedlichen Scopes (javascript und Dart) dem Programmierer einiges ab und ist am Anfang sehr fehleranfällig. Die Fehler sind schwierig zu debuggen, weil der Debugger (javascript-Debugger Firebug oder der Debugger im DartEditor) ebenfalls nicht über die Grenzen des Namensraumes wechseln können.

### 5.3.2 Performance

Um die unterschiedliche Performance des Javascript- und des Dart-Clients zu vergleichen, wurde der VesselInBounds-Event genutzt. Gemessen wurde die Zeit, die benötigt wird, um nach Empfang eines VesselInBounds-Events alle in der Message enthaltenen Schiffe auf die Karte zu rendern. Dazu wurden die Browser Dartium, Chrome und Firefox auf dem Client benutzt.

- in Google Dartium wird bei Aufruf des Dart-Clients der originäre Dart-Code interpretiert. Beim Aufruf des Javascript-Clients wird der Javascript-Code interpretiert.
- in Google Chrome wird bei Aufruf des Dart-Clients der mit dart.js zu javascript kompilierte Dart-Code interpretiert und beim Aufruf des Javascript-Clients der Javascript-Code mit der V8-Engine.
- in Firefox

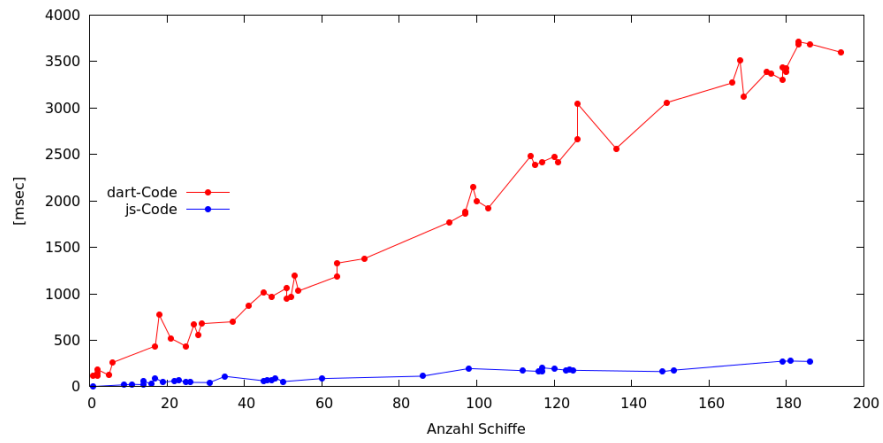


Abbildung 5.7: Dauer des Renders in Dartium

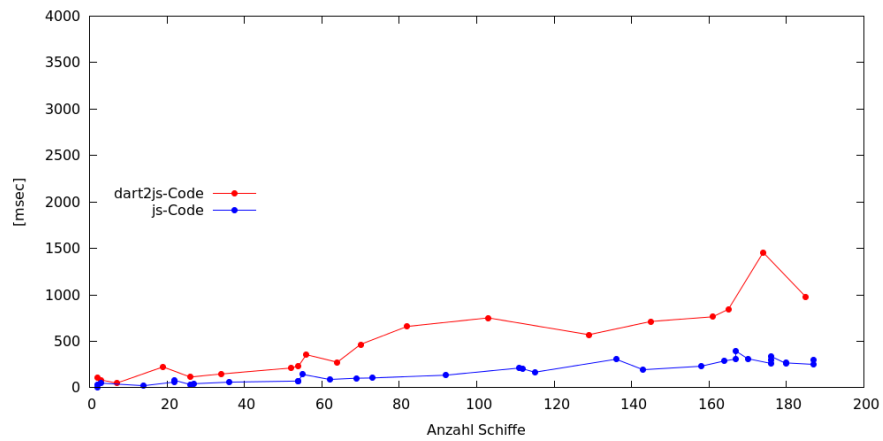


Abbildung 5.8: Dauer des Renders in Chrome

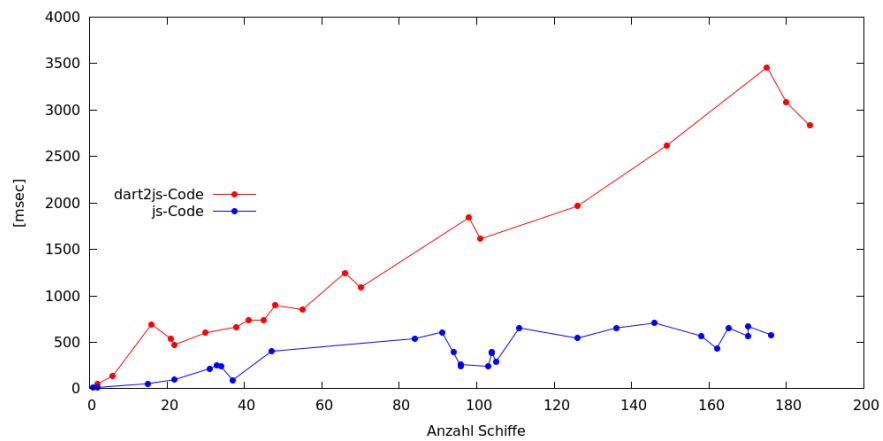


Abbildung 5.9: Dauer des Renders in Firefox

### 5.3.3 Browserunterstützung

#### Dartium

##### Firefox, Chrome, IE, Safari

Der dart-Client kompiliert den in Dart geschriebenen Code zu Javascript.

Dabei traten Fehler auf, die unter Dartium (also im originalen Dart-Code) nicht auftraten. 1. Wird innerhalb des Javascript-Scopes eine Methode auf einen javascript-Proxy (hier `_map`) aufgerufen und ein proxy wird zurückgegeben, dann ist es nicht möglich auf diesen Proxy, der in diesem Fall vom Typ `LatLngBounds` sein müsste, eine Methode der Klasse `LatLngBounds` aufzurufen. => `TypeError: t1.get$_map(...).getBounds$0(...).getSouthWest$0 is not a function`

dart-client: `web/leaflet_maps.dart`

```
List getBounds() var south, west, north, east; js.scoped(() south= _map.getBounds().getSouthWest().lng;
west = _map.getBounds().getSouthWest().lat; north = _map.getBounds().getNorthEast().lng;
east = _map.getBounds().getNorthEast().lat; ); return [west, south, east, north];
```

In diesem Fall wird einfach als work-Around eine andere Methode verwendet (`getBBox-String`), die einen String mit den Bounds zurückgibt. Aus den Teilen dieses Strings werden mit der Methode `parse(string)` der Klasse `double` die Werte der Eckpunkte der Bounds generiert.

```
String getBounds() String bBox; js.scoped(() bBox = _map.getBounds().toBBoxString(); );
return bBox;
```

Weil dadurch der message-Parameter 'bounds' kein number-Array, sondern ein String ist, muss im html5-Server der String einmal zum Float geparkt werden.

2. Ein Feld (IMO") wird auf null und auf > 0 geprüft.

## **6 Fazit**

### **6.1 Ergebnisse**

#### **6.1.1 Die Realtime-Applikation**

Features, die in die Anwendung integriert werden sollten:

- eine aufklappbare Liste, in die der Nutzer favorisierte Häfen speichern kann, die dann mit einem Klick angesteuert werden können.
- eine Legende mit einer Erläuterung der farbigen Schiffssymbole
- zur weiteren Optimierung der Anwendung, sollte die Möglichkeit, die Animation der Richtungsdreiecke und Schiffspolygon über css3-Transition-Funktionen zu realisieren, unbedingt weiterverfolgt werden

#### **6.1.2 Vergleich Javascript und Google Dart**

Wie in den Abbildung zur Struktur der Anwendung erkennbar, ist mit Dart sehr viel einfacher Objektorientiertheit herzustellen. Dadurch wird die Anwendung einfacher zu verstehen und damit auch einfacher zu entwerfen, zu schreiben und zu warten.

### **6.2 Ausblick**

Legende -Satellitendaten in die Anwendung einbinden

## Literaturverzeichnis

- [Lad12] Ladd, Seth: *Dart: Build HTML5 Apps Fast*. <http://www.drdobbs.com/open-source/dart-build-html5-apps-fast/240005631>. Version: September 25 2012
- [Men12] Menon, Vijay: *Using JavaScript from Dart: The js Library*. <http://www.dartlang.org/articles/js-dart-interop/>. Version: September 2012
- [MU12] Malte Ubl, Eiji K.: *Introducing WebSockets: Bringing Sockets to the Web*. <http://www.html5rocks.com/en/tutorials/websockets/basics/>. Version: March 13 2012. – [Online; Stand 13. März 2012]
- [Mü12] Müller, Frank: *Ein Jahr JavaScript-Konkurrent Dart*. <http://www.heise.de/developer/artikel/Ein-Jahr-JavaScript-Konkurrent-Dart-1727116.html>. Version: October 12 2012
- [Tei12] Teixeira, Pedro: *Professional Node.js*. Wrox, 2012. – 408 S.
- [Var11] Varaksin, Oleg: *Kommunikationstechnologien für webbasierte Client-Server-Anwendungen*, Fernuniversität Hagen, Diplomarbeit, 2011
- [Wik13] Wikipedia: *Automatic Identification System*. [http://de.wikipedia.org/wiki/Automatic\\_Identification\\_System](http://de.wikipedia.org/wiki/Automatic_Identification_System). Version: 2013. – [Online; Stand 13. Mai 2013]



## Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Ort, Datum

Unterschrift