Hochschule Bremerhaven

Fachbereich II
Management und Informationssysteme
Informatik B.Sc.

Modul
Infrastruktur

Semesterprojekt von Team B

Eine webbasierte Anwendung zur Darstellung von

Live-AIS-Nachrichten

Vorgelegt von:	Bastian Buch	MatNr. 34480
	Luca Focken	MatNr. 42418
	Leon Stüve	MatNr. 41996
	Lukas Weber	MatNr. 41894
Vorgelegt am:	30. August 2025	
Dozent:in:	Prof. Dr.-Ing. Oliver Radfelder	

Inhaltsverzeichnis

1 Einleitung . . . 7

2 Ablaufumgebung . . . 7

3 Technologien . . . 9

4 Architektur . . . 12
4.1 Session . . . 12
4.2 E-Mails . . . 13
4.3 Dynamischer Inhalt . . . 16
4.4 Daten . . . 17
4.5 Frontend . . . 18
4.5.1 Akkordeon . . . 18
4.5.2 Sessions . . . 19
4.5.3 Leaflet-Karte . . . 20
4.5.4 Tabelle . . . 22
4.6 Deployment . . . 22
4.6.1 Datenbank . . . 23
4.6.2 Webseite . . . 24
4.6.3 CGI-Skripte . . . 24
4.6.4 Queue . . . 24
4.7 Testing und Experimente . . . 25
4.7.1 Unit Tests . . . 25
4.7.2 rhodes2 . . . 25
4.7.3 Wie verhält sich die Datenbank bei übermäßig vielen Daten? . . . 26
4.7.4 Wie verhält sich die Karte bei übermäßig vielen Daten? . . . 27
4.7.5 Wie verhält sich die Queue bei übermäßig vielen Logins? . . . 28

5 User-Durchläufe . . . 29
5.1 Login . . . 29
5.1.1 Aufruf der Seite . . . 29
5.1.2 Initialisierung nach Laden der Seite . . . 29
5.1.3 Aktueller Zustand . . . 29
5.1.4 Benutzeraktion: Login . . . 30
5.1.5 Backend . . . 30
5.1.6 Kein Match bei dem Abgleich mit der Datenbank . . . 30
5.1.7 E-Mail-Simulation . . . 30
5.1.8 Frontend-Anpassungen . . . 31
5.2 Logout . . . 31
5.2.1 Benutzeraktion: Logout . . . 31

5.2.2 Backend-Verarbeitung . . . 31
5.2.3 Frontend-UI-Anpassungen . . . 32
5.3 Karte . . . 32
5.3.1 Öffnen der Karte . . . 32
5.3.2 Initialisierung der Leaflet-Karte . . . 32
5.3.3 Laden der Karteninfos . . . 32
5.3.4 Periodische Updates . . . 33
5.3.5 Benutzeraktion: Marker-Popup öffnen . . . 33
5.4 Tabelle . . . 33
5.4.1 Benutzeraktion: Tabelle öffnen . . . 33
5.4.2 Backend-Verarbeitung . . . 33
5.4.3 Darstellung im Frontend . . . 34
5.4.4 Synchronisation mit Marker-Popups . . . 34
5.4.5 Dauerbetrieb . . . 34

6 Anwendung aus Nutzungssicht . . . 34
6.1 Login . . . 34
6.2 Karte . . . 35
6.3 Tabelle . . . 35

7 Versionsgeschichte . . . 36

8 Beobachtungen . . . 37
8.1 Messungen . . . 37
8.1.1 Skript-Laufzeiten . . . 37
8.1.2 STEP-Infra-Vergleich . . . 39
8.1.3 Laufzeiten zu Useranzahlen . . . 40
8.1.4 Systemauslastung zu Useranzahlen . . . 41
8.2 Monitoring . . . 41
8.3 Mögliche zukünftige Änderungen . . . 42
8.3.1 Daten-Format . . . 42
8.3.2 Encryption . . . 42

9 Quellen . . . 44
9.1 General . . . 44
9.2 Leaflet . . . 44
9.3 Javascript . . . 44
9.4 MariaDB . . . 45
9.5 Gnuplot . . . 45

10 Selbstreflexion . . . 46
10.1 Bastian Buch . . . 46

10.2 Luca Focken . . . 48
10.3 Leon Stüve . . . 49
10.4 Lukas Weber . . . 51

Literaturverzeichnis . . . **54**

Abbildungsverzeichnis . . . **57**

Tabellenverzeichnis . . . **58**

Listingverzeichnis . . . **59**

Abkürzungsverzeichnis . . . **60**

Anhang . . . **61**

I CGI . . . **62**
I infra-get-positions.sh . . . 62
II infra-get-table.sh . . . 62
III infra-update-positions.sh . . . 63

II Data . . . **64**
I allow-mariadb-access.sh . . . 64
II cleaner.sh . . . 65
III init.sql . . . 65
IV watcher.sh . . . 66
V worker.sh . . . 66

III Deploy . . . **68**
I deploy-hopper.sh . . . 68
II deploy-local.sh . . . 68
III deploy-prod.sh . . . 71

IV Session . . . **72**
I infra-login.sh . . . 72
II infra-logout.sh . . . 72
III infra-web-lib.sh . . . 73

V Tests . . . **75**
I all-scripts . . . 75
I.1 do-all.sh . . . 75
I.2 graph.gpi . . . 76
I.3 test-scripts.sh . . . 77

II database-test . . . 79
II.1 main.gp . . . 79
II.2 main.sh . . . 79
II.3 measure.sh . . . 80
III monitoring . . . 81
III.1 app.js . . . 81
III.2 build-stats-now.sh . . . 82
III.3 get-stats-now.sh . . . 82
III.4 index.html . . . 83
III.5 main.css . . . 84
III.6 stats-now.gp . . . 84
III.7 watcher.sh . . . 85
IV queue-test . . . 86
IV.1 main.gpi . . . 86
IV.2 queue-test-lib.sh . . . 87
IV.3 restart-queue.sh . . . 90
IV.4 test-queue.sh . . . 91
IV.5 users.sh . . . 91
V rhodes2.sh . . . 91
VI simulate-user . . . 92
VI.1 do-all.sh . . . 92
VI.2 multi-user.sh . . . 93
VI.3 single-user.sh . . . 93
VI.4 png.gpi . . . 95
VII step-vs-infra . . . 96
VII.1 do-all.sh . . . 96
VII.2 main.sh . . . 98
VII.3 step-get-positions.sh . . . 98
VII.4 png.gpi . . . 100
VIII stress-test . . . 100
VIII.1 do-all.sh . . . 100
VIII.2 get-n-user-avg-cpu-usage.sh . . . 101
VIII.3 main.gp . . . 102
IX unit-tests.sh . . . 103

VI Utils . . . 104
I do-all.sh . . . 104
II get-data.sh . . . 105
III make-png.gpi . . . 106
IV show-all-project-related.sh . . . 108
V show-commits-fitting-keyword.sh . . . 108

VII Queue **108**
I infra-queue-lib.sh 108
II step-queue-lib.sh 110
III restore.sh 113
IV watcher.sh 114
V worker.sh 114

VIII WWW **115**
I app.js 115
II index.html 125
III main.css 126

Selbstständigkeitserklärung **131**

1 Einleitung

In dem Modul Infrastruktur haben wir die Aufgabe erhalten, in Teams von drei oder vier Personen ein Projekt gemeinsam auszuarbeiten und in die Realität umzusetzen. Dieses Projekt sollte eine Vielzahl von über den Verlauf des Semesters erlernten Technologien, Strukturen und Techniken vereinen.

Die Basis sollte der Server Rhodes darstellen, welcher Automatic Identification System (AIS)-Daten aussendet. Die Aufgabe bestand nun darin, eine Webanwendung bereitzustellen, welche das Einsehen der aktuellsten Daten dynamisch und in Echtzeit in einer Leaflet-Karte sowie einer Tabelle erlaubt. Darüber hinaus musste eine Login-Funktion eingebaut sein, welche den Zugriff überhaupt erst ermöglicht.

Das Hauptziel stellte dabei das Erlernen der effizienten Arbeit innerhalb eines Teams dar - also das Planen, das Umsetzen und das Verstehen von allem Eingesetzten. Darüber hinaus wurde besonderer Fokus auf das Ineinandergreifen der einzelnen Komponenten und insbesondere in dem Web als Umgebung gesetzt.

Das Ergebnis dieser Arbeit findet sich unter:
https://informatik.hs-bremerhaven.de/docker-infra-2025-b-web/infra-2025/.
Die E-Mail lautet “daddeldi“ und das Passwort “daddeldu“.

2 Ablaufumgebung

Der Docker oder die Virtuelle Maschine (VM) auf unseren lokalen Maschinen stellt den Mittelpunkt unserer Arbeitsumgebung dar und befindet sich in Abbildung 2.1 dementsprechend auch im Zentrum. In diesen Machinen programmieren wir.

Im Docker findet sich die Verzeichnisstruktur unseres Projekts, der Apache2 als Webserver sowie die Common Gateway Interface (CGI)-Skripte, die dieser ausführt.

Die Verzeichnisstruktur stellt die Basis unserer Teamarbeit dar und wurde dementsprechend als einer der ersten Aspekte von uns ausgearbeitet. Jeder Teil des Projekts hat ein Verzeichnis zugewiesen bekommen:

Die Skripte für das Deployment der Anwendung in die VM, den Docker des Individuums oder den geteilten befinden sich im deploy-Verzeichnis.

Die schriftliche Ausarbeitung sowie zur Planung zugehörige Dateien und Abbildungen befinden sich im latex-Verzeichnis.

Die verschiedenen Tests, sowohl Unit Tests als auch solche, die die Anwendung als Gesamtprodukt untersuchen, befinden sich im tests-Verzeichnis.

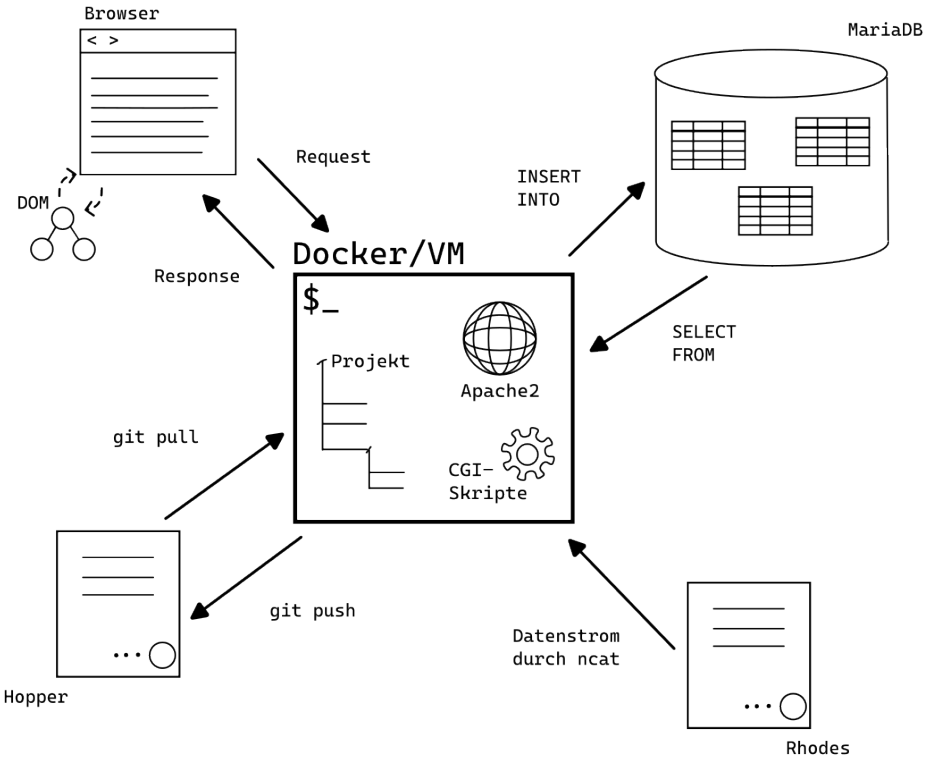


Abbildung 2.1: Darstellung der Ablaufumgebung

Hilfsskripte für die Bearbeitung befinden sich im utils-Verzeichnis.

Das toplevel src-Verzeichnis ist seinerseits in verschiedene Unterverzeichnisse eingeteilt, welche den Quellcode der verschiedenen Einheiten der Anwendung beinhalten. In ihm befinden sich die Verzeichnisse: cgi, data, queue, session und www.

Die deployte Anwendung wird auf Anfrage des Benutzenden durch den Apache2-Webserver ausgeliefert, welcher auf dem Docker als Dienst läuft. Dargestellt sind die Anfragen in der Grafik durch Pfeile zwischen Browser und Docker.

Die im cgi-bin-Verzeichnis ansässigen Skripte generieren Inhalte, die auf Anfrage vom Apache2-Server ausgeliefert werden. Sie stellen unsere Kommunikation mit dem Backend aus dem

Frontend dar. Die restlichen Skripte arbeiten im Hintergrund als Teil des Backends. Sie pflegen Daten in die Datenbank ein, verwalten die loszusendenden E-Mails oder Überwachen den Datenstrom zu Rhodes.

Jener ist auf der Abbildung unten rechts dargestellt. Rhodes ist der Server, von dem wir die zu verwaltenden AIS-Daten empfangen.

Die Datenbank, die das Ziel dieser Nachrichten darstellt, werden in dem Datenbankmanagementsystem (DBMS) MariaDB verwaltet, welches in der Hochschulumgebung auf dem Server Hilbert läuft - oder lokal in unseren VMs. Abgebildet ist dies in der Grafik oben rechts mitsamt Insertion und Selection.

Über den Browser, oben links in der Grafik, verwenden Nutzende die Anwendung. Die Daten werden dynamisch in einem Akkordeon-User Interface (UI) dargestellt. Dieses ist aufgeteilt in drei Segmente: Login/Logout, eine Karte und eine Tabellenansicht. In der Grafik unter dem Browser befindet sich das Document Object Model (DOM), welches als Schnittstelle agiert, welche die dynamische Manipulation des Dokuments durch JavaScript erlaubt.

Der Server Hopper ist unser zentrales Speichermedium und hosted unser Git-Repository. Er ermöglicht die Teamarbeit in der Hochschulinfrastruktur und befindet sich in der unteren linken Ecke der Darstellung.

3 Technologien

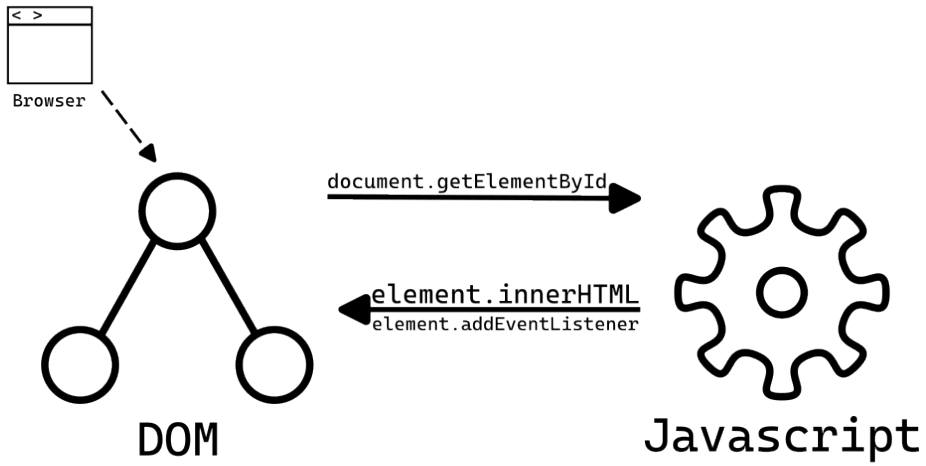


Abbildung 3.1: Javascript-DOM-Schnittstelle

Die verwendeten Technologien sind solche, mit welchen wir bereits aus dem ersten Semester vertraut waren und diese, die wir dieses Semester neu kennengelernt haben.

Darunter haben wir JavaScript (JS) als Skripting-Sprache, welche über das DOM das dynamische Verändern der angezeigten Inhalte im Web ermöglicht. Das DOM liefert die Objektrepräsentation des Hypertext Markup Language (HTML) in Form einer Baumstruktur.

Methoden wie `document.getElementById` erlauben uns den Zugriff auf Elemente im Dokument in JS und damit die dynamische Änderung des angezeigten Inhalts. Ebenso lassen sich Event-Listener registrieren, die beim Auslösen bestimmter Events, in Form von Callbacks, darauf reagieren und den hinter der Funktion liegenden Code ausführen.

In der Grafik 3.1 ist das Zusammenspiel von JS und DOM dargestellt durch die zwei Pfeile.

Ebenfalls verwenden wir hier Leaflet - eine externe Bibliothek, welche uns die dynamische Kartenansicht ermöglicht. Wir initialisieren eine Karte mitsamt zugehörigem Tileset, in der in Form von Polylines die Positionsmeldungen der Schiffe der letzen Stunde dargestellt sind. Die aktuellen Positionen werden dabei durch Marker angezeigt.

Die Elemente wie Polylines und Marker werden dann zu der vorher etablierten Karte hinzugefügt und dementsprechend von Leaflet in der Karte dargestellt.

Dem nachfolgenden Klassendiagramm, Abbildung 3.2, kann man die von uns verwendeten Klassen und Methoden entnehmen. Leaflet gebraucht also einen objektorientieren Aufbau mit Vererbung. Die unterliegende Polymorphie-Struktur kann anhand der Pfeilrichtungen ausgemacht werden.

Polylines bestehen aus mehreren LatLngs, was es uns ermöglicht, die verschiedenen Schiffspositionen abzubilden. Eine Map weist ebenso eine arbiträre Anzahl an Layern auf - es lassen sich bei Belieben neue Layer einfügen.

Zu dem Einfügen und Abfragen der Daten in unserer relationalen Datenbank nutzen wir die Query-Sprache Structured Query Language (SQL). Diese verwenden wir mit MariaDB als Verwaltungssoftware der Datenbanken. MariaDB arbeitet nach dem Server-Client-Prinzip. Heißt: Der vom Apache2 verwendete User www-data greift als Client abfragend auf den Server zu.

Apache2 ist dabei unser Webserver. Das cgid-Modul ermöglicht uns das Verwenden von CGI-Skripten. Somit können wir dynamisch Inhalte im Backend generieren und an das Frontend senden.

CGI bezeichnet die Möglichkeit, Prozesse aufzurufen und dessen Ausgabe in den Standardkanal als Antwort an den Anfrage stellenden Client über Hypertext Transfer Protocol (HTTP) zu senden.

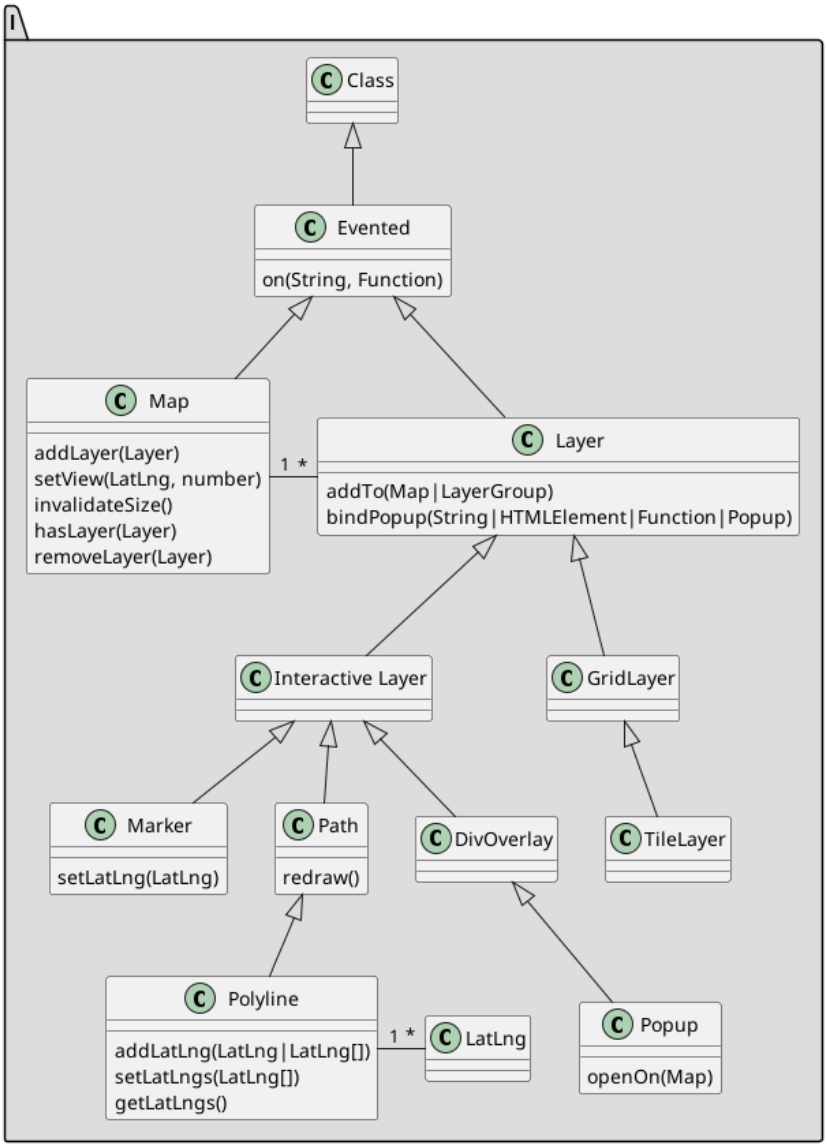


Abbildung 3.2: Leaflet-Klassendiagramm

Über das HTTP-Protokoll senden Clients Anfragen an den Webserver. Im Header sind einige Informationen über die Anfrage aufgelistet, welche der Server verwendet, um die passende Antwort durch das selbe Protokoll zurückzusenden.

Die Aufgabe des Protokolls ist die Formalisierung und damit Vereinbarung der Kommunikationsart und -weise - in diesem Fall zwischen Client und Server.

Unsere Komponenten laufen dabei alle auf Linux-Systemen. Die verwendeten Docker laufen auf Debian und unsere virtuellen Maschinen auf Ubuntu, welches ein Debian-Derivat ist. Die verwendeten Long Term Support (LTS)-Versionen ermöglichen Stabilität und die Philosophie Linux'

der Betrachtung aller Daten und Geräte als Textdateien erlaubt die einfache Konfiguration.

4 Architektur

Unsere Architektur ist aufgeteilt in das Frontend, welches Asynchronous JavaScript and XML (AJAX)-Requests lossendet, um damit dynamisch anpassbar zu sein, dementsprechend CGI-Skripte im Backend, welche die abgefragten Daten bereitstellen, einen Anteil, der dafür zuständig ist, die Daten von Rhodes zu empfangen und passend in unsere Datenbanken einzupflegen und letztlich einen Anteil, welcher das Senden der E-Mails vornimmt (hier: simuliert).

4.1 Session

Unser Sessionhandling geschieht in Form von CGI-Skripten: `infra-login.sh` und `infra-logout.sh`.

`infra-login.sh` akzeptiert die Anmeldedaten über den Query String und erstellt genau dann die Session in dem Dateisystem und antwortet mit dem dazugehörigen Cookie, wenn sie korrekt sind.

`infra-logout.sh` löscht die aktuelle Session in dem Dateisystem und entfernt den Cookie.

`infra-web-lib.sh` ist unsere Session-Handling-Bibliothek. Sie enthält die Funktionen, die `infra-logout.sh` und `infra-login.sh` aufrufen.

`createsession` generiert mittels `pwgen` einen zufälligen 40 Zeichen langen Cookie. Innerhalb des Dateisystems, in `/var/www/data`, wird zu diesem dann ein passendes Verzeichnis erstellt, in welchem die E-Mail gespeichert wird. Wir haben uns für dieses Verzeichnis entschieden, da es von `www-data` eingelesen, aber niemals ausgeliefert werden kann.

`deletesession` löscht das zu dem Cookie gehörende Verzeichnis, insofern er gesetzt ist und das Verzeichnis existiert.

`getsession` holt den Cookie-Value aus dem HTTP-Header und setzt, wenn ein dazu passendes Session-Verzeichnis existiert, eine globale Variable `ses_email`, die die passende E-Mail enthält.

`logaction` schreibt eine Zeile in unser Log, welche die E-Mail, den Cookie, den Timestamp und die Aktion selbst (den Skriptnamen), enthält.

`parsequerystring` parsed den Query String abschnittsweise. Als Argumente mitgegebene Variablenbezeichnungen werden mit dem Präfix `get_` auf den passenden Wert gesetzt, insofern sie in dem Query String enthalten sind.

`checkuser` prüft, ob eine Login mit der angegebenen E-Mail-Passwort-Kombination möglich ist.

4.2 E-Mails

Eine Anforderung an die Anwendung war, dass bei dem erfolgreichen Einloggen das Senden einer E-Mail simuliert wird. Dabei verwenden wir die Queue als First In, First Out (FIFO)-Datenstruktur, die uns das gleichmäßige Abarbeiten über einen längeren Zeitraum hinweg ermöglicht, damit bei vielen Anfragen keine Spikes in der Auslastung verursacht werden. Dazu wird diese Verarbeitung der asynchron einkommenden Aufträge durch die Queue serialisiert.

Die Queue haben wir als verkettete Liste innerhalb des Dateisystems, mit einem Head und einem Tail, persistent in `/data/queue` umgesetzt. Der Head und Tail fungieren als Zeiger, die auf das jeweils erste und letzte Element in der Queue verweisen. Jede Node erhält den Namen dieser, die auf sie folgt, oder einen Bindestrich als Platzhalter, wenn sie die letzte ist.

Die Implementierung der Queue haben wir in Form von einer Bash-Bibliothek vorgenommen, die von dem abarbeitendem und hinzufügenden Prozess gesourced werden kann.

Nach einem Login führt www-data im Skript `infra-login.sh` die enqueue-Funktion aus und fügt so eine Node in die Queue ein. Parallel dazu läuft ein Worker, welcher vom User `workuser` ausgeführt wird und alle paar Millisekunden versucht, die Nodes in der Queue abzuarbeiten.

Das Laufen des Workers wird durch einen Watcher gewährleistet. Dieser schaut über `ps`, ob der bekannte Process Identifikator (PID) des letzten laufenden Workers weiterhin existiert und zu diesem gehört. Ist das nicht der Fall, wird ein neuer Worker mit `nohup` gestartet und dessen PID gespeichert.

Um Anomalien bei dem asynchronen Enqueuen und Dequeuen zu verhindern, implementieren wir neben einer enqueue- und einer dequeue-Funktion noch lock- und unlock-Funktionen, die ein Spinlock im Dateisystem realisieren.

Die lock-Funktion macht Gebrauch von dem atomaren Kommando `mkdir`. Schlägt das Erstellen des Locks fehl, existiert dieser also bereits, wartet die Funktion 100ms zwischen weiteren Versuchen. Dies verhindert parallelen Zugriff auf den kritischen Bereich. Für das Wiederholen der Queue nach dem Neustart speichern wir eine Datei im Lock, dessen Namen der blockierenden Funktion entspricht.

Die unlock-Funktion löscht das bestehende Lock und gibt damit die gemeinsam genutzte Ressource frei.

Die Abbildung 4.1 zeigt den Aufbau unserer Queue und wie sich diese in Bezug auf Enqueue und Dequeue verhält.

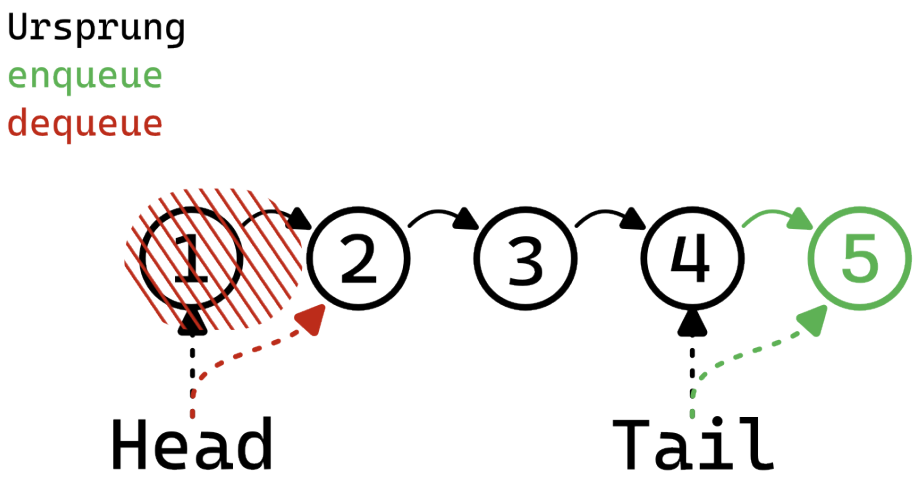


Abbildung 4.1: Queue

Zu Beginn der enqueue-Funktion wird das Lock angefordert und der aktuelle Zeitpunkt in der Variable `timestamp` gespeichert. Dann überprüfen wir, ob alle Argumente mit angegeben wurden. Falls nicht, entfernen wir das Lock und verlassen die Funktion mit `exit 1`.

Andernfalls übergeben wir einer neuen Node, hier Node 5, namens `node-$timestamp` den Timestamp, einen Bindestrich und die drei Argumente jeweils getrennt durch ein Pipezeichen und setzen Schreibrechte für die Gruppe. Im Anschluss lesen wir die Node aus Tail aus - hier Node 4. Wir aktualisieren die Referenz aus Node 4, sodass sie auf Node 5 zeigt. Dann schreiben wir Node 5 in Tail, damit Tail wieder auf die neuste Node zeigt.

Falls die Queue leer ist, also falls Node 5 die erste Node ist, setzen wir nur Tail und zudem Head auf Node 5. In beiden Fällen wird das Lock entfernt.

Zu dem Start der dequeue-Funktion wird ebenfalls das Lock angefordert. Wir holen uns die Referenz aus Head, den Inhalt dieser Node, hier Node 1, um die E-Mail-Abarbeitung zu simulieren und die Referenzen in der Queue zu aktualisieren.

Enthält Head keine Referenz, so ist auch die Queue leer. Dann entfernen wir das Lock und warten mit `sleep` ein paar Sekunden, um weniger Ressourcen zu verbrauchen.

Es kann vorkommen, dass das System in der Bearbeitung eines kritschen Abschnitts herunterfährt. In solchen Fällen, kann es also dazu kommen, dass das Lock nie entfernt wird und die Queue in einem unbearbeitbaren Zustand, ein Deadlock, gelangt. Um dem entgegenzuwirken haben wir ein Skript `restore.sh` geschrieben, welches wir nach dem Start des Systems einmalig ausführen.

```
#!/usr/bin/env bash

cd /data/queue
! test -d 'lock' && exit 0

test -e 'lock/enqueue' && sudo -i -u www-data enqueue
test -e 'lock/dequeue' && sudo -i -u workuser dequeue
rm -rf 'lock'
```

Listing 4.1: restore.sh

Durch die in `/data/queue/lock` angelegte Datei können wir zuordnen, welchen Bereich es wiederherzustellen gilt. Um die Dateirechte zu wahren, führen wir die Funktionen jeweils als dessen primärer Nutzer aus.

```
enqueue() {
  # Read tail ref
  read -r tail_ref < 'tail'
  newest_node="$(echo node-* | tr ' ' '\n' | sort -rn | head -1)"
  test "$tail_ref" = "$newest_node" && return 0

  # Recover if wrong ref
  chmod g+w "$newest_node"
  if test "$tail_ref" != '-'; then
    IFS='|' read -r stamp ref email subject content < "$tail_ref"
    echo "$stamp|$newest_node|$email|$subject|$content" > "$tail_ref"
  else
    echo "$newest_node" > 'head'
  fi
  echo "$newest_node" > 'tail'

  # Log necessary data
  IFS='|' read -r _ _ email _ < "$newest_node"
  echo "$email" >> enqueue.order
  echo "${EPOCHREALTIME/[,.]} - RECOVERED enqueue operation" >> queue.log
}
```

Listing 4.2: Enqueue-Widerherstellung

Kam es während des Enqueuens zu einem Abbruch, überprüfen wir, dass die neuste Node im Tail steht. Ist das nicht der Fall, updaten wir die Referenz und stellen die geforderten User-Rechte

sicher.

```
dequeue() {
  last_node="$(tail -1 email.log | cut -d ' ' -f 1)"
  ! test -f "$last_node" && return 0

  # Update refs
  IFS='|' read -r _ ref _ < "$last_node"
  echo "$ref" > 'head'
  if test "$ref" = '-'; then
    echo '-' > 'tail'
  fi

  rm -f "$last_node"

  # Log necessary data
  IFS='|' read -r _ _ email _ < "$last_node"
  echo "$email" >> dequeue.order
  echo "${EPOCHREALTIME/[,.]} - RECOVERED dequeue operation" >> queue.log
}
```

Listing 4.3: Dequeue-Widerherstellung

Wurde während Dequeue abgebrochen, so überprüfen wir, ob die letzte in dem Log abgearbeitete Node immer noch in der Queue existiert. Ist das der Fall, aktualisieren wir die Referenzen und löschen die Node.

In beiden Fällen wird das Lock nachfolgend entfernt, um das normale Arbeiten der Queue zu ermöglichen.

4.3 Dynamischer Inhalt

Die anzuzeigenden Daten werden, wenn ein valider Cookie in dem HTTP-Header übersendet wird, durch drei CGI-Skripte bereitgestellt.

`infra-get-positions.sh` und `infra-update-positions.sh` werden jeweils bei dem initialen Aufruf der Karte und Updaten dieser aufgerufen.

`infra-get-positions.sh` holt alle Daten aus der Datenbank, bei welchen die Positionsmeldungen nicht älter sind als eine Stunde und das Schiff in den letzten fünf Minuten eine Nachricht gesendet hat und schreibt diese leerzeichengetrennt nach `stdout`. Anschließend werden eine Trennzeile

und der aktuellste Index ausgegeben, welcher dann von `infra-update-positions.sh` verwendet wird, um nur die Nachrichten auszugeben, die zusätzlich aktueller sind.

`infra-get-table.sh` gibt mit `mariadb -H` eine HTML-Tabelle aus, die Maritime Mobile Service Identity (MMSI)s, Schiffsnamen, -Zielorte, -Status und den Zeitstempel dessen letzter Nachricht enthält. Den Output von MariaDB formattieren wir mit `tidy`, woraufhin wir lediglich den Inhalt der `<table>`-Tags nach `stdout` ausgeben.

4.4 Daten

Unsere Datenbank umfasst drei Tabellen: ships, positions und users.

ships: {[mmsi:VARCHAR(10), name:TEXT, dest:TEXT, status:TEXT, lst_msg:TIMESTAMP]}

positions: {[idx, lat:FLOAT4, lon:FLOAT4, time:TIMESTAMP], *mmsi:VARCHAR(10)*}

users: {[email:VARCHAR(320), password:TEXT]}

Die Daten von Rhodes werden über einen `ncat`-Prozess empfangen und von einem Skript `worker.sh` gehandhabt. Diese Verbindung wird von einem Watcher `watcher.sh` sichergestellt.

Das `watcher.sh`-Skript wird jede Minute durch einen Crontab-Eintrag aufgerufen. Es liest den aktuellen Nachrichten-Zeitstempel ein, berechnet, ob mehr als eine Minute seit der letzten eingegangenen Nachricht vergangen ist und, ist dies der Fall, beendet die alte Verbindung, sollte sie noch bestehen, und stellt eine neue Verbindung her.

Das Skript `worker.sh` liest und verarbeitet die eingehenden Nachrichten zeilenweise. Die in den Nachrichten gespeicherten Daten werden typenabhängig zugeordnet und in die Datenbank, insofern passend, eingefügt.

```
mariadb <<< "INSERT INTO
  ships (mmsi, name, dest, status)
  VALUES (
    '$mmsi',
    $(echoSqlValue "$shipname"),
    $(echoSqlValue "$destination"),
    $(echoSqlValue "$status_text")
  )
  ON DUPLICATE KEY UPDATE
    name = IFNULL(name, VALUES(name)),
    dest = IFNULL(dest, VALUES(dest)),
    status = IFNULL(status, VALUES(status)),
    lst_msg = CURRENT_TIMESTAMP;"

```

```
15 if test "$type" -le '3' -o "$type" -eq '18' -a "$lat" != '91'; then
16   mariadb <<< "INSERT INTO
17     positions (mmsi, lat, lon)
18     VALUES ($mmsi, $lat, $lon);"
19 fi
```

Listing 4.4: Einfügen in die Datenbank

Der obere dargestellte Code-Abschnitt handhabt die Schiffseinträge und der untere die Positionsmeldungen.

Es werden nicht mit jeder Statusmeldung alle Informationen über die Schiffe mitgeliefert, weshalb wir die bereits in der Tabelle enthaltenen Informationen nur dann updaten, wenn dazu Informationen übertragen werden. Der Zeitstempel wird immer aktualisiert und auf die aktuelle Zeit gesetzt.

In die Positionsmeldungen wird dann eingefügt, wenn es sich um eine Nachricht des Typen 1, 2, 3 oder 18 handelt. Die von Rhodes ausgesandten Daten können fehlerhaft sein und weisen in diesen Fällen die Latitude 91 auf. Diese Nachrichten filtern wir an dieser Stelle ebenfalls hinaus.

Am Ende des Skripts wird der aktuelle Zeitstempel in die Datei `last-msg` geschrieben, welche vom Watcher verwendet wird.

4.5 Frontend

Unser Frontend besteht aus den klassischen drei Technologien: HTML, Cascading Style Sheet (CSS) und JS. Die Seite wird als HTML ausgeliefert, per CSS gestyled und über JS mit Funktionen versehen. Aufgeteilt in verschiedene Dateien werden diese in die HTML-Datei eingebunden.

4.5.1 Akkordeon

Das HTML besteht aus drei Buttons mit dazugehörigen `div`-Elementen und bildet so eine typische Akkordeon-Struktur, die die funktionalen Komponenten der Session, Karte und Tabelle enthält. Die Funktion des Akkordeons wird umgesetzt in der Verwendung von JS-Code, welcher prädefinierte CSS-Klassen setzt.

Standardmäßig besitzen die Segmente die CSS-Klasse `.panel`, welches `display: none;` setzt. Das aktuell ausgewählte Panel wird dazu mit der Klasse `.panelactive` versehen, welches `display: block;` setzt und das Panel so sichtbar macht.

Die Buttons für die Sektion 2 und 3 haben das HTML-Attribut `disabled`, welches dafür sorgt, dass diese nicht angeklickt werden können. Dies dient dem Zweck, die letzten beiden Funktionalitäten erst durch einen erfolgreichen Loginprozess zu ermöglichen.

Im Anschluss an das erfolgreiche Laden der Seite wird die Funktion `init()` aufgerufen, die `window.onload` zugerordnet wurde.

Diese erfüllt zwei Zwecke: Das Aufrufen der Funktion `initAccordion()` und dem Hinzufügen von onclick-Event-Listenern zu den Login- und Logout-Buttons.

`initAccordion()` holt sich alle Elemente mit dem Klassennamen `.accordion` als Array, iteriert durch dieses durch und fügt onclick-Event-Listener hinzu.

Wir speichern, ob die angesteuerte Sektion bereits aktiv ist, schließen daraufhin alle Panels und öffnen nur das Ausgewählte, indem `.panelactive` zu der Klassenliste des Panels, welches zu dem Button gehört, hinzufügen. So erreichen wir, dass lediglich maximal ein Panel zeitgleich aktiv sein kann.

Der Zustand der Anwendung wird global in JS gehalten. Dabei speichern wir Marker, Polylines und die Zeitstempel der Schiffe als Object Literals. Zu dem frühsten Auslösen von Sektion 2 wird die Variable `mapInitialized` auf `true` gesetzt, damit die Initialisierung des Karten-Objekts nur nach dem erstmaligen Aufruf geschieht. Unabhängig davon wird die Funktion `initMapContent` aufgerufen.

Ist Sektion 3 das Ziel, wird das Auslösen der Funktion auf `buildShipTable()` ein Mal pro Sekunde gesetzt.

Die Intervalle der Karte und der Tabelle werden in zwei verschiedenen Vaiablen gespeichert, die ebenfalls dem globalen Zustand der Anwendung angehören - `mapInterval` und `tableInterval`.

In den Event-Listenern wird auch behandelt, dass für die nun geöffnete Sektion irrelevante Intervalle gelöscht werden, um überflüssige Last zu verhindern.

4.5.2 Sessions

Die AJAX-Requests stellen die Schnittstelle zwischen Front- und Backend dar.

Für Login und Logout werden entsprechende AJAX-Requests an `infra-login.sh` und `infra-logout.sh` gesendet. In dem Query String des Login wird dazu das eingegebene Passwort mitsamt der E-Mail gespeichert.

Enthält die Antwort “true“, war der Login also erfolgreich, wird die Funktion `afterLogin()` ausgeführt, welche den Login-Button versteckt, den Logout-Button sichtbar macht sowie das Anklicken der Buttons der Sektionen 2 und 3 ermöglicht.

`afterLogout()` wird als Reaktion auf `infra-logout.sh` ausgelöst. Es setzt die Eingaben in das Login-Formular zurück, versteckt den Logout-Button und verhindert das Anvisieren der Sektionen 2 und 3 erneut, schließt alle Panels und löscht alle Intervalle.

4.5.3 Leaflet-Karte

Die einmalig aufgerufene `initMap()` setzt `mapInitialized` auf "true" und initialisiert die Leaflet-Karte in das Element mit der ID "map". In einer gleichnamigen Variable wird die Karte global zugänglich gespeichert, um den Zugriff anwendungsweit zu ermöglichen. Dazu wird die Funktion `updateMarkerPopupsFromTable()` ausgeführt, welche die Marker-Popups initialisiert.

Danach wird die Funktion `initMapContent()` aufgerufen, welche alle Inhalte der Karte löscht und dann über einen AJAX-Request die Daten über `infra-get-positions.sh` holt und per `handleMapContent()` in die Karte einarbeitet. Letztlich startet sie das sekündliche Updaten durch `update()`.

Dies geschieht bei jedem Öffnen der Karte. Wir haben uns für diese Lösung entschieden, da wir die Karte nur updaten lassen möchten, wenn sie verwendet wird. Öffneten wir die Karte nach vergangener Zeit ohne Updates erneut, ohne Neu-Initialisierung, so könnten wir mit der aktuellen Struktur nur unter erheblichem Aufwand sicherstellen, dass die Daten korrekt sind und angezeigt werden.

Diese Umsetzung ermöglicht uns ein wesentlich einfacher handzuhabendes System. In einem aufkommenden Segemnt des Dokuments werden wir auf die Laufzeiten der einzelnen Komponenten eingehen. Dort sehen wir, dass dieses ursprüngliche Holen aller Daten in etwa doppelt so lange braucht wie das Updaten. Da dies mit jedem Öffnen der Karte lediglich ein Mal geschieht, ist der feststellbare Unterschied in der Praxis gering.

Es lässt sich hier ebenfalls diskutieren, dass die Karte unter regulärer Anwendung nicht im Sekundentakt geöffnet und geschlossen wird.

`handleMapContent()` ruft `processPositionsResponse()` auf, welches die übertragenen Daten parsed. Zeilenweise werden die Daten leerzeichengetrennt in Variablen gespeichert. Die so gewonnene MMSI dient der Indizierung in dem Array, welches von der Funktion returned wird. Die Latitude, Longitude und der Zeitstempel werden in Dreiertupeln an das zur MMSI gehörende Array in der Variable `polyLines` angehangen. Zusätzlich dazu aktualisiert es den zur MMSI gehörenden Timestamp und setzt den Index der letzten eingegangenen Nachricht in `lstMsg`.

In dem Anschluss daran iterieren wir durch alle MMSIs, die geparsed wurden. Wir rufen die Funktionen `handlePolyline` und `handleMarker` auf. Sie sind in ihrer Funktionsweise identisch und kümmern sich jeweils um das Hinzufügen der Polylines und Marker.

Gibt es das zu behandelnde Objekt für die angegebene MMSI noch nicht, erstellen wir ein neues Objekt und fügen es mittels `.addTo(map)` zur Karte hinzu. Existiert es doch, fügen wir

die Koordinaten zur Polyline hinzu oder aktualisieren die Koordinaten des Markers. Hier ist es wichtig zu beachten, dass zwar das Erstellen neuer Polylines mit einem Array von LatLngs möglich ist, jedoch bei dem Hinzufügen alle LatLngs einzeln behandelt werden müssen. Ist das Objekt nicht mehr auf der Karte gerendert, fügen wir es wieder zu dieser hinzu. Für die Marker speziell speichern wir in der globalen Variable `newMarkerAdded`, ob in dem aktuellen Aufruf neue Marker hinzugefügt wurden.

Nach dem Handhaben der neuen Daten, werden nun alte Daten entfernt. Zuerst betrachten wir die Schiffe.

Die Funktion `removeOldShips()` speichert den aktuellen Zeitstempel und iteratiert durch alle gespeicherten Schiff-Zeitstempel. Es wird geprüft, ob der zu dem Schiff zugehörige Marker existiert und ob mehr als die erlaubten fünf Minuten seit der letzten Meldung vergangen sind.

Ist dies der Fall, entfernen wir den Marker aus der Karte und der globalen Zustandsvariable `markers`. Existiert dann zusätzlich die zugehörige Polyline, wird mit dieser ebengleich umgegangen.

Darauffolgend wird `removeOldPositions()` ausgeführt. Erneut speichern wir den Zeitstempel des Aufrufs. Wir iterieren durch alle Polylines und filtern für jede Polyline die Koordinaten heraus, die älter als eine Stunde sind.

Wurden erfolgreich Positionen herausgefiltert, ist also die Länge der Polyline kleiner als zuvor, wird die Polyline neugezeichnet.

Wird die `update()`-Funktion aufgerufen, tut sie der gerade behandelte Funktion gleich. Nur entspringen die Daten dem Aufruf von `infra-update-positions.sh`, welches nur die Daten sendet, welche noch nicht bearbeitet wurden.

Wurden Marker hinzugefügt, ist `newMarkerAdded` wahr, so rufen wir `updateMarkerPopupsFromTable()` auf und setzen die Variable wieder auf falsch.

Die Funktion `updateMarkerPopupsFromTable()` ruft `buildShipTable()` auf. Diese Funktion baut die Tabelle für Sektion 3.

Sie wird hier aufgerufen, um zu ermöglichen, die Namen der Schiffe nach dem Öffnen der Marker-Popups einzusehen. Die Nutzung davon ist also rein kosmetisch. Wir haben uns für diese Vorgehensweise entschieden, weil bei der Datenfrequenz von Rhodes nicht häufig neue Daten einkommen, sodass wir die negativen Effekte der kurzzeitig gedoppelten Anfrage als verkraftbar einschätzen.

Es wird durch alle Einträge der Tabelle iteriert, um die dort verzeichneten Namen mitsamt der MMSIs in die Label zu schreiben.

Nun, nach dem Abarbeiten von `initMapContent()`, rufen wir `map.invalidateSize()` mit einem Timeout von 0 Millisekunden auf. Dies ist dafür notwendig, die Karte fehlerfrei anzeigen zu

lassen. Tests ohne den Timeout haben nicht immer funktioniert. Unsere Ergebnisse wurden durch Onlinesuchen bestätigt: stackoverflow.com.

4.5.4 Tabelle

Das Öffnen der Sektion 3 führt die Funktion `buildShipTable` aus. Diese holt sich per AJAX-Request die fertige HTML-Tabelle von `infra-get-table.sh` und den Inhalt des Elements mit der Id "ship-table" auf diese.

Wurde ein Popup geöffnet, ist also die Variable `highlightedMMSI` gesetzt, so markiert die Funktion `hightlightShipRow` die Reihe, in welcher die in der Variable gespeicherte MMSI enthalten ist.

Nach dem Ablaufen von `buildShipTable()` wird diese durch ein Interval sekündlich aufgerufen und so die Tabelle sekündlich aktualisiert.

4.6 Deployment

Wir ermöglichen das Deployen auf drei verschiedene Arbeitsumgebungen. Dafür stellen wir drei verschiedene Deploy-Skripte bereit: `deploy-local.sh`, `deploy-hopper.sh` und `deploy-prod.sh`.

Da wichtig war, zu beachten, dass die Anwendungen nach dem Deployen möglichst identisch laufen und funktionieren, haben wir uns ein System überlegt, welches nicht erfordert, gleichen Code für verschiedene Umgebungen mehrmals zu schreiben.

Die Skripte `deploy-hopper.sh` und `deploy-prod.sh` rufen das Skript `deploy-local.sh` mit verschiedenen Parametern auf. `deploy-hopper.sh` wird ausgeführt auf dem individuellen Docker über Secure Shell (SSH) und `deploy-prod.sh` auf dem geteilten Docker über SSH. SSH baut eine Verbindung zwischen den Computern auf, über welche Daten übertragen werden.

Die Logik für das Deployen ist damit zentral in `deploy-local.sh` angesiedelt.

```
#!/usr/bin/env bash

dirname='infra-project'
project_root="$(dirname "$0")/.."

cd "$project_root"
sudo -u infra-2025-b ssh mydocker mkdir -p "$dirname"
tar -cf - . | sudo -u infra-2025-b ssh mydocker bash -c ":
tar -C '$dirname' -xf -
cd '$dirname'
./deploy/deploy-local.sh
```

```
12 "
```

Listing 4.5: Deployen auf den Teamdocker

Die Anwendung wird auf den Docker gesendet und in dem Verzeichnis namens “infra-project“ im Homeverzeichnis gespeichert. Zuerst erstellen wir das Verzeichnis, bewegen uns in das Projekt-Root-Verzeichnis, übertragen die Daten dann schließlich und gehen auf dem Docker in dieses Verzeichnis und rufen dann `deploy-local.sh` auf.

Ohne die `sudo`-Aufrufe erhalten wir gleich das Skript `deploy-hopper.sh`.

Das eigentliche Deployen durch `deploy-local.sh` geschieht stufenweise. Für die Datenbank, die Webseite, die CGI-Skripte und die Queue existieren einzelne Funktionen, welche nacheinander aufgerufen werden. Schlägt eine dieser Funktionen fehl, beendet das Skript innerhalb des Vorganges mit einem Fehler-Exit-Code.

4.6.1 Datenbank

Der Aufbau der Tabellen ist gespeichert in der Datei `init.sql`. Sie wird von MariaDB ausgeführt - die `CREATE TABLE IF NOT EXISTS`-Statements erstellen die Tabellen soweit notwendig.

Daraufhin löst das Skript `allow-mariadb-access.sh` aus, das die Datei `.my.cnf` in das Zielverzeichnis `/var/www/data` bewegt und so den Zugriff auf die Datenbank durch `www-data` erlaubt. Existiert dieses Verzeichnis nicht, wird es außerdem erstellt.

Letztlich fügen wir zwei Crontab-Einträge hinzu für den Watcher, den Cleaner und einen Eintrag in die `~/atreboot.sh`, welche nach einem Neustart ausgeführt wird: `allow-mariadb-access.sh`.

```
1  enterCrontabEntry() {
2    local stamp="$1"
3    local cmd="$2"
4    local path="$(getPath "$3")"
5
6    local entry="$stamp $path$cmd &>/dev/null"
7    local tab="$(crontab -l 2>/dev/null)"
8    grep -F "$entry" >/dev/null <<< "$tab" ||
9      echo -e "$tab\n$entry" | crontab -
10 }
```

Listing 4.6: Deployment von Crontab-Einträgen

Die Crontabs editieren wir durch die Funktion `enterCrontabEntry`. Als Argumente werden die Zeitangaben, das Kommando sowie falls nötig der Pfad zur Projekt-Root mitgegeben.

Wird der zusammengesetzte Eintrag nicht in dem aktuellen Crontab gefunden, wird er an diesen appendiert.

4.6.2 Webseite

Existiert das Webverzeichnis `$USER-web/infra-2025` bereits, wird dieses gelöscht. Das Verzeichnis wird (erneut) erstellt - dann werden die relevanten Daten dorthin kopiert.

4.6.3 CGI-Skripte

Die CGI-Skripte werden in das `cgi-bin`-Verzeichnis kopiert.

4.6.4 Queue

```
deployQueue() {
  local queue_dir="/data/queue"
  mkdir -p "$queue_dir" &&
    chown $USER:www-data "$queue_dir" &&
    chmod g+ws "$queue_dir" &&
    cp ../src/queue/*.sh "$queue_dir" &>/dev/null &&
    enterCrontabEntry '* * * * *' 'sudo -u workuser /data/queue/watcher.sh' 'true;' &&
    enterAtrebootEntry '/data/queue/restore.sh' 'true;' ||
    exit 6

  addGroupWriteableFile "$queue_dir/head" '-\n'
  addGroupWriteableFile "$queue_dir/tail" '-\n'
  addGroupWriteableFile "$queue_dir/queue.log" ''
}
```

Listing 4.7: Deployment der Queue

Für die Queue ist es essentiell zu beachten, dass die Berechtigungen korrekt sind.

Zuerst erstellen wir ein Verzeichnis `queue` in dem Data-Verzeichnis und ändern dann die Gruppe des Verzeichnisses zu www-data und geben dieser Lese- und Schreibrechte. Wir kopieren dann die relevanten Daten dorthin und fügen einen weiteren Crontab-Eintrag hinzu, der den Watcher als workuser minütlich ausführt.

In der Regel möchten wir die Skripte ausführen, die sich in unserem Projekt-Verzeichnis befinden. Um dieses nicht immer mitzugeben, setzen wir die Variable `path` standardmäßig auf dieses.

Wollen wir keinen Pfad an das Kommando voranhängen, führen wir stattdessen einfach `true;` aus, um das Argument nicht leer zu lassen.

Damit die Dateien `head`, `tail` und `queue.log` von `www-data` beschrieben werden dürfen, geben wir der Gruppe Schreibrechte. Damit die Queue funktioniert, müssen wir head und tail mit einem Bindestrich initialisieren, welchen wir als zweites Argument in der Funktion mitgeben.

4.7 Testing und Experimente

4.7.1 Unit Tests

Um die Funktionstüchtigkeit der einzelnen Komponenten in Kooperation gewährleisten zu können, haben wir für die einzelnen Teile Tests geschrieben, die auf mögliche Fehler hin überprüfen.

Momentan überprüft werden die Datenbank in Bezug auf den Cleaner und den Watcher sowie das Sessionhandling der Anwendung. Die einzelnen Teste sind Funktionen in dem Unit-Test-Skript, welche sequentiell aufgerufen werden und bei Fehlern eine desktriptive Fehlermeldung ausgeben und mit einem Fehler-Code vorzeitig beenden.

Der Cleaner-Test fügt Daten ein, welche vom Cleaner erfasst werden sollte. Daraufhin wird der Cleaner ausgeführt und danach überprüft, ob sie immer noch in der Datenbank vorhanden sind.

Der Watcher-Test beendet den aktuell laufenden Worker, führt dann den Watcher aus und überprüft danach, ob er läuft.

Der Session-Test beendet alle aktuellen Session, meldet sich über einen curl-Aufruf an das CGI-Skript an und versucht sich danach auf gleiche Art und Weise abzumelden. Nach beiden Vorgängen wird überprüft, ob das Session-Verzeichnis auf der Serverseite existiert.

4.7.2 rhodes2

Über Rhodes erhalten wir in etwa eine Nachricht pro Sekunde. Wir haben uns in diesem Zuge gefragt, wie sich unsere Anwendung verhalten würde, wenn wir schlagartig mehr Nachrichten pro Sekunde erhalten würden. Besonders war die Frage interessant, an welcher Stelle das Bottleneck auftreten sollte.

Für diese Simulation haben wir eine aktivere Datenquelle benötigt, welche wir `rhodes2` getauft haben.

```
lon_modifier='551'

while true; do
  lat_modifier='100'
  while read -r mmsi; do
    echo "TIMESTAMP|3|$mmsi|STATUS|STATUS_TEXT|TURN|SPEED|ACCURACY|8.$lon_modifier|5
    ↪  3.5$lat_modifier|REST"
    (( ++lat_modifier ))
  done < mmsi.txt
  (( ++lon_modifier ))
done | ncat -lk 1234 &
echo "$!" > database-test/ncat2.pid
```

Listing 4.8: rhodes2.sh

Wir iterieren durch alle in der Datei `mmsi.txt` vorbereiteten MMSI und geben für sie Datenzeilen aus. In der vorbereiteten Ausgabezeile variieren die eingesetzte MMSI, Langitude und Latitude, welche wir nach erfolgreichen Iterationen inkrementieren. So stellen wir sicher, dass sich Positionsmeldungen nicht wiederholen und dass sich MMSIs nicht überschneiden. Durch die Schleifenstruktur hat jede MMSI eine zugehörige Latitude - die Longitude inkrementiert sich bei allen MMSIs gleichmäßig verteilt. Die resultierenden Streckenwege verlaufen auf unserer Kartendarstellung horizontal.

Die Ausgaben werden über ncat wie die von Rhodes nutzbar gemacht. Wir speichern zusätzlich dessen PID ab, damit wir das Skript vollständig automatisiert beenden können.

4.7.3 Wie verhält sich die Datenbank bei übermäßig vielen Daten?

Zuerst haben wir uns gefragt, wie sich die Datenbank-bezogene Komponente der Anwendung unter mehr Last verhält.

Wir haben ein Skript geschrieben, welches `rhodes2` mit unterschiedlichen vielen Anzahlen an MMSIs startet. Pro Aufruf starten wir einen ncat-Prozess, der die Daten in unseren Worker leitet. Wir löschen anfangs alle vorhandenen Positionen und messen dann für 60 Sekunden jede Sekunde, wie viele Positionsmeldungen sich momentan in der Datenbank befinden. Am Ende jedes Durchlaufs beenden wir den rhodes2-Prozess dann.

Diese Ausgaben speichern wir in temporären Dateien ab, welche nach Ablauf aller fünf Durchgänge mit 100 bis 500 MMSIs in eine Datei zusammgeführt werden, welche durch ein gnuplot-Skript den folgenden Graphen erstellt:

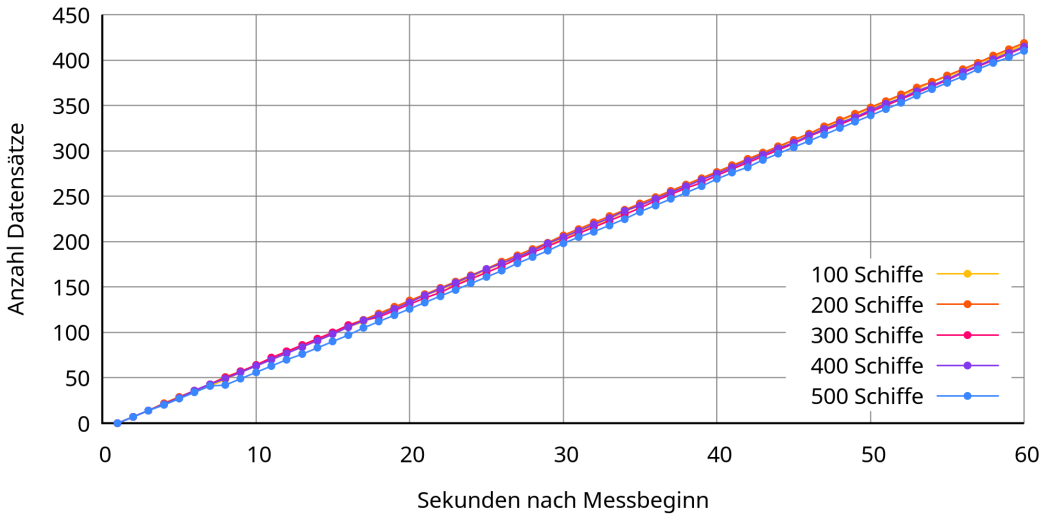


Abbildung 4.2: Anzahl an Datensätzen über eine Minute hinweg

Es ist erkennbar, dass die Änderungsraten über verschiedene Schiffsanzahlen hinweg fast identisch sind. Die durchschnittliche Änderungsrate beträgt 6.92 Positionsmeldungen pro Sekunde. Hierbei ist anzumerken, dass `rhodes2` wesentlich mehr Daten pro Sekunde bereitstellt, als in die Datenbank aufgenommen werden können.

Eine Betrachung des Workers offenbart, dass zwei `mariadb`-Aufrufe sowie drei Subshells die einzigen Prozessaufrufe des Skripts darstellt. Testmessungen ergeben, dass 14 `mariadb`-Aufrufe etwa 0.71 Sekunden benötigen.

Unter der zusätzlichen Betrachtung dessen, dass der Worker lediglich 66 Zeilen umfasst, schlussfolgern wir, dass dieses Einfügen einer jeden vearbeiteten Zeile mit dediziertem `mariadb`-Aufruf das Bottleneck darstellt.

Wir hatten dieses Phänomen in dieser Form auch in den Veranstaltungen thematisiert sowie Lösungsvorschläge behandelt. In diesem Projekt haben wir uns gegen kompliziertere, performantere Lösungsansätze entschieden, da Rhodes durchschnittlich etwa eine Meldung pro Sekunde empfängt. Gehen wir vom Extremfall aus, dass mehrere Schiffe aus Zufall zeitgleich Daten senden, so kann es dazu kommen, dass die Datenverarbeitung an ihre Grenzen kommt. Es ist Beobachtungen zufolge jedoch unwahrscheinlich, dass dieser Fall eintritt.

4.7.4 Wie verhält sich die Karte bei übermäßig vielen Daten?

Wir haben uns gefragt, ob die Karte zu hängen beginnt, wenn übermäßig viele Polylines und Marker gezeichnet werden. Auch war interessant, ob die dynamische Darstellung in der Lage sein würde, die Daten, die in der Datenbank hinzugefügt werden, in Echtzeit hinzuzufügen.

Dafür haben wir `rhodes2` mit 400 MMSIs gestartet und vom Worker verarbeiten lassen.

Manuelle Betrachtungen haben ergeben, dass es zu keinen merkbaren Performance-Einbrüchen kam sowie dass etwa sieben Polylines pro sekündlichem Update aktualisiert wurden. Dies entspricht der Anzahl an Datensätzen, die laut unserer vorherigen Grafik und Schlussfolgerung darzustellen sind.

4.7.5 Wie verhält sich die Queue bei übermäßig vielen Logins?

Wir wollten testen, ob die Reihenfolge in der Abarbeitung beibehalten wird, ob alle E-Mails ohne Duplikate verarbeitet werden und wie lange es dauert, alle E-Mails abzuarbeiten, wenn eine hohe Anzahl an Logins vorgenommen wird.

Um dies zu untersuchen, haben wir eine Bibliothek geschrieben, welche die erforderlichen Funktionalitäten implementiert.

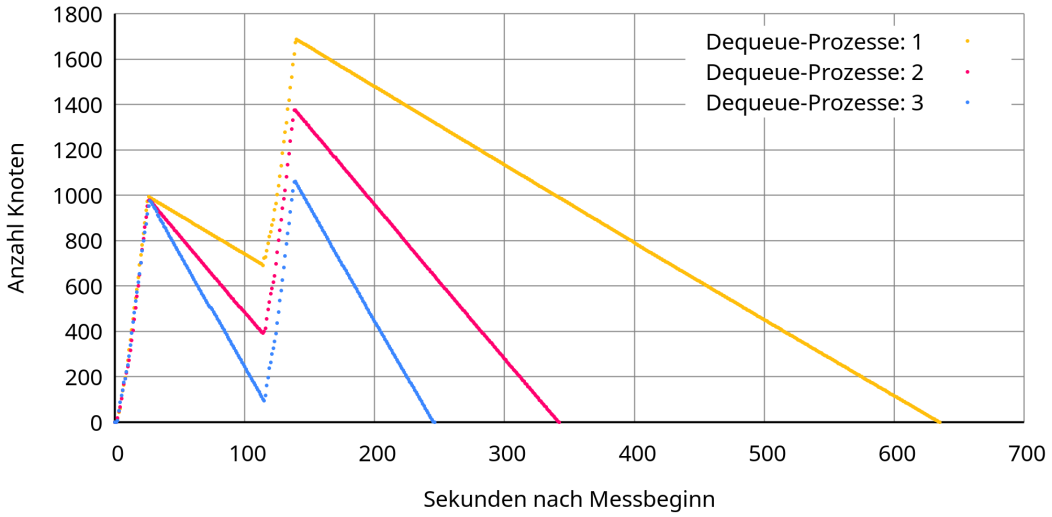


Abbildung 4.3: Anzahl von Elementen in Queue über Zeit

Die Experimente haben wir mit insgesamt 2.000 Logins durchgeführt, zuerst 1.000, anschließend wurde zwei Minuten gewartet und erneut 1.000 Logins. Wir haben uns für diese Vorgehensweise entschieden, da wir zwei Wellen von Logins simulieren wollten.

Das Abarbeiten der Logins hat mit einem worker-Prozess aufgrund des `sleep 0.3` etwa 635 Sekunden gedauert, was man der übergestellten Darstellung entnehmen kann.

Bei zwei worker-Prozessen halbiert sich die Abarbeitungszeit, welche sich bei drei Prozessen noch weiter verringert. In der eigentlichen Anwendung haben wir uns für einen entschieden, da diese Anzahl an Logins ungewöhnlich wäre.

Wir überprüfen die Reihenfolge von Enqueue und Dequeue durch eine Funktion, welche zwei Logging-Dateien miteinander vergleicht, an welche wiederum bei dem Enqueuen und Dequeuen appendiert wird. Dies hat keine Fehler ergeben.

Dann haben wir überprüft, ob alle E-Mails verarbeitet wurden. Dafür haben wir uns den E-Mail-Log angeschaut, welcher ebenfalls durch die Dequeue-Funktion erstellt wird, und haben überprüft, ob die Menge der Login-Versuche pro User der Dequeue-Menge pro User entspricht. Es wurden alle Logins einmalig bearbeitet.

5 User-Durchläufe

5.1 Login

5.1.1 Aufruf der Seite

Unser `index.html` wird von dem Browser des Users geladen. Dieser erstellt mithilfe des DOM die HTML-Elemente und deren Struktur. Ein Akkordeon mit den Sektionen: Login, Karte und Tabelle. Das eingebundene Stylesheet der CSS-Datei `main.css` sorgt für das Layout und ermöglicht eine Manipulation der Sichtbarkeit dieser Elemente mit Klassen wie `hidden` und `panelactive` oder Attributen wie `disabled`. Zudem wird unsere JS-Datei `app.js` geparsed, welche ebenfalls eingebunden ist.

5.1.2 Initialisierung nach Laden der Seite

Der JS-`window.onload`-Event-Handler ruft die Funktion `init()` auf. Diese fügt `onclick`-Event-Listener für die Buttons des Akkordeons sowie für den Login/Logout-Button hinzu.

5.1.3 Aktueller Zustand

Die Login-Sektion ist aktiv, während die Karte und Tabelle `disabled` sind. Dadurch kann auch nur dieser Bereich aufgeklappt/geschlossen werden. Innerhalb dieser Sektion wird zwischen zwei HTML-Formular-Elementen für Login/Logout unterschieden. Die Sichtbarkeit regelt die Zuweisung der Klasse `hidden`. Erkennbar sind ein Button, auf dem “Login“ steht, sowie zwei Eingabefelder mit Platzhalter-Text: “E-Mail“ und “Passwort“.

5.1.4 Benutzeraktion: Login

An dieser Stelle kann der User nun seine bereits in der Datenbank vorhandenen Credentials eintippen und per Klick auf den Button die Loginfunktion `login(email, pwd)` ausführen.

5.1.5 Backend

Diese Funktion erzeugt eine GET-Anfrage an `infra-login.sh?email=<email>&password=<password>`. Anhand der extrahierten Credentials wird `checkuser` aufgerufen. Diese Funktion führt eine SQL-Abfrage der MariaDB-Tabelle "users" aus:

Im Falle eines erfolgreichen Abgleichs wird per `createsession` eine neue Session angelegt. Dazu wird ein Verzeichnis `/var/www/data/session-<cookie>` erstellt, wobei cookie ein zufälliger String ist. Per `logaction`-Funktion wird dieser Vorgang in `/var/www/data/web.log` protokolliert (Zeitstempel, Cookie, E-Mail, ...) und es erfolgt eine Response an das Frontend in Form von Text mit "true" als Keyword.

5.1.6 Kein Match bei dem Abgleich mit der Datenbank

Wenn keine passenden Credentials gefunden werden, beendet sich `infra-login.sh`, ohne die "true"-Response auszugeben. Das Frontend erkennt dadurch den fehlgeschlagenen Login-Versuch - somit wird keine Session angelegt und es findet auch keine E-Mail-Simulation statt. In `login(email, pwd)` wird dies dadurch behandelt, dass die Eingabefelder zurückgesetzt werden, sich deren Rand für 1,5 Sekunden rot färbt und per Entfernen der `hidden`-Klasse ein zusätzliches `div`-Element erscheint, welches den User textuell auf die ungültige Eingabe hinweist.

5.1.7 E-Mail-Simulation

Nach einem erfolgreichen Login wird asynchron zu der Verarbeitung im Frontend, ebenfalls das Versenden einer E-Mail simuliert. Dies geschieht durch den Aufruf von `enqueue`. Die Abarbeitung der E-Mails wird in einer Warteschlange innerhalb des Dateisystems organisiert. Parallele Zugriffe werden durch das Spinlock-Prinzip der Funktionen `lock/unlock` verhindert. Jede Mail erhält eine eindeutige Datei `node-<timestamp>` mit dem Inhalt `timestamp | - | email | subject | content`, wobei der Bindestrich als Platzhalter für spätere Referenzen auf Nachfolger-Knoten innerhalb der verketteten Liste dienen. Operationen der Queue werden in `queue.log` protokolliert. Ein Worker-Prozess `(worker.sh)` läuft kontinuierlich im Hintergrund und verarbeitet die Nodes per `dequeue`-Funktion. Dabei werden die Inhalte (email, subject und content) in `email.log` geschrieben und der Knoten danach gelöscht. Ein

watcher.sh-Skript stellt per worker.pid sicher, dass worker.sh stetig läuft und startet den Prozess andernfalls neu.

5.1.8 Frontend-Anpassungen

Sollte die Server-Response jedoch einem "true" entsprechen, so wird Set-Cookie: session=<cookie> gesetzt und die afterLogin(email)-Funktion ausgeführt. Das Logout-Formular ist nun sichtbar und wiederum Login ausgeblendet. Der Akkordeon-Header für Sektion 1 wird auf Logout gesetzt und es wird textuell "Angemeldet als: <E-Mail>" ausgegeben. Außerdem erscheint ein neuer Button mit der Aufschrift "Logout". Die Akkordeon-Buttons für die Karte und Tabelle werden per .disabled = false freigeschalten.

5.2 Logout

Vorausgesetzt wird ein zuvor erfolgreicher Login.

5.2.1 Benutzeraktion: Logout

Sollte die Logout-Sektion noch nicht geöffnet sein, so kann dies durch das Anklicken des entsprechenden Akkordeon-Buttons erreicht werden. Der in initAccordion() deklarierte Event-Listener erkennt den Klick und klappt das Panel mithilfe von panelactive auf. Dabei werden die zuvor offenen Sektionen der Karte oder Tabelle geschlossen. Sollte nun der User auf den Logout-Button klicken, so wird logout() aufgerufen, welches per XMLHttpRequest eine GET-Anfrage an infra-logout.sh im Backend sendet.

5.2.2 Backend-Verarbeitung

getsession liest den Session-Cookie aus der Anfrage und ermittelt die zugehörige E-Mail. logaction protokolliert in /var/www/data/web.log wie zuvor für den Login. Existiert eine gültige Session, so wird per Textresponse von:

Set-Cookie: session=; Expires=Thu,01 Jan 1970 00:00:00 GMT, der Cookie im Browser gelöscht.

deletesession entfernt das Verzeichnis /var/www/data/session-<cookie>.

5.2.3 Frontend-UI-Anpassungen

Nach dem erfolgreichen Abschluss von `logout()` wird `afterLogout()` ausgeführt. Jenes sorgt dafür, dass die HTML-Elemente des UI für Sektion 1, auf den Zustand zurückgesetzt werden, welchen sie beim initialen Laden der Seite haben. Die Buttons für die Karte/Tabelle sind nun wieder `disabled`.

5.3 Karte

Vorausgesetzt wird ein zuvor erfolgreicher Login.

5.3.1 Öffnen der Karte

Sollte die Karten-Sektion noch nicht geöffnet sein, so kann dies durch das Anklicken des entsprechenden Akkordeon-Buttons erreicht werden. Der in `initAccordion()` deklarierte Event-Listener erkennt den Klick und klappt das Panel mithilfe von `panelactive` auf. Dabei werden die zuvor offenen Sektionen der Karte oder Tabelle geschlossen.

5.3.2 Initialisierung der Leaflet-Karte

`initMap()` wird aufgerufen und erstellt ein `map`-Objekt mit dem Mittelpunkt `[53.54, 8.5835]`, den Koordinaten der Hochschule, und fester Zoomstufe. Es werden die Hintergrund-Layer von OpenStreetMap geladen.

5.3.3 Laden der Karteninfos

`initMapContent()` startet eine erste XMLHttpRequest-GET-Abfrage an `infra-get-positions.sh` im Backend.

Es werden Positionsdaten der letzten Stunde angefordert von Schiffen, die in den letzten 5 Minuten mindestens eine AIS-Nachricht verschickt haben. Am Ende wird dann noch der aktuellste Positionsindex mitgegeben. `handleMapContent(responseText)` zerlegt diese Daten und erstellt für jede MMSI ein Array von Koordinaten (lat, lon, timestamp). Es werden Marker sowie Polylines basierend auf Positionen angelegt. Die Marker von Schiffen ohne regelmäßige Updates werden entfernt und veraltete Positionen gegebenenfalls aus den Polylines abgeschnitten.

5.3.4 Periodische Updates

`setInterval(update, 1000)` ruft einmal pro Sekunde `update()` auf, welches einen AJAX-Request mit GET `../cgi-bin/infra-update-positions.sh?lstMsg=<last_idx>` an das Backend auslöst. Der relevante Unterschied zu dem `infra-get-positions.sh`-Skript besteht darin, dass nur Positionsdaten mit einem Index größer als dem zuletzt verwendeten gefiltert und im Textformat übertragen werden. Somit können dann die bestehenden Marker/Polylines erweitert werden, ohne immer den gesamten Datensatz anzufordern.

5.3.5 Benutzeraktion: Marker-Popup öffnen

Der Mausklick auf einen Marker löst das `popupopen`-Event aus und setzt die Variable `highlightedMMSI` auf den zugehörigen Wert des Schiffs. Umgekehrt wird bei dem Schließen eines bereits geöffneten Popup-Feldes per `popupclose`-Event die Variable wieder auf `null` zurückgesetzt.

5.4 Tabelle

Vorausgesetzt wird ein zuvor erfolgreicher Login.

5.4.1 Benutzeraktion: Tabelle öffnen

Sollte die Tabellen-Sektion noch nicht geöffnet sein, so kann dies durch das Anklicken des entsprechenden Akkordeon-Buttons erreicht werden. Der in `initAccordion()` deklarierte Event-Listener erkennt den Klick und klappt das Panel mithilfe von `panelactive` auf. Dabei werden die zuvor offenen Sektionen Logout oder Karte geschlossen.

5.4.2 Backend-Verarbeitung

`buildShipTable()` wird aufgerufen und sendet per XMLHttpRequest eine GET-Anfrage an `infra-get-table.sh` im Backend. `getsession` prüft ob der User eingeloggt ist. Falls dem so ist, erfolgt eine SQL-Abfrage der Tabelle “ships“ für alle aktiven Schiffe.

Dieser Output wird dann per `mariadb -H` und `tidy` als HTML-Format zurückgegeben. Jedoch filtert zuvor noch `cleanHtmlByTidy()` überflüssige Tags heraus, sodass nur `<table>`-Inhalt an das Frontend geschickt wird.

5.4.3 Darstellung im Frontend

Der HTML-Content wird in das Div-Element `<div id="ship-table">` geschrieben. Es erscheint eine HTML-Tabelle mit einer Zeile pro aktivem Schiff und Spalten für MMSI, Schiffsnamen, Zielort, aktuellem Status und dem Zeitstempel der Letzte Nachricht. Die `main.css` sorgt hierbei für Tabellenränder, Farben und mögliches Highlighting.

5.4.4 Synchronisation mit Marker-Popups

Sollte die JS-Variable `highlightedMMSI` durch die Karte zuvor gesetzt worden sein, dann ruft `buildShipTable()` die Funktion `highlightShipRow()` auf. Diese iteriert durch die Zeilen der Tabelle und sucht nach einem Match für die MMSI. So kann per Vergabe der `highlighted`-Klasse eine Zeile farblich hervorgehoben werden.

5.4.5 Dauerbetrieb

Da bei dem Öffnen der Tabellen-Sektion `setInterval(buildShipTable, 1000)` gesetzt wird, geschieht der Aufbau der Tabelle im Sekundentakt und die Daten bleiben stets aktuell.

6 Anwendung aus Nutzungssicht

6.1 Login

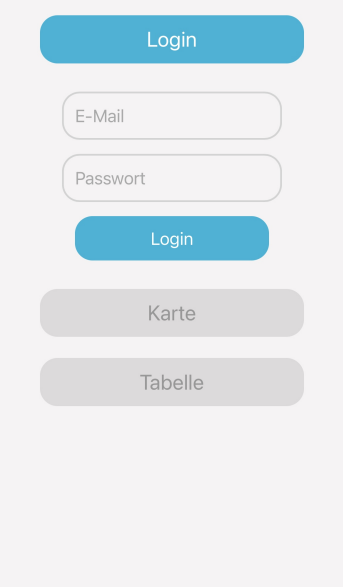
Das Akkordeon ist innerhalb des Browserfensters per `flex-direction: column` vertikal orientiert und zentriert. Wir versuchen die Farbgebung möglichst schlicht und modern zu halten. Dazu wählten wir einen hellen Hintergrund mit Blau als Akzentfarbe der aktivierten Buttons. Dem gegenüber stehen ausgegraute Buttons mit dem `disabled`-Attribut und das Hovern mit der Maus verändert den Cursor per `cursor: not-allowed`. `border-radius: 20px` sorgt für abgerundete Ecken der UI-Elemente, welche weicher wirken und somit zu der Benutzerfreundlichkeit beitragen. Ein fehlgeschlagener Loginversuch führt zu einer roten Fehlermeldung und die Ränder der Eingabefelder nehmen ebenfalls kurzzeitig diese Warnfarbe an.

6.2 Karte

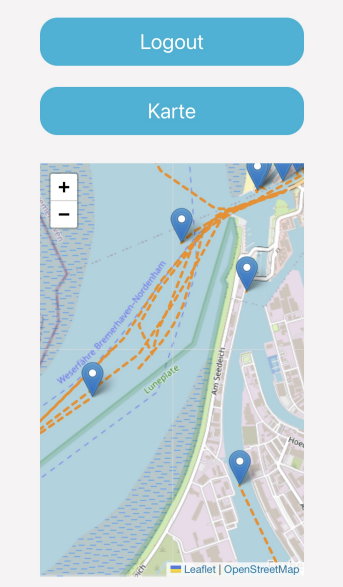
In dem zweiten Akkordeon-Panel ist unsere Leaflet-Karte eingebettet. Sie nimmt 70% der Höhe des Browserfensters ein (`#map { height: 70vh;}`). In Bezug auf eine dynamische Breite passt sich diese responsiv, mittels `width: 80%` an. Allerdings kann sich der User anhand der standardmäßigen Leaflet-Funktionalitäten, per Drag der Maus auf der Weltkarte frei bewegen und den Zoom verändern. Aktive Schiffe werden als blaue Marker dargestellt und deren Bewegungen anhand von gestrichelten, orangenen Linien mithilfe von `color` und `dashArray` gerendert. Die Interaktion mit den Markern öffnet/schließt jeweils ein kleines Popupfenster, in dem die MMSI und der Name des dazugehörigen Schiffs eingeblendet werden.

6.3 Tabelle

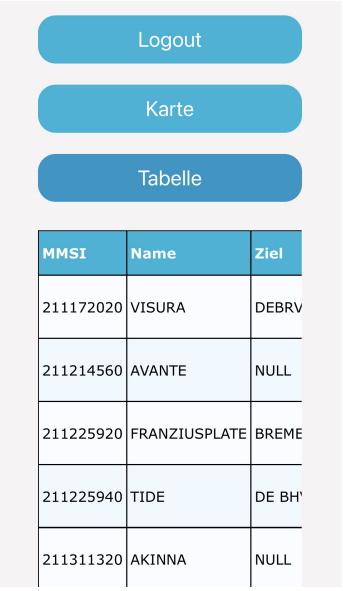
Die Tabelle ist unter dem dritten Akkordeon-Panel eingebettet und nutzt jeweils eine Scrollbar bei zu vielen Einträgen bzw. einer kleinen Displaygröße, wie bei der Nutzung von Handys, per `overflow-y: scroll`. Die Überschriften orientieren sich an dem blau-weiß Farbschema des Akkordeons. Angezeigt werden die MMSI, der Schiffsname, das aktuelle Ziel, Status und der Zeitstempel der zuletzt gesendeten AIS-Nachricht. Für eine bessere Lesbarkeit wechseln sich die Hintergrundfarben zeilenweise mit `table tr:nth-child(even)` ab. Zusätzlich verwenden wir einen dunkleren Farbton für `table tr:hover`. Wurde zuvor während der Verwendung der Karte ein Popup geöffnet, so nutzen wir die Klasse `highlighted`, um die dazugehörige Zeile der Tabelle mit einem dunklem Orangeton hervorzuheben.



(a) Login



(b) Karte



(c) Tabelle

7 Versionsgeschichte

Tabelle 7.1: Beispiele wichtiger Commits

Datum Autor:in	Nachricht	Kommentar
30.07. Leon Stüve	DATA: Adds first version of a worker.sh - Parse data from rhodes - Adds function to get relevant data from messages - Adds function to check for empty strings - Adds function to import data into database	Hier haben wir die erste Version des Workers hinzugefügt, welcher sich um das Einspeichern in die Datenbank kümmert.
04.08. Lukas Weber	SESSION: Adds basic session handling This implements basic session handling utilizing the file system for storing data persistently, as requested. We are able to login and logout.	Mit diesem Commit begann die Funktionalität der Anwendung.
18.08. Luca Focken	TESTS: Adds rhodes2 To test the handling of lots of data of our database and website we implement a script that produces test data by echoing random numbers for random mmsis.	`rhodes2` stellt den Anfang unserer Tests dar.
20.08. Bastian Buch	WWW: Adds ship names to marker popups Added the function updateMarkerPopupsFromTable(). On opening the table section every marker that matches to the MMSIs of the table gets their popup adjusted to show the Name and MMSI.	Ab diesem Punkt befand sich unsere Anwendung in einem betrachtlichen Zustand.

Git war in unserer Arbeit ein zentrales Werkzeug. Unsere Commit-Nachrichten hatten wir bereits in der frühen Vorbereitung auf ein definiertes Format festgesetzt, um die Arbeit mit ihnen zu streamlinen.

Über sie können wir nachlesen, welche inhaltlichen Änderungen zwischen Commits stattgefunden haben. Entwickelte man die Anwendun in der Zukunft weiter, könnte man durch sie auch einen dedizierten Changelog einführen.

Durch `git checkout <hash>` können wir auf alte Versionen der Software zurück gehen. Die Deploy-Skripte funktionieren dabei bislang allesamt identisch in der Verwendung.

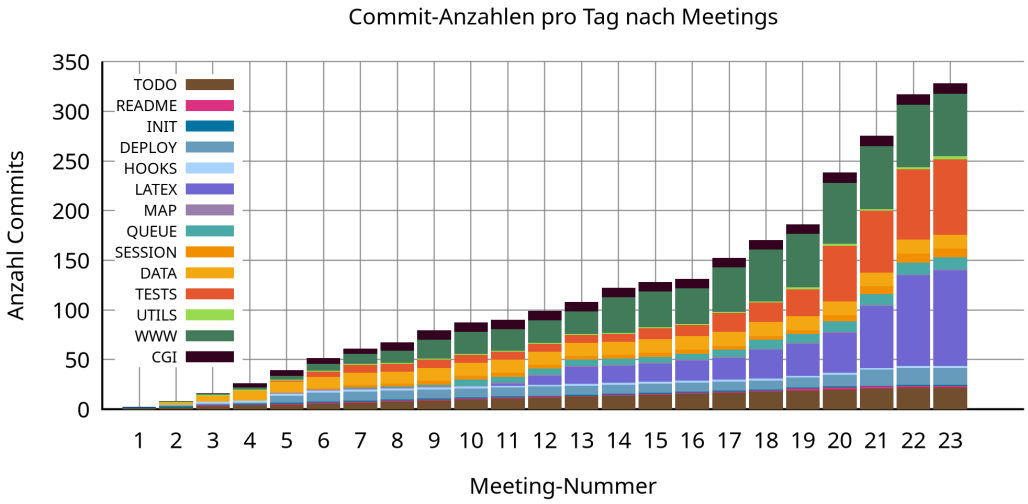


Abbildung 7.1: Commit-Verlauf

Die Commits sind so formattiert, dass sie ein Schlüsselwort aufweisen, welches in der Datei `keywords.txt` existiert. Darauf folgt dann die deskriptive Titelzeile, welche den Satz ‘This commit...‘ vollendet. Der Körper ist beliebig bestehend aus Fließtext und Auflistungen. Zusätzlich haben wir alle Commits englisch verfasst.

Überstehend, in der Abbildung 7.1, sieht man zusätzlich noch einmal, wie häufig die von uns verwendeten Schlüsselwörter nach welchem Teamtreffen verwendet wurden.

8 Beobachtungen

8.1 Messungen

Gegen Fertigstellung der vollfunktionstüchtigen Anwendung haben wir verschiedene Messungen durchgeführt, um dessen Leistungsfähigkeit, Durchlaufzeiten und dementsprechend Umweltfreundlichkeit zu beurteilen.

8.1.1 Skript-Laufzeiten

Überstehend zu sehen ist die durchschnittliche Laufzeit der verschiedenen Skripte bei zehn sequentiellen Messungen. Es ist hevorzuheben, dass `get-positions.sh` die mit Abstand längste Laufzeit aufweist. `update-positions.sh` und `get-table.sh` benötigen etwa ein Viertel der Zeit.

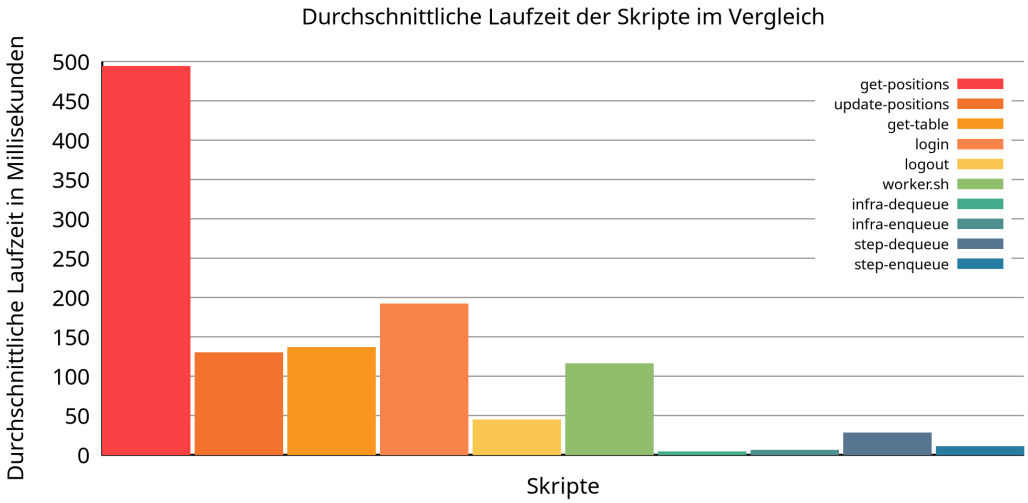


Abbildung 8.1: Laufzeiten der einzelnen Skripte

Das bedeutet, dass das Reduzieren der Anfragen an `get-positions.sh` effektive positive Veränderungen in Beziehung auf die Performance verursacht. Auch kann der Grafik entnommen werden, dass `get-table.sh` ungefähr gleich zeitintensiv ist, sodass das Abfragen aller Tabellendaten bei jedem Aufruf verkraftbar ist.

Wir haben uns hier für die etwas besondere Struktur entschieden, das Aktualisieren der Kartenpositionen und der Tabelle getrennt vorzunehmen. Wie anhand der Grafik erkennbar ist, stellt diese Trennung, da im Regelfall nur eine Anfrage pro Sekunde versendet wird, keinen performancetechnischen Nachteil dar.

Dazu haben wir den Anspruch vertreten wollen, die beiden Komponenten der Karte und Tabelle modular voneinander unabhängig implementieren und testen zu wollen. In Ausblick auf zukünftige Anforderungen vertreten wir die Ansicht, dass diese Entscheidung die Flexibilität ermöglicht.

Manuelle Messungen haben dazu ergeben, dass 100 Durchläufe vom Umwandeln durch sed und Erstellen durch MariaDB durch den Schalter `-H` mit `tidy` in ihrer Laufzeit etwa identisch sind.

Der resultiernde Laufzeit-Unterschied in `infra-login.sh` und `infra-logout.sh` ergibt sich nach genauerem Betrachten der Skripte aus dem Prozessaufruf von MariaDB. Auch die Laufzeit von `worker.sh` lässt sich dadurch erklären, denn ein MariaDB-Aufruf benötigt etwa 50 Millisekunden.

Die Infra-Enqueue- und -Dequeue-Prozesse sind im Vergleich zu den anderen Skripten sehr schnell. Der Hauptgrund liegt darin, dass in diesen Skripten kein Prozess gestartet wird. Die

STEP-Äquivalente hingegen starten Prozesse, sodass ihre Laufzeit mehr als dem Doppelten entspricht.

Dieses Phänomen wird in dem auffolgenden Abschnitt detaillierter analysiert.

8.1.2 STEP-Infra-Vergleich

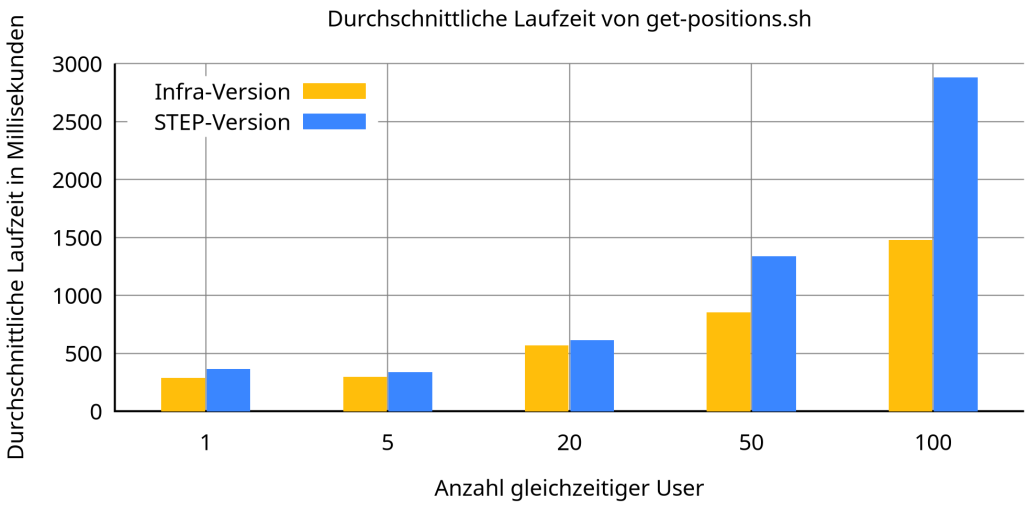


Abbildung 8.2: STEP-Infra-Vergleich

Für diese Betrachtung haben wir das `infra-get-positions.sh`-Skript mit den Mitteln aus STEP implementiert. Dann haben wir die durchschnittlichen Laufzeiten bei variierenden Anzahlen gleichzeitiger Nutzenden gemessen.

Wir haben drei Prozessstarts in der Infra-Version sowie zwölf in der STEP-Version identifiziert.

In allen fünf Versuchen wies die Infra-Version eine geringere Laufzeit auf. Dazu ist der Trend erkennbar, dass mit ansteigenden User-Anzahlen die Differenz zwischen den Versionen größer wird.

Wir haben bei dem Entwerfen der Skripte versucht, die Prozessstartanzahl zu minimieren und nur dann Prozesse gestartet, wenn wir mit den uns bekannten Mitteln keine alternativen Lösungsansätze ausarbeiten konnten.

Ob die Anwendung komplexer werden würde, ersetzten wir alle Prozessaufrufe durch Alternativen, können wir nicht über unseren Wissensstand hinweg einschätzen. Den aktuellen Stand empfinden wir als übersichtlich und verständlich.

8.1.3 Laufzeiten zu Useranzahlen

Für diesen Test messen wir die durchschnittliche Laufzeit der Skripte, die per AJAX aufgerufen werden, jeweils zu verschiedenen Useranzahlen.

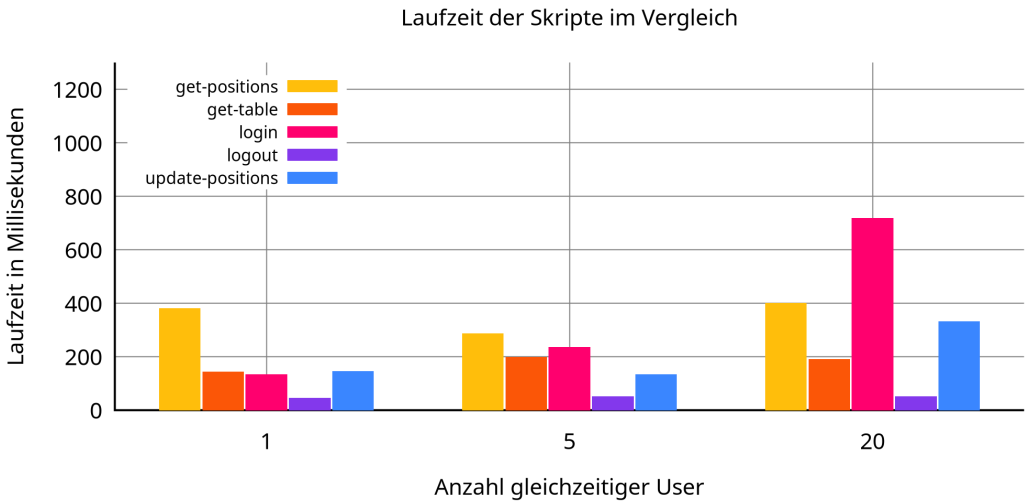


Abbildung 8.3: Laufzeiten zu Useranzahlen

Die Laufzeiten sind mit Ausnahme von `infra-login.sh` und `infra-update-position.sh` vergleichbar. Es ist ein Anstieg von fünf auf 20 gleichzeitig Nutzende für `infra-get-positions.sh` sichtbar - ansonsten sind größere Differenzen für `infra-login.sh` und `infra-update-positions.sh` zu vermerken.

Dies erklären wir damit, dass pro Loginversuch ein Aufruf von MariaDB notwendig ist. Die geringeren Laufzeiten der Skripte `infra-get-positions.sh`, `infra-get-table.sh` und `infra-update-positions.sh` resultieren daraus, dass das Einloggen die Aufrufe verstreut.

Für das Simulieren der Nutzenden haben wir `single-user.sh` geschrieben, welches die typischen Aktionen eines einzelnen Users simuliert und die gemessenen Laufzeiten in eine Logdatei appendiert. Über zwei Argumente wird mitgegeben, wie oft die Karte und Tabelle jeweils aktualisiert werden sollen.

In `multi-user.sh` rufen wir dieses Skript so oft im Hintergrund auf, wie in dem ersten Argument mitgegeben.

Die durchschnittliche Laufzeit haben wir dann durch die Logdateien berechnet.

8.1.4 Systemauslastung zu Useranzahlen

Für das Analysieren der Systemauslastung gebrauchen wir dieselben Skripte. Nun aber Lesen wir die Central Processing Unit (CPU)-Auslastung aus, indem wir während eines jeden Durchlaufs `hbv_dockerstats` in einer Endlosschleife aufrufen und den Durchschnitt berechnen.

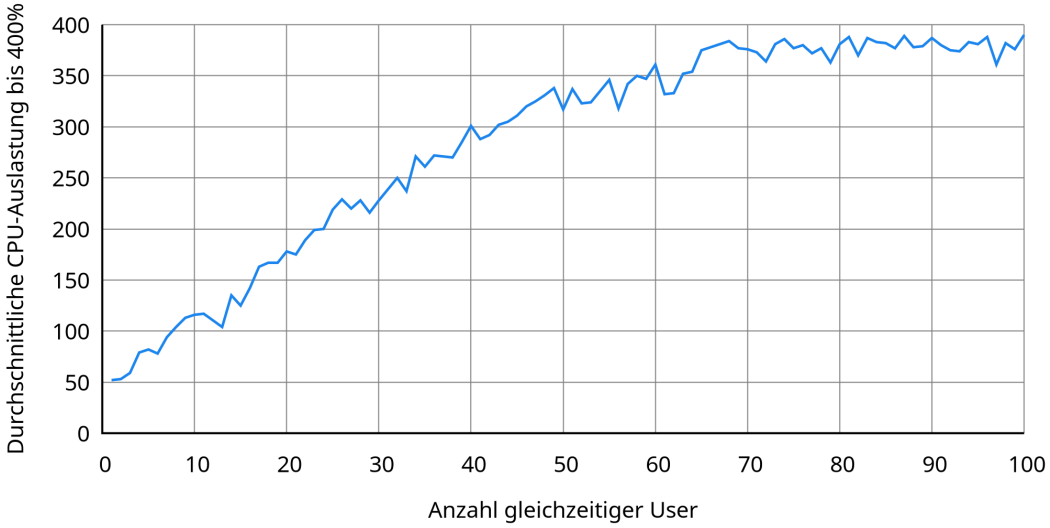


Abbildung 8.4: Systemauslastung zu Useranzahlen

Ab einer Anzahl von etwa 40 gleichzeitigen Nutzenden erreicht die Auslastung 300%. Bis dahin steigt die Auslastung annähernd linear an - danach flacht die Auslastung etwas ab, da wir uns der Maximalauslastung nähern.

Die Grundlast liegt bei etwa 50%, da das Monitoring, das Abfragens der Daten von Rhodes und das Simulieren der E-Mails durchgehend Prozesse im Hintergrund arbeiten lassen.

8.2 Monitoring

/docker-infra-2025-b-web/infra-2025/monitoring/

Das Überwachen unserer Anwendung ermöglichen wir mit einer dedizierten Seite. Sie enthält einen Graph, welcher die vergangene Minute umfasst und sekündlich aktualisiert wird. Er zeigt die Anzahl laufender Sessions, neuer Positionsmeldungen und aktiver Schiffe an.

Wir haben uns gegen das Anzeigen der aktuellen Systemauslastung entschieden, da der Zugriff auf diese lediglich unergonomisch sowie unter der Annahme geschieht, dass Prozesse von uns nicht weiter verwaltbar auf dem System laufen.

8.3 Mögliche zukünftige Änderungen

8.3.1 Daten-Format

Für die Positionsdaten der Schiffe per `infra-update-positions.sh` verwenden wir aktuell einfache Textzeilen, wie diese:

`'211599340' 53.5364 8.58041 2025-08-26 23:53:06 2025-08-26 23:53:06`

Der Vorteil der Darstellungsweise liegt bei einer einfachen Erzeugung innerhalb der Bash in Kombination mit MariaDB sowie einer schlanken Struktur mit demnach geringer Datenlast.

Jedoch verwalten wir die darin enthalten Daten in unserer `app.js` anhand von Array-Indizes. Je mehr unterschiedliche Infos zu einer MMSI gespeichert werden, desto unübersichtlicher wird der Umgang. Und andere Programmiersprachen, bei denen Arrays keine dynamische Speichergrößen vorweisen, sprechen ebenfalls gegen die Verwendung eines solchen Konzeptes.

Sollten nun also zukünftig Erweiterungen, wie zum Beispiel die eines Schiffstypen, hinzukommen, bietet sich eine objektorientiertere Lösung mit JavaScript Object Notation (JSON)-Notation an:

```
[
  {
    "mmsi": "211331640",
    "lat": 53.54718,
    "lon": 8.58392,
    "pos_time": "2025-08-26 21:23:59",
    "ship_time": "2025-08-26 21:24:02",
    "idx": 12345
  }
]
```

Listing 8.1: Mögliche JSON-Umsetzung

Diese wäre dann anwendungsfreundlicher und leichter zu debuggen. Ebenso zerstört die Skalierbarkeit nicht direkt ein bestehendes Parsing. Jedoch ist die Generierung damit auch aufwändiger für die Bash geworden, sodass es zu größeren Verzögerungen zwischen dem Anfragen und Erhalten der Daten kommen kann.

8.3.2 Encryption

Ein Aspekt, der von der Aufgabenstellung nicht explizit gefordert wurde, jedoch in jeder reellen Softwareanwendung mit Nutzerdaten relevant wird, besteht darin, diese Daten beispielsweise

für die Übertragung oder das Speichern innerhalb einer Datenbank zu verschlüsseln.

Als Experiment haben wir für Letzteres einen funktionierenden Ansatz getestet. Dazu wurden bei dem manuellen Einfügen von Usern in MariaDB und das Prüfen der Credentials mit `checkuser` die Passwörter mit dem Advanced Encryption Standard (AES)-128 ver-/entschlüsselt [1].

```
checkuser() {
  local input_email="$1"
  local input_password="$2"
  read -r p_key < /var/www/data/hashkey

  if test -n "$(mariadb --defaults-file=/var/www/data/.my.cnf -N -s <<< "SELECT email
  ↪  FROM users
    WHERE email='$input_email'
    AND AES_DECRYPT(password, '$p_key') = '$input_password';")"; then
    ses_email="$input_email"
    return 0

  fi
  return 1
}
```

Listing 8.2: Mögliche Erweiterung von checkuser

Der private Schlüssel wurde dafür einmalig generiert und in der Datei “hashkey“ abgelegt. Der Ort dieser Datei und deren Zugriffseinschränkung per Portable Operating System Interface (POSIX)-Rechte, muss dabei mit bedacht werden.

Allerdings ist unser Verfahren in der Praxis unsicher, da alle Passwörter mit demselben privaten Schlüssel verschlüsselt werden und allgemein Hashing-Verfahren sinnvoller für die Authentifizierung genutzt werden können, ohne Gefahr zu laufen, die Klartextvariante einer nutzenden Person wiederherstellen zu können. Dann wäre es aktuell noch im Bereich des Möglichen, dass identische Passwörter unterschiedlicher Nutzende zu gleichen Ciphertexten führen.

Außerdem besteht oft das größere Risiko darin, wie die eingegebenen Credentials des Frontends sicher für den Abgleich mit der Datenbank zu dem Server gelangen. Eine Frage, mit der sich bei einer Fortsetzung unseres Projektes definitiv weiter auseinandergesetzt werden muss.

9 Quellen

9.1 General

Den Großteil unserer Ausarbeitung konnten wir anhand der Mittel aus STEP und Infra bereits umsetzen. An einigen Stellen finden jedoch auch externe Funktionen und Befehle Verwendung, auf die wir an diesem Punkt einmal näher eingehen wollen.

Um die AIS-Nachrichten richtig in die Tabelle einzubetten, haben wir uns für jeden Typen die Art der Daten angeschaut [2].

Darunter fallen auch optionale Spielerreien wie `:nth-child()` [3], welches wir in unserem Stylesheet nutzen, um Tabellenzeilen abwechselnd zu färben.

Auch die Tutorials zu Gnuplot [4] und JS [5] dienten uns als Hilfestellung.

9.2 Leaflet

Sowie die Möglichkeit die Attribute von Markern [6] oder Polylines [7] anzupassen, wodurch wir unsere Positionsdaten als orangene, gestrichtelte Linien visualisieren. Wie aus unserem Leaflet-Klassendiagramm zu entnehmen ist, nutzen wir auch diverse Methoden wie `invalidateSize()` oder `bindPopup()` die sich in der praktischen Anwendung, außerhalb der bloßen Logik als notwendig erweisen [8].

9.3 Javascript

In manchen Bereichen wie JS gibt es natürlich auch viele unterschiedliche Herangehensweisen für die Lösung eines Problems und ebensoviele Bibliotheken mit Methoden und Klassen.

So nutzen wir die `Element.classList` [9] als DOM-Werkzeug für die Selektierung bestimmter Elemente anhand ihrer CSS-Klassen - wie `getElementsByClassName()` [10].

Um zu überprüfen ob ein erfolgreicher Login vorliegt nutzen wir `String.includes()` [11].

Für bestimmte Aktionen, die aus der `Bash` bekannt sind, mussten vergleichbare JS-Alternativen gefunden werden. So zum Beispiel für das Parsing und die Stringmanipulationen per `toUpperCase()` [12], `filter()` [13], `parseFloat()` [14], `replace()` [15], `split()` [16], `trim()` [17] sowie `EPOCHREALTIME` mit JS-`Date` [18] und `Date.parse()` [19].

Damit wir alte Positionsmeldungen und Schiffe von der Karte löschen können, nutzen wir `delete` [20].

Bei dem Setzen der markierten Tabellenzeilen iterieren wir durch die gesamte Tabelle und überspringen dabei Zellen, die nicht existieren oder leer sind. Dafür nutzen wir `continue` [21] anstatt eines `if`-Statements, damit der Code übersichtlicher bleibt.

Für das Monitoring nutzen wir ein Gnuplot, welches wir sekündlich austauschen. Uns war vorab bewusst, dass `Blob`s als Quellen von Bildern gesetzt werden können. Die Lösung dafür fanden wir in einer Stackoverflow-Antwort [22]. Wir setzen den `responseType` [23] auf `blob` und erstellen eine `ObjectURL` um die Quelle des Bildes auszutauschen.

9.4 MariaDB

Auch der Umgang mit SQL erforderte Recherche, da wir die AIS-Nachrichten mitsamt Zeitstempel speichern wollten, um späteres Filtern zu ermöglichen. Dafür haben wir den Typen `TIMESTAMP` [24] genutzt und mittels `INTERVAL` [25] gefiltert.

Für das Deployment und Erstellen der Tabellen bietet sich dann auch die Nutzung von `IF NOT EXISTS` [26] an, damit die Tabellen auch nur dann erstellt werden, wenn sie nicht existieren.

Um Informationen von Schiffen zu speichern, die erst durch spätere Nachrichten bereitgestellt werden, haben wir nach einer Möglichkeit gesucht, wie man Daten in den Tabellen richtig überschreiben kann. Wir nutzen `ON DUPLICATE KEY` [27], um die neuen Daten den MMSIs zuzuordnen und `IFNULL` [28], damit bereits vorhandene Daten nicht durch `NULL` überschrieben werden.

9.5 Gnuplot

Um die Grafiken schöner darzustellen nutzen wir eine handvoll Einstellungen, die Gnuplot bereitstellt.

Damit die Legende der Grafiken anschaulicher und anders plaziert wird nutzen wir den Key `key` [29] und geben diesem bestimmte Werte mit.

Für die Histogramme nutzen wir noch die Option `rowstacked` [30] sowie `xticlabels` [31], um anzugeben, welche Werte an der X-Achse stehen soll.

test

10 Selbstreflexion

10.1 Bastian Buch

Noch bevor ich von der PO 2017 auf die aktuelle Fassung gewechselt hatte, war Infrastruktur kein fester Bestandteil der Pflichtmodule. Da bei mir STEP schon einige Jahre zurücklag, ergab sich eine deutliche Diskrepanz zwischen meinem Wissensstand und dem Versuch, Vernetzte Systeme zu belegen. Nach dem Wechsel stellte sich die erste Teilnahme von Infra als Überforderung heraus. Zudem waren meine Teampartner Wiederholer des Moduls, wodurch während der Veranstaltungen lediglich eine weitere Person aus unserer Gruppe präsent war. Ich hatte mich dann dagegen entschieden, dem Vorschlag der damaligen Gruppe zu folgen, die Anforderungen funktional aufzuteilen und dadurch kein wirklich umfassendes Verständnis aller Technologien zu erlangen. Nach Absprache mit Herrn Radfelder fasste ich den Entschluss, noch ein Jahr zu warten, dabei Module wie Datenbanken 1 und Webprogrammierung zu belegen sowie diverse Programmier-Scheine, vor deren praktischer Arbeit ich zuvor Selbstzweifel gepflegt hatte.

Die wichtigste Erkenntnis daraus besteht darin, die vermittelten Inhalte der Veranstaltungen nicht als alleinigen Input für den eigenen Wissensstand zu sehen, sondern als Impulse, die zum Selbststudium und programmatischem Experimentieren anregen sollen. Außerdem erwarb ich solide Kenntnisse in Bezug auf SQL, JavaScript, XML/XHTML und Java, die es mir ermöglichten, Defizite an anderen Stellen – vor allem in der Bash – nachzuarbeiten, ohne in allen Bereichen bei Null anfangen zu müssen.

Das Modul Infrastruktur zeichnet sich besonders dadurch aus, breit gefächerte Parallelen zu weiteren Modulen zu schaffen. Neben den bereits erwähnten Bereichen bietet die Gruppenarbeit ein praxisnahes Umfeld für grundlegende Softwareentwicklung. Das Erstellen sinnvoller Lasttests war ein ständiger Begleiter und vermittelte ein Gespür für Performance und Größenordnungen. Dies sowie kleinere Einblicke in atomare Prozesse und deren Synchronisation erinnerten mich an Parallelprogrammierung bzw. das Multicore-Praktikum. Weitere Parallelen waren die Arbeit mit dem Versionsverwaltungstool Git, das Aufgreifen bereits bekannter Diagrammtypen und Visualisierungen, sowie ein gesamtheitliches Zusammenführen in Form einer wissenschaftlichen Ausarbeitung.

Im Hinblick auf meine Arbeitsweise habe ich mir im Verlauf des Semesters umfangreiche handschriftliche Notizen angefertigt. Dies war notwendig, da keine Vorlesungsfolien vorliegen und die Tutorialseiten nicht dazu dienen diese Aufgabe zu übernehmen. Außerdem ist es bei Live-Coding und Eigenrecherche sinnvoll, relevante Informationen für später festzuhalten. Eine Vor- und Nachbearbeitung der Veranstaltungen war ein fester Bestandteil meiner Wochenplanung. Dies hilft dabei sich vor Beginn der nächsten Veranstaltung erneut in die bisherige Thematik hineinzuversetzen. Die Nachbearbeitung bestand aus dem Ergänzen der Notizen auf Basis von Praxiserfahrungen und Recherche.

Als persönliche Stärke würde ich dabei mein schnelles Verständnis von Code hervorheben, jedoch fällt es mir schwer, eigenständig Aufgabenstellungen zu entwickeln, ohne dass diese von außen vorgegeben werden. Das erneute Schreiben von Skripten diente vorwiegend dem Üben der Syntax und dem Wiederholen bestimmter Konstrukte wie etwa Flags. Der dafür aufgewandte Zeitrahmen hing jedoch auch davon ab, mit welchem Thema wir uns aktuell beschäftigten und wie weit ich bereits während der Präsenzzeit mit der Umsetzung gekommen war. Die zuvor angesprochene Reiteration der Inhalte war davon jedoch unabhängig. Persönlich ist es für mich etwas schwer abzuschätzen, wo die Grenze zwischen Code liegt, der aus dem Stegreif geschrieben werden kann, und dem Punkt, ab dem das Nachschlagen in den MAN-Pages oder das Nutzen externer Quellen wie Stack Overflow angemessen ist. Intuitiv würde ich mich hier auf die im Semesterprojekt spezifizierten Technologien beschränken, da diese als Bausteine für die Umsetzung vorausgesetzt werden. Dahingehend bin ich zuversichtlich, inzwischen eine hinreichende Kontinuität im Umgang mit diesen erworben zu haben, ohne in den Unterlagen nachschlagen zu müssen. Jedoch unterlaufen mir dabei noch häufiger Flüchtigkeitsfehler, oder es braucht mehrere Anläufe die Syntax, erneut aus der Erinnerung, korrekt zu reproduzieren. Dies sind Anzeichen, dass mir noch weitere Praxisübung helfen könnte.

Der Einfluss der Gruppenarbeit stellte sich für mich als Bereicherung heraus. Zum einen als Unterstützung bei Wissenslücken und Verständnisproblemen, die vorwiegend daraus resultierten, dass ich die Einführungsveranstaltung STEP nicht im vorherigen Semester belegt hatte. Als auch als Diskussionsgrundlage jeglicher Art. Dabei ging es nicht nur um Inhalte, sondern auch darum, zu trainieren über Software/Code zu sprechen. Zu den Soft Skills zählen beispielsweise die Herangehensweise an Problemstellungen – also deren Zerlegung in Komponenten, die Modellierung und das Berücksichtigen aller relevanten Aspekte. Hier unterschieden sich die individuellen Prioritäten der Teammitglieder: Sprachkonventionen und Code-Style (Clean Code), das Ersetzen komplexer oder fremder Sprachkonstrukte durch verständlichere Syntax, Separation of Purpose usw.

Die Arbeit mit Git erwies sich als weniger problematisch als zu Beginn erwartet. Mir sind nur zwei Fälle bekannt, in denen Merge-Konflikte auftraten, die manuell aufgelöst werden mussten. Dies liegt daran, dass wir hauptsächlich gemeinsam während der Teammeetings gecoded haben. Dabei wechselte sich die schreibende Person, welche seinen Bildschirm überträgt, zwischen den Sessions immer wieder mal ab und die Anderen nutzen die Zeit für Recherche, Mitdenken/Diskutieren und lokales Testen. Der fertige Zwischenstand wurde dann als Commit vor Beginn des nächsten Moduls gepusht. Am Anfang und Ende dieser Treffen wurden jeweils unsere TODOs besprochen. Diese bestanden aus einem Vorbereitungsanteil und den Zielen, die wir währenddessen umsetzen wollten. Ersterer konnte alle betreffen, z. B. sich Lösungsansätze für ein aktuelles Problem zu überlegen, oder individuell verteilte Aufgaben, die keine gemeinsame Bearbeitung erforderten – etwa Bugfixes, Formatierung, kleinere Unit-Tests oder CSS-Optimierungen. Jeder musste seine Ergebnisse so präsentieren, dass alle Teammitglieder diese vollständig nachvollziehen konnten. Gegebenenfalls wurde anschließend als Gruppe evaluiert, ob noch Änderungen vorgenommen werden sollen.

Abschließend würde ich Infrastruktur als eines der Informatik-Fächer bezeichnen, bei dem Studierende am meisten lernen können. Ein Grund dafür ist sicherlich die geforderte aktive Mitarbeit und das Selbststudium – im Gegensatz zu Modulen, in denen lediglich Inhalte eines Foliensatzes auswendig gelernt und in der Klausur einmalig abgefragt werden. Dennoch wird die BashProgrammierung auch in Zukunft nicht zu meinen Favoriten zählen. Allerdings erkenne ich an, dass es sinnvoll ist, als Informatiker auch in diesem Bereich einen gewissen Erfahrungssatz erworben zu haben.

10.2 Luca Focken

Welche der Technologien beherrsche ich aus dem Stegreif?

Ich denke, dass ich fast alle Technologien, die wir dieses Semester behandelt haben, zu einem gewissen Maß, ohne lange zu überlegen, anwenden kann. Sei es das Arbeiten mit einem Linux-System, das schreiben einer SQL-Query oder das bauen einer simplen JavaScript-Anwendung. Bei dem Aufsetzten eines Apaches muss ich manchmal noch nachschauen, in welchen Dateien die Ports eingestellt werden müssen und welche Module ich genau aktivieren möchte.

Wie übe ich die Artefakte ein?

Währen des Semesters haben mir vor allem die Wochenaufgaben dabei geholfen, die neu gelernten Technologien einzuüben. Außerdem habe ich am Anfang der vorlesungsfreien Zeit zwei private Projekte angefangen, bei denen ich auch mit MariaDB, Apache2 und JavaScript arbeite, wodurch ich weitere Sachen gelernt habe.

Welche Methode funktioniert bei mir besonders gut?

Am “leichtesten“fällt mir das Arbeiten mit Watchern und, dass lernen und anwenden von neuen Konzepten wie die Queue.

Wo habe ich immer wieder Schwierigkeiten?

Meine größten Probleme habe ich dabei, “gute“ Namen für meine Variablen, Funktionen und Skripte zu finden, sodass andere aber auch ich selbst besser zuordnen können, was genau diese tun.

Wo klafft eine Lücke zwischen Verständnis und Handlungskompetenz?

Während des Semesters hatte ich kein so klares Bild darüber wie genau die einzelnen Funktionen für das Umgehen mit Sessions funktionieren. Erst durch das intensivere Arbeiten damit, habe ich wirklich verstanden wie die einzelen Funktionen miteinander arbeiten. Ich denke, dass ich die Bibliothek einmal selbst hätte programmieren sollen, damit ich direkt ein besseres Bild darüber gehabt hätte.

Wieviel Zeit habe ich im Semester zum Einüben investiert?

Ich würde sagen, dass ich pro Woche ungefähr zwei bis drei Stunden zu Hause in Infrastruktur investiert habe. Dabei gab es mal Wochen, wo ich deutlich mehr an Sachen gearbeitet habe, aber auch Wochen, wo ich deutlich weniger gemacht habe, weil andere Module mehr Zeit in Anspruch genommen haben.

Welche Aspekte der Teamarbeit haben sich bei mir verändert?

Ich versuche seit der Teamarbeit mehr Kommentare zu schreiben und meinen Code übersichtlicher zu gestalten, sodass andere besser und schneller verstehen können, was ich erreichen möchte. Dadurch fällt es mir auch leichter, älteren Code direkt nachvollziehen zu können ohne.

Was habe ich durch das Arbeiten mit git und die notwendige Aufteilung von Funktionalität in Einheiten persönlich gelernt?

Ich habe Schwierigkeiten dabei kleinschrittig zu arbeiten. Durch das arbeiten mit git jedoch, habe ich viel darüber gelernt wie genau man an große Projekte rangeht und wie man jene in viele kleine Schritte aufteilen und strukturiert bearbeiten kann. Diese Erfahrung hat mir auch bei privaten Projekten geholfen.

Habe ich meine Fähigkeit, über Software/Code zu sprechen, verbessern können?

Über Software bzw. Code zu sprechen ist eine Sache, die mir nicht leicht fällt. Ich habe Schwierigkeiten dabei meine Gedanken anderen Personen zu erklären. Die Teamarbeit hat etwas geholfen, aber es ist immer noch eine Sache, in die ich mehr Übung und Zeit investieren muss.

Welche Verbindungen zu anderen Modulen sind mir klar geworden?

Es sind mir Verbindungen zu allen Modulen, die ich dieses Semester besucht habe aufgefallen. Die größten Überschneidungen waren, das Arbeiten mit Algorithmen, das Umgehen mit einer Datenbank und die Ähnlichkeit von Interrupts und Events.

Abschließen möchte ich sagen, dass ich dieses Semester noch mehr Spaß hatte als Letztes. Der Hauptgrund dafür ist die Teamarbeit in Infra und Rechnerarchitektur gewesen, da ich so viele neue Sachen gelernt habe, die ich so privat nie entdeckt hätte. Außerdem hat sich auch die Art wie ich Code schreibe verändert, da ich sehr viel neue Inspiration aufgenommen habe.

10.3 Leon Stüve

Als letzten Teil der Ausarbeitung soll jedes Teammitglied eine Selbstreflexion schreiben in der darauf eingegangen werden soll, wie man sich im Semester entwickelt, wie die neuen Konzepte aus den Vorlesungen eingeübt wurden und wo eventuell noch Schwächen sind.

In einer der ersten Vorlesungen haben wir uns erneut mit der Funktionsweise eines Watchers beschäftigt. Ich denke das ich in der Lage bin in einigen Minuten einen Watcher zu programmieren und auch die einzelnen Komponenten eines Watchers verstanden habe. Eingeübt habe ich

dies und auch andere neue Konzepte dieses Semester durch das regelmäßige Programmieren dieser. Somit habe ich mir soweit es möglich war täglich oder alle zwei Tage 30 Minuten Zeit genommen und habe die Konzepte aus den Vorlesungen erneut zu erarbeiten oder versucht diese auswendig zu programmieren. Ich habe allerdings die Erfahrung gemacht, dass ich diese Art von Konzepten am besten lerne, indem ich diese in Projekten anwende, seien es private oder für das Modul. Auf diese Art kann ich mich später an diese Beispiele erinnern und die so verwendeten Konzepte besser im Kopf behalten. Somit bin ich immer auf der Suche nach kleinen Projekten um das neu gelernte anzuwenden.

Ein neues großes Konzept dieses Semesters war die Verwendung von git. Ich habe in der Vergangenheit bereits git verwendet, allerdings nur für meine privaten Projekte und hatte somit keine Erfahrung wie dies in der Teamarbeit verwendet wird. Besonders im Laufe dieses Projektes habe ich gelernt, dass git nicht nur zur Versionierung sondern auch zur Kooperation gemacht wurde. Ich habe außerdem gelernt, wie wichtig klar formulierte Commit-Nachrichten sind, damit nicht nur ich verstehe welche Änderungen vorgenommen wurden, sondern alle aus dem Team. Auch die Unterteilung des Codes in sinnvolle Einheiten hat geholfen, Merge-Konflikte zu vermeiden, da so praktisch nie Änderungen von unterschiedlichen Personen an denselben Funktionen vorgenommen wurden. Dies hat gezeigt, dass eine sinnvolle Unterteilung des Codes nicht nur die Lesbarkeit erhöht sondern auch das Arbeiten im Team vereinfacht. Ich denke ich habe ein gutes Verständnis erlangt, über die git-Funktionalitäten, welche wir in den Vorlesungen behandelt haben und habe somit eine gute Grundlage geschaffen, um in den nächsten Modulen auf diesem Wissen aufzubauen.

Dieses Semester haben wir außerdem die Programmiersprache JavaScript kennengelernt. Angefangen mit AJAX-Request, die es JavaScript ermöglichen, Nachrichten an einen Server zu schicken und asynchron auf dessen Antwort zu reagieren. Ich denke ich habe die in der Vorlesung vorgestellte Funktionsweise dieser Anfragen verstanden und kann diese auch in meinem Code verwenden und bin gespannt, was noch alles mit dieser Art der Kommunikation möglich ist in den kommenden Semestern. Ein weiteres Konzept, welches wir kennengelernt haben, ist die funktionale Programmierung. Dies ist für mich ein größtenteils neues Konzept und bereitet mir noch einige Schwierigkeiten. Ich denke ich habe verstanden, warum diese Art der Programmierung nützlich sein kann und dass es viele neue Möglichkeiten bietet. Allerdings fällt es mir aktuell noch schwer mich an diese Art der Programmierung zu gewöhnen und in meinen eigenen Projekten anzuwenden, ohne das die Lesbarkeit des Codes deutlich abnimmt.

Verbindungen zu anderen Modulen wurden mir besonders während der Arbeit an unserem Projekt klar. So war es zum Beispiel wichtig den Inhalt von Datenbanken-I zu beherrschen. Nicht nur mussten die entsprechenden Tabellen erstellt werden, in denen unsere Daten gespeichert werden, sondern es mussten auch Daten eingefügt, aktualisiert, abgefragt und gelöscht werden. Die grundlegenden Funktionalitäten wurden zwar in der Veranstaltung vorgestellt, allerdings hat das zusätzliche Wissen aus Datenbanken-I geholfen diese Aufgaben effizienter zu bearbeiten. Auch die Art und Weise wie Tabellen innerhalb eines Textes beschrieben werden können, haben wir in Datenbanken-I gelernt, auch dies haben wir in der Ausarbeitung angewendet. Somit ist

es gut, dass ich die Aufgaben für Datenbanken-I regelmäßig bearbeitet habe und es mir so gut möglich war alle neuen Konzepte des Moduls zu erlernen und somit in diesem Projekt zu verwenden.

Ein weiterer Aspekt, welcher mir während der Teamarbeit aufgefallen ist, ist, dass es mir leichter fällt Dinge zu erklären wenn ich eine graphische Darstellung von dem Konzept habe, welches ich erklären möchte. Ich habe gemerkt, dass ich anderenfalls dazu neige in meinen Erklärungen zu springen was das Verstehen erschwert. Dies müssen keine klar strukturierten oder aufwendigen Darstellungen sein, oft reicht auch bereits schon eine simple Skizze in einem Zeichenprogramm wie Inkscape oder Paint. Diese Darstellungen sollen mir lediglich helfen, den roten Faden nicht zu verlieren und festzuhalten was bereits erwähnt wurde und was noch nicht.

Zusammenfassend würde ich sagen, dass ich dieses Semester eine Menge Neues gelernt habe. Ich denke ich habe die vermittelten Konzepte größtenteils verstanden und kann diese auch anwenden. Ich denke ich habe große Fortschritte gemacht was die Programmierung mit Bash und JavaScript angeht. Ich muss zugeben, dass das Programmieren mit der Bash, zumindest mit unseren aktuellen Mitteln, mir wirklich Freude bereitet, obwohl ich diesem im letzten Semester doch noch eher skeptisch gegenüberstand. Ich werde mich auf jeden Fall noch weiter mit dem Konzept der funktionalen Programmierung auseinandersetzen müssen, da dies bestimmt auch ein Thema in den kommenden Semestern bleiben wird und ich dies somit beherrschen muss. Ich freue mich auf das kommende Semester und bin gespannt, wie wir auf dem erlangten Wissen aufbauen werden.

10.4 Lukas Weber

Die größte Entwicklung, die ich selbst bei mir über den Verlauf des Semesters feststellen konnte, war in Bezug auf Servertechnologien.

Gleich in der zweiten Veranstaltung haben wir gelernt, wie wir (im Eigenantrieb) den Apache2-Webserver installieren und konfigurieren. Während mir die Idee hinter Webservern sowie einige ihrer großen Namen keinesfalls eine Neuheit waren, hatte ich mich bislang aufgrund meines lediglich rudimentären Verständnisses von Netzwerken nie daran getraut, selbst mit ihnen zu experimentieren. Aus Spaß habe ich nun sogar angefangen zu versuchen, einen (wirklich sehr) einfachen Webserver selbst zu programmieren.

Und gerade dadurch, dass ich ein wenig mehr über virtuelle Maschinen in Linux gelernt habe - Schlüsselworte Lima, KVM und QEMU - habe ich mir eher zugetraut, mich mit diesem wirklich wichtigen Aspekt der Softwarewelt vertraut zu machen.

Bislang hatte ich immer alleine an Projekten mit git gearbeitet. In der Arbeit im Team habe ich ein wesentlich tieferes Gefühl für die Werkzeuge entwickelt, die git einem bereitstellt - ergo Pushen, Pullen oder auch fortgeschrittenere Konzepte wie das Lösen von Merge Conflicts oder Rebasen, Tags und ein wenig über Branches.

Was ich damit primär ausdrücken möchte: Im zweiten Semester konnte ich einige Dinge erfolgreich lernen. Rückblickend denke ich, dass ich die in den Veranstaltungen behandelten Aspekte gut beherrsche - und, dass ich den Großteil davon auch wirklich ohne weitere Gedanken verlieren zu müssen aus dem Stegreif runtertippen könnte.

Wohl am fehleranfälligsten ist hier eine effizientere Form der Queue - aber auch hier ist das Konzept deutlich und nach etwas Fehlerbehebung gelingt mir eine funktionstüchtige Implementation.

Einzig das Setzen der Cookies kann ich nicht guten Gewissens aus dem Stegreif vornehmen. Das Prinzip der Cookies als Schlüssel-Wert-Paare stellt keineswegs das Problem dar, sondern viel eher, dass ich momentan nicht zufriedenstellend einschätzen kann, welche Einstellungen und Präfixe wann angebracht sind. Siehe hier beispielsweise `__Secure-` als Präfix.

Dieses Semester hatte ich mich auch an der Lernmethode probiert, welche wir in der Einführungsveranstaltung kennengelernt haben: Spaced Repetition. Ich hatte mir dafür eine Terminal User Interface (TUI)-Anwendung geschrieben, in welche ich Karten eintragen konnte und die mir solche vorlegt, die ich zu bearbeiten habe.

Auch diesbezüglich bin ich nun ein wenig schlauer als vorher. Ich habe festgestellt, dass jedes kleine Detail einzutragen und so wiederholen zu wollen für mich einfach nicht funktioniert. Ich fühle mich eher von der Anmasse von Karten erschlagen und verliere rasch mein Elan. Für Details hingegen sagt es mir deutlich zu und auch nächstes Semester möchte ich diese Technik (und vielleicht sogar meine Anwendung) verwenden - für Details und Dinge, die man “stumpf“ auswendig lernen muss, hat es beeindruckend funktioniert. Für den Rest eignet sich für mich ordinäres, selteneres Wiederholen.

Auch im Bereich der Teamarbeit gab es meinerseits ein paar Änderungen - jedenfalls im Vergleich mit dem ersten Semester. In dieser Teamarbeit konnte ich etwas verinnerlichen, dass es in einem leistungsstarken Team auch angebracht ist, beispielsweise kleine Aufgaben zu verteilen. Dies steht im Kontrast zum ersten Semester, wo unsere Teamarbeit wesentlich gekippter im Verhältnis war.

Ich denke, dass meine Fähigkeit, über Code zu sprechen, weiterhin dem erwarteten Stand gerecht wird. Dazu konnte ich einige Kontaktpunkte zwischen der Infrastruktur-Veranstaltung und anderen Veranstaltungen identifizieren. Darunter fallen beispielsweise Datenbanken 1 (Schreiben von SQL-Statements) und Rechnerarchitektur (kritische Abschnitte).

Etwas neben den Themen dieses Moduls habe ich ansonsten weiter über Nischen dieses Feldes gelernt, die mich interessieren. Beispielsweise habe ich mich - zum ersten Mal weniger oberflächlich - damit auseinandergesetzt, wie man Neovim mit dem Lazy-Pluginmanager mit lazy-loaded Plugins erweitert.

Zusammenfassend fühle ich mich bezüglich meiner Kenntnisse ein wenig selbstsicherer. Dieses Jahr habe ich mich deshalb auch endlich getraut, meinen Hauptcomputer von Windows auf

Linux umzuwechseln und in die Gebiete einzutauchen, vor welchen ich mich bisher gescheut habe.

Literaturverzeichnis

[1] Oracle. „Encryption and Compression Functions.“ en, besucht am 26. Aug. 2025. Adresse: https://dev.mysql.com/doc/refman/8.0/en/encryption-functions.html#function_aes-encrypt.

[2] ITU. „AIS Messages.“ en, besucht am 30. Juli 2025. Adresse: https://www.navcen.uscg.gov/ais-messages.

[3] M. contributors. „:nth-child().“ en, besucht am 21. Aug. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/CSS/:nth-child.

[4] O. Radfelder. „gnuplot.“ de, besucht am 18. Aug. 2025. Adresse: https://informatik.hs-bremerhaven.de/oradfelder/gnuplot.html.

[5] O. Radfelder, *Infrastruktur 2025 Labor Modularisierung*, de. besucht am 4. Aug. 2025. Adresse: https://informatik.hs-bremerhaven.de/oradfelder/infra-2025-lab-mod.pdf.

[6] V. Agafonkin. „Marker.“ en, besucht am 17. Aug. 2025. Adresse: https://leafletjs.com/reference.html#marker.

[7] V. Agafonkin. „Polyline.“ en, besucht am 17. Aug. 2025. Adresse: https://leafletjs.com/reference.html#polyline.

[8] V. Agafonkin. „Layer.“ en, besucht am 17. Aug. 2025. Adresse: https://leafletjs.com/reference.html#layer.

[9] M. contributors. „Element: classList property.“ en, besucht am 17. Aug. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/API/Element/classList.

[10] M. contributors. „Document: getElementsByClassName() method.“ en, besucht am 17. Aug. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementsByClassName.

[11] M. contributors. „String.prototype.includes().“ en, besucht am 17. Aug. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/includes.

[12] M. contributors. „String.prototype.toUpperCase().“ en, besucht am 17. Aug. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/toUpperCase.

[13] M. contributors. „Array.prototype.filter().“ en, besucht am 17. Aug. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter.

[14] M. contributors. „parseFloat().“ en, besucht am 17. Aug. 2025. Adresse: https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/parseFloat.

[15] M. contributors. „String.prototype.replace().“ en, besucht am 17. Aug. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replace.

[16] M. contributors. „String.prototype.split().“ en, besucht am 26. Aug. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/split?utm.

[17] M. contributors. „String.prototype.trim().“ en, besucht am 17. Aug. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/trim.

[18] M. contributors. „Date.“ en, besucht am 17. Aug. 2025. Adresse: https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Date.

[19] M. contributors. „Date.parse().“ en, besucht am 17. Aug. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/parse.

[20] M. contributors. „delete.“ en, besucht am 17. Aug. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete.

[21] M. contributors. „continue.“ en, besucht am 21. Aug. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/continue.

[22] Ogglas, *Comment on: Üsing JavaScript to display a Blob”*, Last accessed 28 August 2025, Apr. 2019. besucht am 28. Aug. 2025. Adresse: https://stackoverflow.com/questions/7650587/using-javascript-to-display-a-blob/44069294#44069294.

[23] M. contributors. „XMLHttpRequest: responseType property.“ en, besucht am 21. Aug. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/responseType.

[24] MariaDB. „TIMESTAMP.“ en, besucht am 28. Juli 2025. Adresse: https://mariadb.com/docs/server/reference/data-types/date-and-time-data-types/timestamp.

[25] MariaDB. „Date and Time Units.“ en, besucht am 31. Juli 2025. Adresse: https://mariadb.com/docs/server/reference/sql-functions/date-time-functions/date-and-time-units.

[26] MariaDB. „CREATE TABLE.“ en, besucht am 28. Juli 2025. Adresse: https://mariadb.com/docs/server/reference/sql-statements/data-definition/create/create-table.

[27] MariaDB. „INSERT ON DUPLICATE KEY UPDATE.“ en, besucht am 30. Juli 2025. Adresse: https://mariadb.com/docs/server/reference/sql-statements/data-manipulation/inserting-loading-data/insert-on-duplicate-key-update.

[28] MariaDB. „IFNULL.“ en, besucht am 30. Juli 2025. Adresse: https://mariadb.com/docs/server/reference/sql-functions/control-flow-functions/ifnull.

[29] E. Merritt. „Key.“ en, besucht am 24. Aug. 2025. Adresse: https://gnuplot.sourceforge.net/docs_4.2/node192.html.

[30] E. Merritt. „histograms.“ en, besucht am 24. Aug. 2025. Adresse: http://www.gnuplot.info/docs_6.1/loc5790.html.

[31] E. Merritt. „xticlabels.“ en, besucht am 24. Aug. 2025. Adresse: http://www.gnuplot.info/docs_6.1/loc10099.html.

Abbildungsverzeichnis

2.1 Darstellung der Ablaufumgebung 8
3.1 Javascript-DOM-Schnittstelle 9
3.2 Leaflet-Klassendiagramm 11
4.1 Queue 14
4.2 Anzahl an Datensätzen über eine Minute hinweg 27
4.3 Anzahl von Elementen in Queue über Zeit 28
7.1 Commit-Verlauf 37
8.1 Laufzeiten der einzelnen Skripte 38
8.2 STEP-Infra-Vergleich 39
8.3 Laufzeiten zu Useranzahlen 40
8.4 Systemauslastung zu Useranzahlen 41

Tabellenverzeichnis

7.1 Beispiele wichtiger Commits 36

Listingverzeichnis

4.1 restore.sh 15
4.2 Enqueue-Widerherstellung 15
4.3 Dequeue-Widerherstellung 16
4.4 Einfügen in die Datenbank 18
4.5 Deployen auf den Teamdocker 23
4.6 Deployment von Crontab-Einträgen 23
4.7 Deployment der Queue 24
4.8 rhodes2.sh 26
8.1 Mögliche JSON-Umsetzung 42
8.2 Mögliche Erweiterung von checkuser 43

Abkürzungsverzeichnis

AIS	Automatic Identification System	7
VM	Virtuelle Maschine	7
CGI	Common Gateway Interface	7
DBMS	Datenbankmanagementsystem	9
UI	User Interface	9
DOM	Document Object Model	9
HTML	Hypertext Markup Language	10
SQL	Structured Query Language	10
HTTP	Hypertext Transfer Protocol	10
LTS	Long Term Support	11
AJAX	Asynchronous JavaScript and XML	12
FIFO	First In, First Out	13
PID	Process Identifikator	13
MMSI	Maritime Mobile Service Identity	17
CSS	Cascading Style Sheet	18
JS	JavaScript	10
SSH	Secure Shell	22
CPU	Central Processing Unit	41
JSON	JavaScript Object Notation	42
AES	Advanced Encryption Standard	43
POSIX	Portable Operating System Interface	43
TUI	Terminal User Interface	52

Anhang

I CGI

I infra-get-positions.sh

```
#!/usr/bin/env bash
source infra-web-lib.sh

getsession

if test -n "$ses_email"; then
  logaction
fi
echo 'content-type: text/plain'
echo

sql_query='SELECT p.mmsi, p.lat, p.lon, p.idx , p.time, s.lst_msg
  From positions AS p, ships AS s
  WHERE p.time > CURRENT_TIMESTAMP - INTERVAL 1 HOUR
  AND s.lst_msg > CURRENT_TIMESTAMP - INTERVAL 5 MINUTE
  AND s.mmsi = p.mmsi
  ORDER BY p.mmsi, p.idx;'
most_recent_idx='0'

while read -r mmsi lat lon idx pos_ts ship_ts; do
  echo "'$mmsi' $lat $lon $pos_ts $ship_ts"
  test "$most_recent_idx" -lt "$idx" && most_recent_idx="$idx"
done <<< "$(mariadb --defaults-file="/var/www/data/.my.cnf" -N <<< "$sql_query" |
↪  sed 's_\t_ _g')"

# We output the most recent index since we use it for updating
echo "$most_recent_idx"
```

II infra-get-table.sh

```
#!/usr/bin/env bash

htmlToTable() {
  mariadb --defaults-file=/var/www/data/.my.cnf -H <<< "SELECT
```

```
      mmsi AS MMSI,
      name AS Name,
      dest AS Ziel,
      status AS Status,
      lst_msg AS 'Letzte Nachricht'
    FROM ships
    WHERE lst_msg > CURRENT_TIMESTAMP - INTERVAL 5 MINUTE;" |
      tidy -q 2>/dev/null
}

cleanHtmlByTidy() {
  local delimitersFound=0
  echo '<table>'
  while read -r line; do
    test "$delimitersFound" -eq 1 && echo "$line"

    # Since we do not want the opening tag and anything that comes after the
    # closing tag, we increment a comparison value to check for the starting and
    # closing cases
    test "$line" = '<table border="1">' \
      -o "$line" = '</table>' &&
      ((++delimitersFound))
  done
}

source infra-web-lib.sh

getsession # sets ses_email
if test -n "$ses_email"; then
  echo -e 'content-type: text/html\n'
  htmlToTable | cleanHtmlByTidy
fi
```

III infra-update-positions.sh

```
#!/usr/bin/env bash

echo 'content-type: text/plain'
echo
```

```

most_recent_idx=${QUERY_STRING#*=}
sql_query="SELECT p.mmsi, p.lat, p.lon, p.idx, p.time, s.lst_msg
  From positions AS p, ships AS s
  WHERE p.time > CURRENT_TIMESTAMP - INTERVAL 1 HOUR
  AND s.lst_msg > CURRENT_TIMESTAMP - INTERVAL 5 MINUTE
  AND s.mmsi = p.mmsi
  AND p.idx > $most_recent_idx
  ORDER BY p.mmsi, p.idx;"

# Output mmsi, lat and lon line-by-line
# as long as the corresponding index is larger than the last known one
while read -r mmsi lat lon idx pos_ts ship_ts; do
  test -z "$mmsi" && break
  echo "'$mmsi' $lat $lon $pos_ts $ship_ts"
  test -n "$idx" && test "$most_recent_idx" -lt "$idx" && most_recent_idx="$idx"
done <<< "$(mariadb --defaults-file=/var/www/data/.my.cnf -N <<< "$sql_query" | sed
↪  's_\t_ _g')"

# We output the most recent index since we use it for updating
echo "$most_recent_idx"
```

II Data

I allow-mariadb-access.sh

```
#!/usr/bin/env bash

# Copy MariaDB config to /var/www/data so that www-data can connect to the
# database
test -e ~/.my.cnf &&
  sudo -u www-data mkdir -p /var/www/data &&
  cat ~/.my.cnf |
  sudo -u www-data bash -c 'cat > /var/www/data/.my.cnf'
```

II cleaner.sh

```
#!/usr/bin/env bash

mariadb <<< 'DELETE
  FROM positions
  WHERE time < CURRENT_TIMESTAMP - INTERVAL 1 WEEK;'
```

III init.sql

```
CREATE TABLE IF NOT EXISTS ships (
  mmsi VARCHAR(10),
  name TEXT,
  dest TEXT,
  status TEXT,
  lst_msg TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (mmsi)
);

CREATE TABLE IF NOT EXISTS positions (
  mmsi VARCHAR(10) REFERENCES ships(mmsi),
  lat FLOAT4,
  lon FLOAT4,
  time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  idx INT AUTO_INCREMENT,
  PRIMARY KEY (idx)
);

CREATE TABLE IF NOT EXISTS users (
  email VARCHAR(320),
  password TEXT,
  PRIMARY KEY (email)
);
```

IV watcher.sh

```
#!/usr/bin/env bash

# Prepare for execution
cd "$(dirname "$0")" || exit 1
rhodes='194.94.217.72'
minute_in_microseconds='60000000'

# Get time delta between last message and now
touch last-msg ncat.pid
read -r msg_timestamp < last-msg
timestamp="${EPOCHREALTIME/[.,]}"
diff="$(( timestamp - msg_timestamp ))"

# Start new ncat process if necessary
if test "$diff" -gt "$minute_in_microseconds"; then
  read -r pid < ncat.pid
  test -n "$pid" &&
    ps "$pid" | grep 'ncat.*worker.sh' &&
    kill "$pid"
  nohup ncat --recv-only -e worker.sh "$rhodes" 8082 &
  echo "$!" > ncat.pid
fi
```

V worker.sh

```
#!/usr/bin/env bash

assignVariables() {
  # This function handles the different AIS message types and assigns variables
  # accordingly. Currently, it handles messages of type 1, 2, 3, 5, 18 and 24
  # and does nothing otherwise.
  local IFS='|'
  unset status_text shipname lon lat destination

  if test "$type" -le "3"; then
    read -r _status status_text _turn _speed _accuracy lon lat _rest <<< "$rest"
```

```
12    elif test "$type" -eq "5"; then
13      read -r _imo _callsign shipname _shiptype _shiptype_text _to_bow _to_stern \
14        _to_port _to_starboard _epfd _eta _epfd_text _month _day _hour _minute \
15        _draught destination _dte <<< "$rest"
16    elif test "$type" -eq "18"; then
17      read -r _speed _accuracy lon lat _rest <<< "$rest"
18    elif test "$type" -eq "24"; then
19      read -r subtype rest <<< "$rest"
20      if test "$subtype" = "A"; then
21        read -r _type shipname <<< "$rest"
22      fi
23    fi
24  }
25
26  echoSqlValue() {
27    # To insert properly, we need to parse empty values as null for SQL. This
28    # function returns empty values as null and wraps non-empty variables in
29    # single quotation marks.
30    test -n "$1" &&
31      echo "'$1'" ||
32      echo 'null'
33  }
34
35  insertIntoDatabase() {
36    # Handle ships first and update instead of inserting when necessary
37    mariadb <<< "INSERT INTO
38      ships (mmsi, name, dest, status)
39      VALUES (
40        '$mmsi',
41        $(echoSqlValue "$shipname"),
42        $(echoSqlValue "$destination"),
43        $(echoSqlValue "$status_text")
44      )
45      ON DUPLICATE KEY UPDATE
46        name = IFNULL(name, VALUES(name)),
47        dest = IFNULL(dest, VALUES(dest)),
48        status = IFNULL(status, VALUES(status)),
49        lst_msg = CURRENT_TIMESTAMP;"
50
51    # Insert sent positions (if any)
52    # Also, filter out incorrect positional data (lat = 91)
53    if test "$type" -le '3' -o "$type" -eq '18' -a "$lat" != '91'; then
```

```
    mariadb <<< "INSERT INTO
      positions (mmsi, lat, lon)
      VALUES ($mmsi, $lat, $lon);"
  fi
}

# Handle all incoming messages by rhodes sequentially
while IFS="|" read -r _timestamp type mmsi rest; do
  assignVariables
  insertIntoDatabase
  echo "${EPOCHREALTIME/[,.]}" > last-msg
done
```

III Deploy

I deploy-hopper.sh

```
#!/usr/bin/env bash

dirname='infra-project'
project_root="$(dirname "$0")/.."

cd "$project_root"
ssh mydocker mkdir -p "$dirname"
tar -cf - . | ssh mydocker bash -c ":
tar -C '$dirname' -xf -
cd '$dirname'
./deploy/deploy-local.sh
"
```

II deploy-local.sh

```
#!/usr/bin/env bash

# -------
#  Utils
```

```
5  # -------
6
7  addGroupWriteableFile() {
8    local filename="$1"
9    local inittext="$2"
10
11   if ! test -e "$filename"; then
12     echo -en "$inittext" > "$filename" &&
13     chmod g+w "$filename" &&
14     return 0
15
16     echo "Failed to make group-writeable file [$filename]" &>2
17     exit 7
18   fi
19 }
20
21 getPath() {
22   local path="$1"
23   test -z "$path" && path="$(realpath .)/../"
24   echo "$path"
25 }
26
27 enterCrontabEntry() {
28   local stamp="$1"
29   local cmd="$2"
30   local path="$(getPath "$3")"
31
32   local entry="$stamp $path$cmd &>/dev/null"
33   local tab="$(crontab -l 2>/dev/null)"
34   grep -F "$entry" >/dev/null <<< "$tab" ||
35     echo -e "$tab\n$entry" | crontab -
36 }
37
38 enterAtrebootEntry() {
39   local file="$HOME/atreboot.sh"
40   local path="$(getPath "$2")"
41   local cmd="$path$1 &>/dev/null"
42
43   # Adds if new file
44   ! test -e "$file" &&
45     echo -e "#!/usr/bin/env bash\n$cmd" > "$file" &&
46     chmod +x "$file" &&
```

```
47      return
48
49    # Adds to file if possible
50    read -r shebang < "$file"
51    grep -q '^#!.*bash' <<< "$shebang" || exit 7
52    grep -q "^$cmd$" "$file" || echo "$cmd" >> "$file"
53  }
54
55  # ---------
56  #  Deploys
57  # ---------
58
59  deployDatabase() {
60    mariadb < ../src/data/init.sql &&
61      ../src/data/allow-mariadb-access.sh ||
62      exit 1
63    enterCrontabEntry '* * * * *' 'src/data/watcher.sh' &&
64      enterCrontabEntry '0 0 * * *' 'src/data/cleaner.sh' &&
65      enterAtrebootEntry 'src/data/allow-mariadb-access.sh' ||
66      exit 2
67  }
68
69  deployWebsite() {
70    local web_dir="/var/www/html/$USER-web/infra-2025"
71    rm -r "$web_dir"/* &>/dev/null
72    mkdir -p "$web_dir" &&
73      cp ../src/www/* "$web_dir" ||
74      exit 3
75  }
76
77  deployCGI() {
78    cp ../src/cgi/* /usr/lib/cgi-bin/ &>/dev/null ||
79      exit 4
80    cp ../src/session/* /usr/lib/cgi-bin/ &>/dev/null ||
81      exit 5
82  }
83
84  deployQueue() {
85    local queue_dir="/data/queue"
86    mkdir -p "$queue_dir" &&
87      chown $USER:www-data "$queue_dir" &&
88      chmod g+ws "$queue_dir" &&
```

```
89      cp ../src/queue/*.sh "$queue_dir" &>/dev/null &&
90      enterCrontabEntry '* * * * *' 'sudo -u workuser /data/queue/watcher.sh' 'true;'
        ↪  &&
91      enterAtrebootEntry '/data/queue/restore.sh' 'true;' ||
92      exit 6
93
94    addGroupWriteableFile "$queue_dir/head" '-\n'
95    addGroupWriteableFile "$queue_dir/tail" '-\n'
96    addGroupWriteableFile "$queue_dir/queue.log" ''
97  }
98
99  deployMonitoring() {
100   local web_dir="/var/www/html/$USER-web/infra-2025"
101   mkdir -p "$web_dir/monitoring" &&
102     cp ../tests/monitoring/www/* "$web_dir/monitoring" &&
103     cp ../tests/monitoring/cgi/* /usr/lib/cgi-bin/ &>/dev/null &&
104     enterCrontabEntry '* * * * *' 'tests/monitoring/data/watcher.sh' ||
105     exit 9
106 }
107
108 cd $(dirname "$0")
109 echo "Now deploying as $USER from $PWD" >&2
110 deployDatabase
111 deployWebsite
112 deployCGI
113 deployQueue
114 deployMonitoring
115 echo "Cool die Anwendung geht :)" >&2
```

III deploy-prod.sh

```
1  #!/usr/bin/env bash
2
3  dirname='infra-project'
4  project_root="$(dirname "$0")/.."
5
6  cd "$project_root"
7  sudo -u infra-2025-b ssh mydocker mkdir -p "$dirname"
8  tar -cf - . | sudo -u infra-2025-b ssh mydocker bash -c ":
```

```
tar -C '$dirname' -xf -
cd '$dirname'
./deploy/deploy-local.sh
"
```

IV Session

I infra-login.sh

```
#!/usr/bin/env bash
source infra-web-lib.sh
parsequerystring email password

if ! checkuser "$get_email" "$get_password"; then
  echo "Content-Type: text/plain"
  echo
  exit 1
fi

createsession "$get_email"
logaction
echo "Content-Type: text/plain"
echo "Set-Cookie: session=$cookie"
echo

echo "true"

# Start email simulation
cd /data/queue
source infra-queue-lib.sh

enqueue "$get_email" "Login" "$get_email logged in successfully"
```

II infra-logout.sh

```
#!/usr/bin/env bash
```

```
source infra-web-lib.sh
getsession # sets ses_email
logaction
if test -n "$ses_email"; then
  echo "Content-Type: text/plain"
  exp="Thu, 01 Jan 1970 00:00:00 GMT"
  echo "Set-Cookie: session=; Expires=$exp"
  echo
  deletesession
fi
```

III infra-web-lib.sh

```
#!/usr/bin/env bash
enable mkdir
enable rm

BASE=/var/www/data

createsession() {
  local email="$1"

  # Removes old session if it exists
  for session in $BASE/session-*; do
    read -r old_email < "$session/email"
    if test "$old_email" = "$email"; then
      rm -rf "$session"
    fi
  done

  # Sets cookie to semi-random 40 char long string
  cookie="$(pwgen 40 1)"
  mkdir -p "$BASE/session-$cookie"
  echo "$email" > "$BASE/session-$cookie/email"
  setsessionvalue begin "${EPOCHREALTIME/[.,]}"
}

deletesession() {
  test -n "$cookie" &&
```

```
27      rm -rf "$BASE/session-$cookie"
28  }
29
30  getsession() {
31    ses_email=""
32    cookie="${HTTP_COOKIE#*session=}"
33    cookie="${cookie%%;*}"
34    if test -n "$cookie" && test -e "$BASE/session-$cookie/email"; then
35      read -r ses_email < "$BASE/session-$cookie/email"
36    fi
37  }
38
39  logaction() {
40    local ts="${EPOCHREALTIME/[.,]}"
41    local action=${0##*/}
42    local log="$ts $cookie $ses_email $action ${@}"
43    echo "$log" >> $BASE/web.log
44  }
45
46  parsequerystring() {
47    local query kv k v i
48    test -z "$QUERY_STRING" && return
49
50    # Since we deconstruct the string in parts up until the first appearing
51    # ampersand, we need to append one at the end to ensure proper handling of the
52    # last key value pair
53    query="$QUERY_STRING&"
54    while test -n "$query"; do
55      kv="${query%%&*}"
56      k="${kv%%=*}"
57      v="${kv#*=}"
58      for i in "$@"; do
59        if test "$i" = "$k"; then
60          printf -v "get_$i" "$v"
61        fi
62      done
63      query="${query#*&}"
64    done
65  }
66
67  checkuser() {
68    local input_email="$1"
```

```
69    local input_password="$2"
70
71    if test -n "$(mariadb --defaults-file=/var/www/data/.my.cnf <<< "SELECT * FROM
      ↪  users
72    WHERE email='$input_email' AND password='$input_password';")"; then
73      ses_email="$input_email"
74      return 0
75    fi
76    return 1
77  }
```

V Tests

I all-scripts

I.1 do-all.sh

```
1   #!/usr/bin/env bash
2
3   create_graphable_data() {
4     # Get all sums
5     while read -r script microseconds; do
6       (($script += $microseconds))
7     done < 'result.dat'
8
9     # Calc avg and pipe into file
10    sort 'result.dat' | cut -d ' ' -f 1 | uniq -c | sed 's/^ *//' |
11      while read -r count script; do
12        avg="$(bc -l <<< "${!script}/($count*1000) + 0.5")"
13        avg="${avg%.*}"
14        echo -n "$script $avg "
15      done >> avg.log
16    echo >> avg.log
17  }
18
19  ./test-scripts.sh $1
20  test -e avg.log && rm avg.log
21  create_graphable_data
```

```
gnuplot graph.gpi && convert graph.svg six:- && echo
```

I.2 graph.gpi

```
set terminal pngcairo size 2048,1024 font 'Verdana,32'
set output 'graph.png'

set title 'Durchschnittliche Laufzeit der Skripte im Vergleich'
set key right top font ', 21'
set xlabel 'Skripte'
set ylabel 'Durchschnittliche Laufzeit in Millisekunden'
set tics scale 0
unset xtics

# Border
set style line 80 linetype 1 linecolor rgb 'black' linewidth 4
set border 3 back linestyle 80

# Background
set style line 81 linetype 1 linecolor rgb '#808080' linewidth 2
set grid back linestyle 81

set boxwidth 0.95
set style fill solid 1.0

plot \
  'avg.log' using (0):2  with boxes lc rgb '#f94144' title 'get-positions', \
  ''        using (1):20 with boxes lc rgb '#f3722c' title 'update-positions', \
  ''        using (2):4  with boxes lc rgb '#f8961e' title 'get-table', \
  ''        using (3):12 with boxes lc rgb '#f9844a' title 'login', \
  ''        using (4):14 with boxes lc rgb '#f9c74f' title 'logout', \
  ''        using (5):10 with boxes lc rgb '#90be6d' title 'worker.sh', \
  ''        using (6):6  with boxes lc rgb '#43aa8b' title 'infra-dequeue', \
  ''        using (7):8  with boxes lc rgb '#4d908e' title 'infra-enqueue', \
  ''        using (8):16 with boxes lc rgb '#577590' title 'step-dequeue', \
  ''        using (9):18 with boxes lc rgb '#277da1' title 'step-enqueue'
```

I.3 test-scripts.sh

```
#!/usr/bin/env bash

jar="user-$$.jar"
path="$(dirname "$0")"

measure() {
  for _ in $(seq ${3:-1}); do
    s="${EPOCHREALTIME/[,.]}"
    eval $2
    e="${EPOCHREALTIME/[,.]}"
    log "$1" $((e-s))
  done
}

log() {
  process="$1"
  time="$2"
  echo "$process $time" >> $path/result.dat
}

test_website() {
  for _ in $(seq ${1:-1}); do
    measure 'login' "curl  -s  -b '$jar' -c '$jar' 'https://informatik.hs-bremerhaven.de/docker-lucfocken-web/cgi-bin/infra-login.sh?email=luca&password=1234'" >/dev/null
    measure 'get_table' "curl  -s  -b '$jar' -c '$jar' 'https://informatik.hs-bremerhaven.de/docker-lucfocken-web/cgi-bin/infra-get-table.sh'" >/dev/null
    last_index="$(measure 'get_position' "curl  -s  -b '$jar' -c '$jar' 'https://informatik.hs-bremerhaven.de/docker-lucfocken-web/cgi-bin/infra-get-positions.sh'" | tail -n1)"
    measure 'update_position' "curl  -s  -b '$jar' -c '$jar' 'https://informatik.hs-bremerhaven.de/docker-lucfocken-web/cgi-bin/infra-update-positions.sh?$last_index'" >/dev/null
    measure 'logout' "curl -s  -b '$jar' -c '$jar' 'https://informatik.hs-bremerhaven.de/docker-lucfocken-web/cgi-bin/infra-logout.sh'"
    rm "$jar"
  done
}

```

```
delete_queue() {
  rm tail
  rm head
  rm enqueue.order
  rm dequeue.order
  rm queue.log
  rm email.log
  rm $1-queue-lib.sh
}

setup_infra_queue() {
  cp ../../src/queue/$1 .
  echo "-" > tail
  echo "-" > head
}

main() {
  measure "insert_into_database" "echo '1970-01-01
  ↪  20:35:34|3|1|5|Moored|nan|0|false|20|20|0|511|31|0|Not available|false|0' |
  ↪  ../../src/data/worker.sh" $1
  rm last-msg

  test_website $1

  setup_infra_queue "infra-queue-lib.sh"
  measure 'infra_enqueue' 'bash -c "source infra-queue-lib.sh; enqueue 1 1 1"' $1
  measure 'infra_dequeue' 'bash -c "source infra-queue-lib.sh; dequeue"' $1
  delete_queue "infra"

  setup_infra_queue "step-queue-lib.sh"
  measure 'step_enqueue' 'bash -c "source step-queue-lib.sh; enqueue 1 1 1"' $1
  measure 'step_dequeue' 'bash -c "source step-queue-lib.sh; dequeue"' $1
  delete_queue "step"
}

test -e result.dat && rm result.dat
main $1
```

II database-test

II.1 main.gp

```
set terminal pngcairo size 2048,1024 font 'Verdana,32'
set output 'main.png'

set key right bottom
set xlabel 'Sekunden nach Messbeginn'
set ylabel 'Anzahl Datensätze'
set tics scale 0

# Border
set style line 80 linetype 1 linecolor rgb 'black' linewidth 4
set border 3 back linestyle 80

# Background
set style line 81 linetype 1 linecolor rgb '#808080' linewidth 2
set grid back linestyle 81

# Lines
set style line 82 linecolor rgb '#ffbe0b' linewidth 3 pointtype 7 pointsize 2
set style line 83 linecolor rgb '#fb5607' linewidth 3 pointtype 7 pointsize 2
set style line 84 linecolor rgb '#ff006e' linewidth 3 pointtype 7 pointsize 2
set style line 85 linecolor rgb '#8338ec' linewidth 3 pointtype 7 pointsize 2
set style line 86 linecolor rgb '#3a86ff' linewidth 3 pointtype 7 pointsize 2

plot 'main.dat' using 1:2 with linespoints linestyle 82 title '100 Schiffe', \
     'main.dat' using 1:3 with linespoints linestyle 83 title '200 Schiffe', \
     'main.dat' using 1:4 with linespoints linestyle 84 title '300 Schiffe', \
     'main.dat' using 1:5 with linespoints linestyle 85 title '400 Schiffe', \
     'main.dat' using 1:6 with linespoints linestyle 86 title '500 Schiffe'
```

II.2 main.sh

```
#!/usr/bin/env bash

cd "$(dirname "$0")"
```

```

# Start rhodes2 and ncat process and save pids
../rhodes2.sh "$1" &
ncat -e '../../src/data/worker.sh' localhost 1234 &
echo "$!" > ncat.pid

# Measure the count of positions every second
mariadb <<< 'DELETE FROM positions;'
echo -n '' > main.dat
for i in $(seq ${2:-60}); do
  echo -n "$i "
  mariadb -N <<< 'SELECT COUNT(*) FROM positions;'
  sleep 1
done

# Kill ongoing processes
kill "$(cat ncat.pid)"
kill "$(cat ncat2.pid)"
worker_pid="$(ps -ef | grep 'data/worker.sh' | grep -v grep |
  sed 's/^[^1-9]*//g' | sed 's/ .*//g')"
test -n "$worker_pid" && kill "$worker_pid"

# Create png
#gnuplot main.gp

# Remove unecessary files
rm *.pid
rm last-msg
```

II.3 measure.sh

```
#!/usr/bin/env bash

cd "$(dirname "$0")"

rm 'main.dat'

echo 'Start measuring...'

```

```
for i in {1..5}; do
  ./main.sh "$((i * 100))" 60 >> test-$i.dat
done

echo 'Process data...'

for i in {1..60}; do
  v1="$(head -n $i test-1.dat | tail -n 1)"
  v2="$(head -n $i test-2.dat | tail -n 1 | cut -d ' ' -f 2)"
  v3="$(head -n $i test-3.dat | tail -n 1 | cut -d ' ' -f 2)"
  v4="$(head -n $i test-4.dat | tail -n 1 | cut -d ' ' -f 2)"
  v5="$(head -n $i test-5.dat | tail -n 1 | cut -d ' ' -f 2)"

  echo "$v1 $v2 $v3 $v4 $v5" >> main.dat
done

rm test-*.dat

echo 'Creating plot...'
gnuplot main.gp

echo 'Finished :)'
```

III monitoring

III.1 app.js

```
function buildPlot() {
  let xhr = new XMLHttpRequest()
  xhr.onreadystatechange = function () {
    if ( xhr.readyState == 4 && xhr.status == 200 ) {
      image.src = URL.createObjectURL(xhr.response)
    }
  }
  xhr.open("GET", "../../cgi-bin/build-stats-now.sh")
  xhr.responseType = "blob"
  xhr.send()
}

```

```
function init() {
  image = document.getElementById("stats-now")
  buildPlot()
  setInterval(buildPlot, 1000)
}

window.onload = init
```

III.2 build-stats-now.sh

```
#!/usr/bin/env bash
echo 'content-type: image/png'
echo

gnuplot stats-now.gp
```

III.3 get-stats-now.sh

```
#!/usr/bin/env bash

get_sessions() {
  local sessions="$(echo /var/www/data/session-*)"

  if test "$sessions" = "/var/www/data/session-*"; then
    count="0"
  else
    count="$(echo $sessions | sed 's_ _\n_g' | wc -l)"
  fi

  echo -n "$count "
}

get_new_positions() {
  local positions_count_file='/var/www/data/positions_count'
  ! test -f "$positions_count_file" && echo '0' > "$positions_count_file"
  positions_count_now="$(mariadb --defaults-file=/var/www/data/.my.cnf -N <<< 'SELECT
  ↪  COUNT(*) FROM positions;')"
```

```
  read -r positions_count_old < "$positions_count_file"
  echo "$positions_count_now" > "$positions_count_file"

  echo -n "$((positions_count_now - positions_count_old)) "
}

get_active_ships() {
  active_ships_count="$(mariadb --defaults-file=/var/www/data/.my.cnf -N <<< 'SELECT
  ↪  COUNT(*) FROM ships WHERE lst_msg > CURRENT_TIMESTAMP - INTERVAL 5 MINUTE;')"
  echo "$active_ships_count"
}

data_file='/var/www/data/stats_now.dat'
touch "$data_file"

while true; do
  data_line_count="$(wc -l "$data_file" | cut -d ' ' -f 1)"
  test "$data_line_count" -gt "60" &&
   tail -n 59 "$data_file" > "$data_file.tmp" && mv "$data_file.tmp" "$data_file"

  get_sessions >> "$data_file" &&
    get_new_positions >> "$data_file" &&
    get_active_ships >> "$data_file"

  sleep 1
done
```

III.4 index.html

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Monitoring</title>
    <link rel="stylesheet" href="main.css" />
    <script src="app.js"></script>
  </head>
  <body>
```

```
    <h1>Monitoring</h1>
    <img id="stats-now" alt="Aktueller Zustand">
  </body>
</html>
```

III.5 main.css

```
body {
  text-align: center;
  justify-content: center;
  font-family: Verdana, sans-serif;
}

#stats-now {
  width: 80%;
  height: auto;
}
```

III.6 stats-now.gp

```
set terminal pngcairo size 2048,1024 font 'Verdana,24'

set xrange [1:60]

set bmargin 6
set key bmargin center bottom

set ytics 5
set xlabel 'Verlauf über Sekunden'
unset xtics

set style line 80 \
          linetype 1 \
          linecolor rgb '#ffbe0b' \
          linewidth 3

set style line 81 \
```

```
18          linetype 1 \
19          linecolor rgb '#ff006e' \
20          linewidth 3
21
22  set style line 82 \
23          linetype 1 \
24          linecolor rgb '#3a86ff' \
25          linewidth 3
26
27  plot '/var/www/data/stats_now.dat' using 1 with lines linestyle 80 title 'Anzahl
    ↪  laufender Sessions', \
28       '/var/www/data/stats_now.dat' using 2 with lines linestyle 81 title 'Anzahl
         ↪  neuer Positionsmeldungen', \
29       '/var/www/data/stats_now.dat' using 3 with lines linestyle 82 title 'Anzahl
         ↪  aktiver Schiffe'
```

III.7 watcher.sh

```
1   #!/usr/bin/env bash
2
3   cd "$(dirname $0)"
4
5   data_file='/var/www/data/stats_now.dat'
6   ts_data_file="$(date -r "$data_file"  +'%s')"
7   now_in_seconds="$(( ${EPOCHREALTIME/[.,]} / 1000000 ))"
8   diff="$((now_in_seconds - ts_data_file))"
9
10  if test "$diff" -gt "10"; then
11    pid_file='get-stats-now.pid'
12    test -f "$pid_file" && rm "$pid_file"
13    sudo -u www-data nohup ./get-stats-now.sh &
14    echo "$!" > "$pid_file"
15  fi
```

IV queue-test

IV.1 main.gpi

```
set terminal pngcairo size 2048,1024 font 'Verdana,32'
set output 'main.png'

set xlabel 'Sekunden nach Messbeginn'
set ylabel 'Anzahl Knoten'
set tics scale 0

set style line 80 \
          linetype 1 \
          linecolor rgb 'black' \
          linewidth 4
set border 3 back linestyle 80

set style line 81 \
          linetype 1 \
          linecolor rgb '#808080' \
          linewidth 2
set grid back linestyle 81

set style line 82 \
          linecolor rgb '#ffbe0b' \
          pointtype 7 \
          pointsize 1

set style line 83 \
          linecolor rgb '#ff006e' \
          pointtype 7 \
          pointsize 1

set style line 84 \
          linecolor rgb '#3a86ff' \
          pointtype 7 \
          pointsize 1

plot 'queue1.dat' using 1:2 with points linestyle 82 title 'Dequeue-Prozesse: 1' , \
     'queue2.dat' using 1:2 with points linestyle 83 title 'Dequeue-Prozesse: 2' , \
```

```
37      'queue3.dat' using 1:2 with points linestyle 84 title 'Dequeue-Prozesse: 3'
```

IV.2 queue-test-lib.sh

```
1  #!/usr/bin/env bash
2
3  enablePlotDebug="true"
4
5  login() {
6    local prefix="$(mktemp -d)"
7    local jar="$prefix/ck$$.jar"
8    local base="https://informatik.hs-bremerhaven.de/$USER-web/cgi-bin"
9    local session_path="/var/www/data"
10   curl -s -b "$jar" -c "$jar" "$base/infra-login.sh?email=[1-10]&password=1"
     ↪  >/dev/null
11   curl -s -b "$jar" -c "$jar" "$base/infra-logout.sh" >/dev/null
12
13   rm -rf "$prefix"
14 }
15
16 queueLogins() {
17   for i in $(seq 1 "$1"); do
18    login &
19   done
20 }
21
22 # Collect data from queue to plot data
23 plotQueue() {
24   local path="/data/queue"
25   local second="0"
26   local found="0"
27   test -e "$path/queue.dat" && rm "$path/queue.dat"
28
29   # Debug
30   if test "$enablePlotDebug" = "true"; then
31     echo -e "----------\nDebug: plotQueue"
32     echo -e "Second\tCount"
33   fi
34
```

```
35    while true; do
36      # Check for nodes
37      local count="$(ls $path/node-* 2>/dev/null | wc -l)"
38      echo "$second $count" >> "$path/queue.dat"
39
40      # Debug
41      test "$enablePlotDebug" = "true" &&
42       echo -e "$second\t$count"
43
44      ((++second))
45      # Check if node was found
46      if test "$found" -eq "0" -a "$count" -ne "0"; then
47        found="1"
48      fi
49      # If a node was found at one point and there are no nodes left finish
50      # collecting data and plot
51      if test "$found" -eq "1" -a "$count" -eq "0"; then
52        break
53      fi
54      sleep 1
55    done
56
57    # Debug
58    test "$enablePlotDebug" = "true" &&
59      echo "----------"
60
61    gnuplot main.gp
62    cp main.png /var/www/html/$USER-web/queue.png
63  }
64
65  # Check log if every login was queued
66  checkLog() {
67    cd /data/queue/
68    # Wait until there are no nodes left
69    while ls node-* >/dev/null 2>/dev/null; do
70      sleep 1
71    done
72    # Sort and count all of email.log to check if every account logged in the same
      ↪  amount
73    result="$(cut -d" " -f2 email.log | sort | uniq -c | sed "s_^ *__g" | cut -d" "
      ↪  -f1 | uniq | wc -l)"
74    if test "$result" -ne "1"; then
```

```
75      echo "Found error in /data/queue/email.log!"
76    else
77      echo "Found no error in /data/queue/email.log"
78    fi
79  }
80
81  checkQueueOrder() {
82    # Check if there is a difference between enqueue.order an dequeue.order
83    local path="/data/queue"
84    if test -z "$(diff "$path/enqueue.order" "$path/dequeue.order")"; then
85      echo "Enqueue and dequeue order is correct"
86    else
87      echo "Enqueue and dequeue order is not correct!"
88    fi
89  }
90
91  # Delete Logs
92  deleteLogs() {
93    test -e "/var/www/data/web.log" &&
94      sudo -u www-data rm "/var/www/data/web.log"
95    test -e "/data/queue/email.log" &&
96      sudo -u workuser rm "/data/queue/email.log"
97    test -e "/data/queue/enqueue.order" &&
98      sudo -u workuser rm -f "/data/queue/enqueue.order"
99    test -e "/data/queue/dequeue.order" &&
100     sudo -u workuser rm -f "/data/queue/dequeue.order"
101 }
102
103 # Restart queue-worker while there are nodes
104 restartDequeue() {
105   while ls /data/queue/node-* >/dev/null 2>/dev/null; do
106     pid="$(cat /data/queue/worker.pid)"
107     sudo -u workuser kill "$pid"
108     sudo -u workuser /data/queue/watcher.sh
109     sleep 1
110   done
111 }
112
113 main () {
114   deleteLogs
115   plotQueue &
116   queueLogins 10 &
```

```

  # For more data to plot
  #sleep 120
  #queueLogins 50 &
  #sleep 300
  #queueLogins 300 &
  #sleep 300
  #queueLogins 400 &
  #sleep 300
  #queueLogins 500 &

  wait
  checkLog
  echo "copy main.png to /var/www/html/$USER-web/queue.png"
  checkQueueOrder
}
```

IV.3 restart-queue.sh

```
#!/usr/bin/env bash

source queue-test-lib.sh

deleteLogs
queueLogins 10 &
wait

plotQueue &
restartDequeue &

# To check if process is being restarted
# watch "ps -ef | grep '^workuser .*bash ./worker.sh'"

wait
checkLog &
checkQueueOrder
```

IV.4 test-queue.sh

```
#!/usr/bin/env bash

source queue-test-lib.sh
main
```

IV.5 users.sh

```
query=""
for i in {1..1000}; do
  query="${query},($i,1)"
done
query="INSERT INTO users VALUES ${query#,};"
mariadb -e "$query"
```

V rhodes2.sh

```
#!/usr/bin/env bash

cd "$(dirname "$0")"

echo -n '' > mmsi.txt

for i in $(seq ${1:-400}); do
  echo "$i" >> mmsi.txt
done

lon_modifier='551'

while true; do
  lat_modifier='100'
  while read -r mmsi; do
    echo "TIMESTAMP|3|$mmsi|STATUS|STATUS_TEXT|TURN|SPEED|ACCURACY|8.$lon_modifier|5
    ↪  3.5$lat_modifier|REST"
    (( ++lat_modifier ))
```

```
  done < mmsi.txt
  (( ++lon_modifier ))
done | ncat -lk 1234 &
echo "$!" > database-test/ncat2.pid
```

VI simulate-user

VI.1 do-all.sh

```
#!/usr/bin/env bash

clear_script_data() {
  while read -r script _; do
    printf -v "$script" 0
  done < 'times.log'
}

create_graphable_data() {
  # Get all sums
  while read -r script microseconds; do
    (($script += $microseconds))
  done < 'times.log'

  # Calc avg and pipe into file
  echo -n "$iter " >> avg.log
  sort 'times.log' | cut -d ' ' -f 1 | uniq -c | sed 's/^ *//' |
    while read -r count script; do
      avg="$(bc -l <<< "${!script}/($count*1000) + 0.5")"
      avg="${avg%.*}"
      echo -n "$script $avg "
    done >> avg.log
  echo >> avg.log
}

# Validate usage
cd "$(dirname "$0")"
test -z "$3" &&
  echo 'Please enter three numerical arguments:' >&2 &&
```

```
30   echo './multi-user.sh [number of user #1] [number of user #2] [number of user #3]'
     ↪  >&2 &&
31   exit 1
32 test -f '../times.log' && rm '../times.log'
33 test -f 'avg.log' && rm 'avg.log'
34
35 # Do measurements
36 for iter in "$1" "$2" "$3"; do
37   ../multi-user.sh "$iter" 1 1
38   cp '../times.log' .
39   clear_script_data
40   create_graphable_data
41 done
42
43 gnuplot svg.gpi && convert graph.png six:- && echo
```

VI.2 multi-user.sh

```
1  #!/usr/bin/env bash
2
3  cd "$(dirname "$0")"
4  test -z "$3" &&
5    echo 'Please enter three numerical arguments:' >&2 &&
6    echo './multi-user.sh [number of users] [number of map updates] [number of table
     ↪  retrievals]' >&2 &&
7    exit 1
8  for _ in $(seq "$1"); do
9    ./single-user.sh "$2" "$3" &
10 done
11 wait
```

VI.3 single-user.sh

```
1  #!/usr/bin/env bash
2
3  cd "$(dirname "$0")"
4  jar="user-$$.jar"
```

```
5  base="https://informatik.hs-bremerhaven.de/docker-$USER-web/cgi-bin"
6  test -z "$2" &&
7    echo 'Please enter two numerical arguments:' >&2 &&
8    echo './single-user.sh [number of map updates] [number of table retrievals]' >&2 &&
9    exit 2
10
11 log() {
12   local script="$1"
13   local timetaken="$2"
14   echo "$script $timetaken" >> 'times.log'
15 }
16
17 login() {
18   local credentials="$1"
19   local starttime="${EPOCHREALTIME/[,.]}"
20   curl -s -b "$jar" -c "$jar" "$base/infra-login.sh?$credentials"
21   local endtime="${EPOCHREALTIME/[,.]}"
22   log 'login' "$((endtime - starttime))"
23 }
24
25 map_updates() {
26   local update_times="$2"
27
28   local starttime="${EPOCHREALTIME/[,.]}"
29   idx="$(curl -s -b "$jar" -c "$jar" "$base/infra-update-positions.sh?lstMsg=$1" |
     ↪  tail -1)"
30   local endtime="${EPOCHREALTIME/[,.]}"
31   log 'update_positions' "$((endtime - starttime))"
32
33   sleep 1
34   test $((++do_map)) -eq "$update_times" || map_updates "$idx" "$update_times"
35 }
36
37 map() {
38   local update_times="$1"
39
40   local starttime="${EPOCHREALTIME/[,.]}"
41   latest_map_index="$(curl -s -b "$jar" -c "$jar" "$base/infra-get-positions.sh" |
     ↪  tail -1)"
42   local endtime="${EPOCHREALTIME/[,.]}"
43   log 'get_positions' "$((endtime - starttime))"
44
```

```
  sleep 1
  map_updates "$latest_map_index" "$update_times"
}

table() {
  local update_times="$1"

  local starttime="${EPOCHREALTIME/[,.]}"
  curl -s -b "$jar" -c "$jar" "$base/infra-get-table.sh" &>/dev/null
  local endtime="${EPOCHREALTIME/[,.]}"
  log 'get_table' "$((endtime - starttime))"

  sleep 1
  test $((++do_table)) -eq "$update_times" || table "$update_times"
}

# Logout is a reserved keyword
unlogin() {
  local starttime="${EPOCHREALTIME/[,.]}"
  curl -s -b "$jar" -c "$jar" "$base/infra-logout.sh"
  local endtime="${EPOCHREALTIME/[,.]}"
  log 'logout' "$((endtime - starttime))"
}

# Login
login_successful="$(login 'email=a&password=b')"
test "$login_successful" != 'true' && echo 'Login unsuccessful' >&2 && exit 1

# Simulate app usage
map "$1"
table "$2"

# Logout
unlogin || echo 'Logout failed' >&2
rm "$jar"
```

VI.4 png.gpi

```
set terminal pngcairo size 2048,1024 font 'Verdana,32'
```

```
2  set output 'graph.png'
3
4  set title 'Laufzeit der Skripte im Vergleich'
5  set key left top font ', 24'
6  set xlabel 'Anzahl gleichzeitiger User'
7  set ylabel 'Laufzeit in Millisekunden'
8  set tics scale 0
9  set xrange [-0.5:2.5]
10 set yrange [0:1300]
11
12 # Border
13 set style line 80 linetype 1 linecolor rgb 'black' linewidth 4
14 set border 3 back linestyle 80
15
16 # Background
17 set style line 81 linetype 1 linecolor rgb '#808080' linewidth 2
18 set grid back linestyle 81
19
20 set boxwidth 0.95 relative
21 set style fill solid 1.0
22
23 plot 'avg.log' using  3              with histogram lc rgb '#ffbe0b' title
   ↪  'get-positions', \
24      'avg.log' using  5              with histogram lc rgb '#fb5607' title
        ↪  'get-table', \
25      'avg.log' using  7:xticlabels(1) with histogram lc rgb '#ff006e' title 'login',
        ↪  \
26      'avg.log' using  9              with histogram lc rgb '#8338ec' title
        ↪  'logout', \
27      'avg.log' using 11              with histogram lc rgb '#3a86ff' title
        ↪  'update-positions'
```

VII step-vs-infra

VII.1 do-all.sh

```
1  #!/usr/bin/env bash
2
3  clear_script_data() {
```

```
  while read -r script _; do
    printf -v "$script" 0
  done < 'times.log'
}

create_graphable_data() {
  # Get all sums
  while read -r script microseconds; do
    (($script += $microseconds))
  done < 'times.log'

  # Calc avg and pipe into file
  echo -n "$iter " >> avg.log
  sort 'times.log' | cut -d ' ' -f 1 | uniq -c | sed 's/^ *//' |
    while read -r count script; do
      avg="$(bc -l <<< "${!script}/($count*1000) + 0.5")"
      avg="${avg%.*}"
      echo -n "$script $avg "
    done >> avg.log
  echo >> avg.log
}

# Validate usage
cd "$(dirname "$0")"
test -z "$3" &&
  echo 'Please enter three numerical arguments:' >&2 &&
  echo './multi-user.sh [number of user #1] [number of user #2] [number of user #3]'
  ↪  >&2 &&
  exit 1
test -f '../../simulate-user/times.log' && rm '../../simulate-user/times.log'
test -f 'avg.log' && rm 'avg.log'

# Do measurements
for iter in $@; do
  ../../simulate-user/multi-user.sh "$iter" 1 1
  cp '../../simulate-user/times.log' .
  clear_script_data
  create_graphable_data
done

gnuplot svg.gpi && convert graph.png six:- && echo
```

VII.2 main.sh

```
#!/usr/bin/env bash

cd "$(dirname "$0")"
numbers='1 5 20 50 100'

# Infra
./time-visualization/do-all.sh $numbers >/dev/null
cp ./time-visualization/graph.png infra.png
cut -d ' ' -f 1,3 ./time-visualization/avg.log > temp.log

# Step
scp step-get-positions.sh mydocker:/usr/lib/cgi-bin/infra-get-positions.sh
./time-visualization/do-all.sh $numbers >/dev/null
cp ./time-visualization/graph.png step.png
scp ../../src/cgi/infra-get-positions.sh
↪  mydocker:/usr/lib/cgi-bin/infra-get-positions.sh

# Merge log files
test -f comp.log && rm comp.log
while read -r left; do
  ((++count))
  read -r right <<< "$(cut -d ' ' -f 3 ./time-visualization/avg.log | head -$count |
  ↪  tail -1)"
  echo "$left $right" >> comp.log
done < temp.log

# Make final graph =)
cd time-visualization/
gnuplot svg.gpi &&
  convert graph.png six:- &&
  echo
```

VII.3 step-get-positions.sh

```
#!/usr/bin/env bash
BASE='/var/www/data'
```

```
3
4  # Get session
5  ses_email=''
6  cookie="$(echo "$HTTP_COOKIE" | sed 's/^.*session=//' | sed 's/;.*$//')"
7  if test -n "$cookie" && test -e "$BASE/session-$cookie/email"; then
8    ses_email="$(cat "$BASE/session-$cookie/email")"
9  fi
10
11 if test -n "$ses_email"; then
12   # Log action
13   ts="$(date +'%s%N')"
14   action="$(echo "$0" | sed 's_^.*/__')"
15   log="$ts $cookie $ses_email $action"
16   echo "$log" >> $BASE/web.log
17 fi
18 echo 'content-type: text/plain'
19 echo
20
21 sql_query='SELECT p.mmsi, p.lat, p.lon, p.idx , p.time, s.lst_msg
22   From positions AS p, ships AS s
23   WHERE p.time > CURRENT_TIMESTAMP - INTERVAL 1 HOUR
24   AND s.lst_msg > CURRENT_TIMESTAMP - INTERVAL 5 MINUTE
25   AND s.mmsi = p.mmsi
26   ORDER BY p.mmsi, p.idx;'
27 most_recent_idx='0'
28
29 while read -r mmsi lat lon idx pos_ts ship_ts; do
30   echo "'$mmsi' $lat $lon $pos_ts $ship_ts"
31   if test "$most_recent_idx" -lt "$idx"; then
32     most_recent_idx="$idx"
33   fi
34 done <<< "$(echo "$sql_query" | mariadb --defaults-file="/var/www/data/.my.cnf" -N |
   ↪  sed 's_\t_ _g')"
35
36 # We output the most recent index since we use it for updating
37 echo "$most_recent_idx"
```

VII.4 png.gpi

```
set terminal pngcairo size 2048,1024 font 'Verdana,32'
set output 'graph.png'

set title 'Durchschnittliche Laufzeit von get-positions.sh'
set key left top
set xlabel 'Anzahl gleichzeitiger User'
set ylabel 'Durchschnittliche Laufzeit in Millisekunden'
set tics scale 0
set xrange [-0.5:4.5]

# Border
set style line 80 linetype 1 linecolor rgb 'black' linewidth 4
set border 3 back linestyle 80

# Background
set style line 81 linetype 1 linecolor rgb '#808080' linewidth 2
set grid back linestyle 81

set boxwidth 0.95 relative
set style fill solid 1.0

plot '../comp.log' using 2:xticlabels(1) with histogram lc rgb '#ffbe0b' title
↪  'Infra-Version', \
     '../comp.log' using 3:xticlabels(1) with histogram lc rgb '#3a86ff' title
     ↪  'STEP-Version'
```

VIII stress-test

VIII.1 do-all.sh

```
#!/usr/bin/env bash

cd "$(dirname "$0")"

test -z "$1" &&
  echo 'Missing argument: [max amount user]' >&2 &&
```

```
  exit 1
data_file='avg_cpu_usage.dat'

test -f "$data_file" && rm "$data_file"

for i in $(seq "$1"); do
  echo -n "$i " >> "$data_file"
  ./get-n-user-avg-cpu-usage.sh "$i" >> "$data_file"
  echo "Finished user $i of $1"
done

gnuplot main.gp && convert usage.png six:- && echo
```

VIII.2 get-n-user-avg-cpu-usage.sh

```
#!/usr/bin/env bash

cd "$(dirname "$0")"

test -z "$1" &&
  echo 'Missing argument: [amount user]' >&2 &&
  exit 1

dat_file="$1-user-cpu-usage.dat"
pid_file='cpu_usage.pid'

# Get cpu usage
test -f "$dat_file" && rm "$dat_file"
while true; do
  cpu_usage="$(hbv_dockerstats | tail -n 1 | cut -d ' ' -f 7)"
  rounded="${cpu_usage%.*}"
  echo "$rounded" >> "$dat_file"
done &
echo "$!" > "$pid_file"

# Starts multiple user simulation
../simulate-user/multi-user.sh "$1" '15' '10'

read -r pid < "$pid_file"
```

```
kill "$pid"

# Calculates avg cpu usage
while read -r usage; do
  ((usage_sum+=$usage))
done < "$dat_file"
count="$(wc -l < "$dat_file")"
avg="$(bc -l <<< "$usage_sum / $count + 0.5")"
avg="${avg%.*}"

echo "$avg"
rm "$dat_file"
```

VIII.3 main.gp

```
set terminal pngcairo size 2048,1024 font 'Verdana,32'
set output 'usage.png'

set xlabel 'Anzahl gleichzeitiger User'
set ylabel 'Durchschnittliche CPU-Auslastung bis 400%'
set yrange [0:400]
set tics scale 0

# Border
set style line 80 linetype 1 linecolor rgb 'black' linewidth 4
set border 3 back linestyle 80

# Background
set style line 81 linetype 1 linecolor rgb '#808080' linewidth 2
set grid back linestyle 81

# Lines
set style line 82 linecolor rgb '#1c86ee' linewidth 5

plot 'avg_cpu_usage.dat' using 1:2 with lines linestyle 82 notitle
```

IX unit-tests.sh

```
#!/usr/bin/env bash

test_cleaner() {
  mariadb <<< "INSERT INTO positions (mmsi, lat, lon, time)
         VALUES ((SELECT mmsi FROM ships LIMIT 1), 0, 0, CURRENT_TIMESTAMP - INTERVAL
         ↪  8 DAY);"
  local path="$(dirname "$0")/../src/data"
  $path/cleaner.sh
  result="$(mariadb <<< "SELECT * FROM positions WHERE time < CURRENT_DATE - INTERVAL
  ↪  7 DAY;")"
  if test -z $result; then
    return 0
  fi 2>/dev/null
  echo 'cleaner check failed!' >&2
  return 1
}

test_watcher(){
  local data_path="$(dirname "$0")/../src/data"
  # This will only work if ncat was started through the watcher
  read -r pid < "${data_path}/ncat.pid"
  test -n "$pid" &&
    ps "$pid" | grep -q 'ncat.*worker.sh' &&
    kill "$pid"

  echo "0" > "${data_path}/last-msg"
  "${data_path}/watcher.sh" &> /dev/null

  # We need to wait for the watcher to finish starting up the background process
  sleep 1

  read -r pid < "${data_path}/ncat.pid"
  ps "$pid" | grep -q 'ncat.*worker.sh' && return 0
  echo 'watcher check failed!' >&2
  return 1
}

test_session(){
  local prefix="$(mktemp -d)"
```

```
38    local jar="$prefix/ck$$.jar"
39    local base="https://informatik.hs-bremerhaven.de/docker-lucfocken-web/cgi-bin"
40    local cred="email=daddeldi&password=daddeldu"
41    local session_path="/var/www/data"
42
43    # Every ongoing session will be deleted
44    sudo -u www-data rm -r $session_path/session-* 2> /dev/null
45
46    curl -s -b "$jar" -c "$jar" "$base/infra-login.sh?$cred"
47
48    if test ! -d $session_path/session-*; then
49      echo 'session check failed - login unsuccessful!' >&2
50      return 1
51    fi
52
53    curl -s -b "$jar" -c "$jar" "$base/infra-logout.sh?$cred"
54
55    if test -d $session_path/session-*; then
56      echo 'session check failed - logout unsuccessful!' >&2
57      return 1
58    fi
59
60    test -n "$prefix" && rm -rf "$prefix"
61    return 0
62  }
63
64  test_cleaner &&
65    test_watcher &&
66    test_session &&
67    echo 'Alles klappt super :)'
```

VI Utils

I do-all.sh

```
1  #!/usr/bin/env bash
2
3  cd "$(dirname "$0")"
```

```
./get-data.sh
gnuplot make-png.gpi
convert boxes-git-commits.png six:- && echo
```

II get-data.sh

```
#!/usr/bin/env bash

# Echoes data of current meeting as a single line
#
# Format:
# MeetingNr FirstKeyword SecondKeyword ... LastKeyword TotalAmount
echo_meeting_data() {
  echo -n "$meeting_nr "
  while read -r keyword; do
    # '!' introduces indirection, as the
    # substituted value becomes the new
    # variable to substitute the value of with
    # See also: man bash, Parameter Expansion
    echo -n "${!keyword:-0} "
  done < ../../.githooks/keywords.txt
  echo "$total"
}

# Saves data for current meeting
# and increments to next meeting
save_data_for_meeting() {
  echo_meeting_data >> commits.dat && ((++meeting_nr))
}

# Increments the counter related to the
# keyword and saves the data when useful
handle_hash() {
  local keyword="$1"
  ((++$keyword))
  ((++total))

  # Continue with next meeting on TODO update
  test "$keyword" = 'TODO' && save_data_for_meeting
```

```
}

# Global state to assign commits to meetings
# and count the total amount of commits
meeting_nr=1
total=0
test -f commits.dat && rm commits.dat

# Get all hashes from oldest to newest
cd "$(dirname "$0")"
first_hash='4ce528af6f4b7be260456f514c76ac7ff1a136c0'
hashes="$(git rev-list --reverse $first_hash^..HEAD)"

# Go through all hashes one by one and handle them
for githash in $hashes; do
  isolated_keyword="$(git log --oneline -1 "$githash" |
    cut -d ' ' -f 2 |
    sed 's/:$//')"
  handle_hash "$isolated_keyword"
done

# Also, save current data regardless
# of whether the last commit was a TODO
save_data_for_meeting
```

III make-png.gpi

```
set terminal pngcairo size 2048,1024 font 'Verdana,32'
set output 'boxes-git-commits.png'

set title 'Commit-Anzahlen pro Tag nach Meetings'
set key left top font ',22'
set xlabel 'Meeting-Nummer'
set ylabel 'Anzahl Commits'
set xrange [-1:24]
set tics scale 0

# Border
set style line 80 linetype 1 linecolor rgb 'black' linewidth 4
```

```
13 set border 3 back linestyle 80
14
15 # Background
16 set style line 81 linetype 1 linecolor rgb '#808080' linewidth 2
17 set grid back linestyle 81
18
19 set style histogram rowstacked
20 set boxwidth 0.9 relative
21 set style fill solid 1.0
22
23 plot 'commits.dat' using  2:xticlabels(1) with histogram lc rgb '#704E2E' title
   ↪  'TODO', \
24      'commits.dat' using  3:xticlabels(1) with histogram lc rgb '#DB3180' title
        ↪  'README', \
25      'commits.dat' using  4:xticlabels(1) with histogram lc rgb '#0075A2' title
        ↪  'INIT', \
26      'commits.dat' using  5:xticlabels(1) with histogram lc rgb '#669BBC' title
        ↪  'DEPLOY', \
27      'commits.dat' using  6:xticlabels(1) with histogram lc rgb '#A9D3FF' title
        ↪  'HOOKS', \
28      'commits.dat' using  7:xticlabels(1) with histogram lc rgb '#6F66D2' title
        ↪  'LATEX', \
29      'commits.dat' using  8:xticlabels(1) with histogram lc rgb '#997EAC' title
        ↪  'MAP', \
30      'commits.dat' using  9:xticlabels(1) with histogram lc rgb '#48A9A6' title
        ↪  'QUEUE', \
31      'commits.dat' using 10:xticlabels(1) with histogram lc rgb '#F18F01' title
        ↪  'SESSION', \
32      'commits.dat' using 11:xticlabels(1) with histogram lc rgb '#F3A712' title
        ↪  'DATA', \
33      'commits.dat' using 12:xticlabels(1) with histogram lc rgb '#E4572E' title
        ↪  'TESTS', \
34      'commits.dat' using 13:xticlabels(1) with histogram lc rgb '#97DB4F' title
        ↪  'UTILS', \
35      'commits.dat' using 14:xticlabels(1) with histogram lc rgb '#417B5A' title
        ↪  'WWW', \
36      'commits.dat' using 15:xticlabels(1) with histogram lc rgb '#32021F' title 'CGI'
37
```

IV show-all-project-related.sh

```
#!/usr/bin/env bash

start_hash='4ce528af6f4b7be260456f514c76ac7ff1a136c0'

commits="$(git rev-list $start_hash^..HEAD)"
for commit in $commits; do
  git log -1 "$commit" && echo
done
```

V show-commits-fitting-keyword.sh

```
#!/usr/bin/env bash

[[ -z $1 ]] && exit 1
hashes=$(git log --oneline |
  grep "^[a-z0-9]* ${1^^}:" |
  cut -d ' ' -f 1)
for hash in $hashes; do
  git log "$hash" -1 && echo
done
```

VII Queue

I infra-queue-lib.sh

```
#!/usr/bin/env bash
enable sleep
enable mkdir
enable rm

lock() {
  # Implements a spinlock in the filesystem
  # Setting the first argument causes the default 0.1s wait time to be
```

```
  # overridden by the specified value
  while ! mkdir 'lock' 2> /dev/null; do
    echo "${EPOCHREALTIME/[.,]} - wait for lock" >> queue.log
    sleep "${2:-0.1}"
  done
  echo > "lock/$1"
}

unlock() {
  rm -rf 'lock'
}

enqueue() {
  lock enqueue

  # Create node
  timestamp="${EPOCHREALTIME/[.,]}"
  if test -z "$1" -o -z "$2" -o -z "$3"; then
    echo "${EPOCHREALTIME/[.,]} - failed enqueue operation (missing arguments)" >>
    ↪  queue.log
    unlock
    exit 1
  fi

  echo "$timestamp|-|$1|$2|$3" > "node-$timestamp"
  chmod g+w "node-$timestamp"

  # Set pointers
  read -r tail_ref < 'tail'
  if test "$tail_ref" != '-'; then
    IFS='|' read -r stamp ref email subject content < "$tail_ref"
    echo "$stamp|node-$timestamp|$email|$subject|$content" > "$tail_ref"
  else
    echo "node-$timestamp" > 'head'
  fi

  echo "node-$timestamp" > 'tail'

  # Log
  echo "${EPOCHREALTIME/[.,]} - successful enqueue operation" >> queue.log
  echo "$1" >> enqueue.order
  unlock
```

```
}

dequeue() {
  lock dequeue 1
  read -r head_ref < 'head'
  if test "$head_ref" != '-'; then
    IFS='|' read -r stamp ref email subject content < "$head_ref"

    # Email processing
    echo "$head_ref" "$email" "$subject" "$content" >> email.log

    # Set pointers
    echo "$ref" > 'head'
    if test "$ref" = '-'; then
      echo '-' > 'tail'
    fi

    # Delete node
    rm -f "$head_ref"
  else
    unlock
    sleep 3
    return 2
  fi

  # Log
  echo "${EPOCHREALTIME/[.,]} - successful dequeue operation" >> queue.log
  echo "$email" >> dequeue.order
  unlock
}
```

II step-queue-lib.sh

```
#!/usr/bin/env bash
lock() {
  # Implements a spinlock in the filesystem
  # Setting the first argument causes the default 0.1s wait time to be
  # overridden by the specified value
  while ! mkdir 'lock' 2> /dev/null; do
```

```
7     echo "$(date +'%s%N') - wait for lock" >> queue.log
8     if test -z "$2"; then
9       sleep 0.1
10    else
11      sleep 2
12    fi
13  done
14  echo > "lock/$1"
15 }
16
17 unlock() {
18   rm -rf 'lock'
19 }
20
21 enqueue() {
22   lock enqueue
23
24   # Create node
25   timestamp="$(date +'%s%N')"
26   if test -z "$1" -o -z "$2" -o -z "$3"; then
27     echo "$(date +'%s%N') - failed enqueue operation (missing arguments)" >>
       ↪  queue.log
28     unlock
29     exit 1
30   fi
31
32   echo "$timestamp|-|$1|$2|$3" > "node-$timestamp"
33   chmod g+w "node-$timestamp"
34
35   # Set pointers
36   tail_ref="$(cat 'tail')"
37   if test "$tail_ref" != '-'; then
38     stamp="$(cat "$tail_ref" | cut -d'|' -f 1)"
39     ref="$(cat "$tail_ref" | cut -d'|' -f 2)"
40     email="$(cat "$tail_ref" | cut -d'|' -f 3)"
41     subject="$(cat "$tail_ref" | cut -d'|' -f 4)"
42     content="$(cat "$tail_ref" | cut -d'|' -f 5)"
43     echo "$stamp|node-$timestamp|$email|$subject|$content" > "$tail_ref"
44   else
45     echo "node-$timestamp" > 'head'
46   fi
47
```

```
48    echo "node-$timestamp" > 'tail'
49
50    # Log
51    echo "$(date +'%s%N') - successful enqueue operation" >> queue.log
52    echo "$1" >> enqueue.order
53    unlock
54  }
55
56  dequeue() {
57    lock dequeue 1
58    head_ref="$(cat 'head')"
59    if test "$head_ref" != '-'; then
60      stamp="$(cat "$head_ref" | cut -d'|' -f 1)"
61      ref="$(cat "$head_ref" | cut -d'|' -f 2)"
62      email="$(cat "$head_ref" | cut -d'|' -f 3)"
63      subject="$(cat "$head_ref" | cut -d'|' -f 4)"
64      content="$(cat "$head_ref" | cut -d'|' -f 5)"
65
66      # Email processing
67      echo "$head_ref" "$email" "$subject" "$content" >> email.log
68
69      # Set pointers
70      echo "$ref" > 'head'
71      if test "$ref" = '-'; then
72        echo '-' > 'tail'
73      fi
74
75      # Delete node
76      rm -f "$head_ref"
77    else
78      unlock
79      sleep 3
80      return 2
81    fi
82
83    # Log
84    echo "$(date +'%s%N') - successful dequeue operation" >> queue.log
85    echo "$email" >> dequeue.order
86    unlock
87  }
```

III restore.sh

```
#!/usr/bin/env bash

cd /data/queue
! test -d 'lock' && exit 0

test -e 'lock/enqueue' && sudo -i -u www-data enqueue
test -e 'lock/dequeue' && sudo -i -u workuser dequeue
rm -rf 'lock'

enqueue() {
  # Read tail ref
  read -r tail_ref < 'tail'
  newest_node="$(echo node-* | tr ' ' '\n' | sort -rn | head -1)"
  test "$tail_ref" = "$newest_node" && return 0

  # Recover if wrong ref
  chmod g+w "$newest_node"
  if test "$tail_ref" != '-'; then
    IFS='|' read -r stamp ref email subject content < "$tail_ref"
    echo "$stamp|$newest_node|$email|$subject|$content" > "$tail_ref"
  else
    echo "$newest_node" > 'head'
  fi
  echo "$newest_node" > 'tail'

  # Log necessary data
  IFS='|' read -r _ _ email _ < "$newest_node"
  echo "$email" >> enqueue.order
  echo "${EPOCHREALTIME/[,.]} - RECOVERED enqueue operation" >> queue.log
}

dequeue() {
  last_node="$(tail -1 email.log | cut -d ' ' -f 1)"
  ! test -f "$last_node" && return 0

  # Update refs
  IFS='|' read -r _ ref _ < "$last_node"
  echo "$ref" > 'head'
  if test "$ref" = '-'; then
```

```
    echo '-' > 'tail'
  fi

  rm -f "$last_node"

  # Log necessary data
  IFS='|' read -r _ _ email _ < "$last_node"
  echo "$email" >> dequeue.order
  echo "${EPOCHREALTIME/[,.]} - RECOVERED dequeue operation" >> queue.log
}
```

IV watcher.sh

```
#!/usr/bin/env bash
cd /data/queue

# Get worker pid and exit if it's still running
if test -e 'worker.pid'; then
  read -r pid < worker.pid
  ps "$pid" | grep 'worker.sh' && exit 0
fi

# Restart worker and save pid
nohup ./worker.sh &
echo "$!" > worker.pid
```

V worker.sh

```
#!/usr/bin/env bash
enable sleep
source infra-queue-lib.sh

while true; do
  dequeue
  sleep 0.3
done
```

VIII WWW

I app.js

```
let map
let mapInterval = null
let mapInitialized = false

let polyLines = {}
let markers = {}
let shipTimestamps = {}
let lstMsg = 0
let newMarkerAdded = false

let tableInterval = null
let highlightedMMSI = null

/* ------------------------------ INIT ------------------------------ */

function init() {
  initAccordion()

  // Login and logout buttons
  document.getElementById("loginBtn").addEventListener("click", () => {
    let email = document.getElementById("email").value
    let password = document.getElementById("password").value
    login(email, password)
  })
  document.getElementById("logoutBtn").addEventListener("click", () => {
    logout()
  })
}

function initAccordion() {
  let sections = document.getElementsByClassName("accordion")

  // Adds event listeners for all sections
  for (let i = 0; i < sections.length; ++i) {
    sections[i].addEventListener("click", (e) => {
      let target = e.target
```

```
37        let isActive = target.classList.contains("active")
38        closePanels()
39
40        // Open panel of clicked button if desired
41        if (!isActive) {
42          target.classList.add("active")
43          target.nextElementSibling.classList.add("panelactive")
44
45          if (i === 1) {
46            // Init leaflet for section 2
47            if (!mapInitialized) {
48              initMap()
49            }
50            initMapContent()
51
52            // See: https://stackoverflow.com/questions/779379/why-is-settimeoutfn-0-s
              ↪  ometimes-useful
53            map.invalidateSize()
54          }
55          // Create table for section 3
56          else if (i === 2) {
57            buildShipTable()
58            tableInterval = setInterval(buildShipTable, 1000)
59
60          }
61        }
62
63        // We should clear the interval everytime except for when we just started it
64        if (isActive) {
65          clearMapInterval()
66          clearTableInterval()
67        } else if (i !== 1) {
68          clearMapInterval()
69        } else if (i !== 2) {
70          clearTableInterval()
71        }
72      })
73    }
74  }
75
76  function clearMapInterval() {
77    clearInterval(mapInterval)
```

```
78    mapInterval = null
79  }
80
81  /* ------------------------------ LOGIN ------------------------------ */
82
83  function login(email, pwd) {
84    let xhr = new XMLHttpRequest()
85    let request = `../cgi-bin/infra-login.sh?email=${email}&password=${pwd}`
86    xhr.open("GET", request)
87
88    xhr.onreadystatechange = () => {
89      if (xhr.readyState === 4 && xhr.status === 200 &&
      ↪  xhr.responseText.includes("true")) {
90        document.getElementById("loginError").classList.add("hidden")
91        afterLogin(email)
92      } else {
93        // Reset input fields
94        document.getElementById("login").reset()
95
96        // Colorize the border red for a short moment
97        document.getElementById("email").style.borderColor = "red"
98        document.getElementById("password").style.borderColor = "red"
99        setTimeout(() => {
100         document.getElementById("email").style.borderColor = ""
101         document.getElementById("password").style.borderColor = ""
102       }, 1500)
103
104       // Show error message
105       document.getElementById("loginError").classList.remove("hidden")
106     }
107   }
108   xhr.send()
109 }
110
111 function afterLogin(email) {
112
113   // Cuts the Email-string at the @ letter
114   let displayName = email.includes("@") ? email.split("@")[0] : email
115
116   document.getElementById("login").classList.add("hidden")
117   document.getElementById("logout").classList.remove("hidden")
118   document.getElementById("userEmail").textContent = displayName
```

```
119   document.getElementById("section1Header").textContent = "Logout"
120
121   // Enable section 2 & 3
122   document.getElementById("section2Header").disabled = false
123   document.getElementById("section3Header").disabled = false
124 }
125
126 /* ------------------------------ LOGOUT ------------------------------ */
127
128 function logout() {
129   let xhr = new XMLHttpRequest()
130   xhr.open("GET", "../cgi-bin/infra-logout.sh")
131   xhr.onreadystatechange = () => {
132     if (xhr.readyState === 4 && xhr.status === 200) {
133       afterLogout()
134     }
135   }
136   xhr.send()
137 }
138
139 function afterLogout() {
140   // Reset accordion
141   document.getElementById("login").classList.remove("hidden")
142   document.getElementById("logout").classList.add("hidden")
143   document.getElementById("loginError").classList.add("hidden")
144   document.getElementById("userEmail").textContent = ""
145   document.getElementById("section1Header").textContent = "Login"
146
147   // Disable section 2 & 3
148   document.getElementById("section2Header").disabled = true
149   document.getElementById("section3Header").disabled = true
150
151   // Reset section 1 to allow clean login
152   document.getElementById("login").reset()
153   document.getElementById("email").style.borderColor = ""
154   document.getElementById("password").style.borderColor = ""
155
156   closePanels()
157
158   // Open first panel
159   let firstBtn = document.getElementsByClassName("accordion")[0]
160   firstBtn.classList.add("active")
```

```
161   firstBtn.nextElementSibling.classList.add("panelactive")
162
163   clearTableInterval()
164 }
165
166 function closePanels() {
167   let buttons = document.getElementsByClassName("accordion")
168   let panels = document.getElementsByClassName("panelactive")
169
170   for (let button of buttons) {
171     button.classList.remove("active")
172   }
173   for (let panel of panels) {
174     panel.classList.remove("panelactive")
175   }
176 }
177
178 /* ---------------------------- MAP ---------------------------- */
179
180 function initMap() {
181   map = L.map('map').setView([53.54, 8.5835], 17)
182   L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
183     minZoom: 8,
184     maxZoom: 19,
185     attribution: '<a href="https://openstreetmap.org">OpenStreetMap</a>'
186   }).addTo(map)
187   mapInitialized = true
188
189   // Initialize marker popups
190   updateMarkerPopupsFromTable()
191 }
192
193 function initMapContent() {
194   // Clear old polylines
195   for (let mmsi in polyLines) {
196     map.removeLayer(polyLines[mmsi])
197   }
198   polyLines = {}
199   lstMsg = 0
200
201   let xhr = new XMLHttpRequest()
202   xhr.open("GET", "../cgi-bin/infra-get-positions.sh")
```

```
203  xhr.onreadystatechange = () => {
204    if (xhr.readyState === 4 && xhr.status === 200) {
205      handleMapContent(xhr.responseText)
206      // DOS protection
207      if (mapInterval) {
208        clearMapInterval()
209      }
210      mapInterval = setInterval(update, 1000)
211    }
212  }
213  xhr.send()
214
215  function update() {
216    let xhr = new XMLHttpRequest()
217    xhr.open("GET", `../cgi-bin/infra-update-positions.sh?lstMsg=${lstMsg}`)
218    xhr.onreadystatechange = () => {
219      if (xhr.readyState === 4 && xhr.status === 200) {
220        handleMapContent(xhr.responseText)
221      }
222
223      // Update popups if necessary
224      if (newMarkerAdded) {
225        updateMarkerPopupsFromTable()
226        newMarkerAdded = false
227      }
228    }
229    xhr.send()
230  }
231
232  // Updates polylines and markers given the responseText
233  function handleMapContent(responseText) {
234    // Insert new
235    let latlngs = processPositionsResponse(responseText)
236    for (let mmsi in latlngs) {
237      let pos = latlngs[mmsi]
238      handlePolyline(mmsi, pos)
239      handleMarker(mmsi, pos[pos.length - 1])
240    }
241
242    // Remove old
243    removeOldShips()
244    removeOldPositions()
```

```
245
246     function removeOldShips() {
247       let now = Date.now()
248       let fiveMinInMs = 300000
249
250       for (let mmsi in shipTimestamps) {
251         // Remove markers if no update within 5min
252         if (markers[mmsi] && now - fiveMinInMs > shipTimestamps[mmsi]) {
253           map.removeLayer(markers[mmsi])
254           delete markers[mmsi]
255           if (polyLines[mmsi]) {
256             map.removeLayer(polyLines[mmsi])
257             delete polyLines[mmsi]
258           }
259         }
260       }
261     }
262
263     function removeOldPositions() {
264       let now = Date.now()
265       let hourInMs = 3600000
266
267       // Get latlngs of mmsi and replace if necessary
268       for (let mmsi in polyLines) {
269         // Examine current latlngs
270         let latlngs = polyLines[mmsi]
271           .getLatLngs()
272           .filter((x) => now - x.alt <= hourInMs)
273
274         // Redraw only if necessary
275         if (latlngs.length != polyLines[mmsi].getLatLngs().length) {
276           polyLines[mmsi].setLatLngs(latlngs)
277           polyLines[mmsi].redraw()
278         }
279       }
280     }
281   }
282
283 function processPositionsResponse(responseText) {
284   let acc = {}
285
286   // Parse coordinates into latlngs
```

```
287   let positions = responseText.split("\n")
288   for (let i = 0; i < positions.length - 2; ++i) {
289     let [mmsi, lat, lon, posDate, posTime, shipDate, shipTime] = positions[i].split("
      ↪  ")
290
291     // Add positions
292     if (!acc[mmsi]) {
293       acc[mmsi] = []
294     }
295     acc[mmsi].push([
296       parseFloat(lat),
297       parseFloat(lon),
298       Date.parse(`${posDate} ${posTime}`)
299     ])
300
301     // Update ship timestamps
302     shipTimestamps[mmsi] = Date.parse(`${shipDate} ${shipTime}`)
303   }
304
305   // Update newest index and return all coordinates
306   lstMsg = parseInt(positions[positions.length - 2])
307   return acc
308 }
309
310 function handlePolyline(mmsi, latlngs) {
311   if (!polyLines[mmsi]) {
312     polyLines[mmsi] = L.polyline(latlngs, {
313       color: '#f48c06',
314       weight: 3,
315       dashArray: '6, 6'
316     }).addTo(map)
317   } else {
318     // To not replace the existing data points, we have to add coordinates
319     // sequentially
320     for (let coords of latlngs) {
321       polyLines[mmsi].addLatLng(coords)
322     }
323
324     // Adds after removal
325     if (!map.hasLayer(polyLines[mmsi])) {
326       polyLines[mmsi].addTo(map)
327     }
```

```
328   }
329 }
330
331   function handleMarker(mmsi, pos) {
332     if (!markers[mmsi]) {
333       markers[mmsi] = L.marker(pos).addTo(map)
334       // Popup shows the MMSI
335       .bindPopup(mmsi)
336       .on('popupopen', () => {
337         // Safes the normalized MMSI
338         highlightedMMSI = mmsi.replace(/['"]+/g, "").trim()
339       })
340       .on('popupclose', () => {
341         highlightedMMSI = null
342       })
343
344       // Remember that atleast one new marker got added
345       newMarkerAdded = true
346     } else {
347       markers[mmsi].setLatLng(pos)
348
349       // Adds after removal
350       if (!map.hasLayer(markers[mmsi])) {
351         markers[mmsi].addTo(map)
352       }
353     }
354   }
355 }
356
357 // Adds ship names to the marker popups
358 function updateMarkerPopupsFromTable() {
359   buildShipTable()
360   clearTableInterval()
361
362     let container = document.getElementById("ship-table")
363     let rows = container.querySelectorAll("tr")
364
365     for (let row of rows) {
366       let cells = row.querySelectorAll("td")
367       if (cells.length >= 2) {
368
369       // Table mmsi and shipname
```

```
370       let table_mmsi = cells[0].textContent.replace(/['"]+/g, "").trim()
371       let name = cells[1].textContent.trim()
372
373       // Search through markers for the matching mmsi
374       for (let mmsi in markers) {
375         let normalized = mmsi.replace(/['"]+/g, "").trim()
376         if (table_mmsi === normalized && name.toUpperCase() !== "NULL") {
377           markers[mmsi].bindPopup(`${name} ${mmsi}`)
378           break
379         }
380       }
381     }
382   }
383 }
384
385 /* ----------------------------- TABLE ----------------------------- */
386
387 function buildShipTable() {
388   let xhr = new XMLHttpRequest()
389   xhr.open("GET", "../cgi-bin/infra-get-table.sh")
390
391   xhr.onreadystatechange = () => {
392     if (xhr.readyState === 4 && xhr.status === 200) {
393       let container = document.getElementById("ship-table")
394       container.innerHTML = xhr.responseText
395
396       // If a marker popup got opened
397       if (highlightedMMSI) {
398         highlightShipRow()
399       }
400     }
401   }
402
403   xhr.send()
404 }
405
406 // Searches for the popup MMSI inside of the table and highlights that row
407 function highlightShipRow() {
408   let rows = document.querySelectorAll("#ship-table tr")
409
410   for (let row of rows) {
411     let cells = row.querySelectorAll("td")
```

```
412     if (!cells || cells.length === 0) continue
413
414     let cellValue = cells[0].textContent
415
416     if (cellValue == highlightedMMSI) {
417       row.classList.add("highlighted")
418       break
419     }
420   }
421 }
422
423 function clearTableInterval() {
424   clearInterval(tableInterval)
425   tableInterval = null
426 }
427
428 window.onload = init
```

II index.html

```
1  <!DOCTYPE html>
2  <html lang="de">
3    <head>
4      <meta charset="UTF-8" />
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Akkordeon mit Leaflet-Karte</title>
7      <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css"
8                              integrity="sha256-p4NxAoJBhIIN+hmNHrzRCf9tD/miZyoHS5obTRR
                               ↪  9BMY="
9                              crossorigin=""/>
10     <link rel="stylesheet" href="main.css" />
11     <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"
12             integrity="sha256-20nQCchB9co0qIjJZRGuk2/Z9VM+kNiyxNV1lvTlZBo="
13             crossorigin=""></script>
14     <script src="app.js"></script>
15   </head>
16   <body>
17     <button class="accordion active" id="section1Header">Login</button>
18     <div class="panel panelactive" id="section1Panel">
```

```
19        <form id="login">
20          <input type="email" id="email" placeholder="E-Mail">
21          <input type="password" id="password" placeholder="Passwort">
22          <button type="button" id="loginBtn">Login</button>
23        </form>
24
25        <div id="loginError" class="error hidden">
26          Login fehlgeschlagen. Bitte überprüfen Sie Ihre Eingaben!
27        </div>
28
29        <div id="logout" class="hidden">
30          <p>Angemeldet als: <span id="userEmail"></span></p>
31          <button id="logoutBtn">Logout</button>
32        </div>
33      </div>
34
35      <button class="accordion" id="section2Header" disabled>Karte</button>
36      <div class="panel" id="section2Panel">
37        <div id="map"></div>
38      </div>
39
40      <button class="accordion" id="section3Header" disabled>Tabelle</button>
41      <div class="panel" id="section3Panel">
42       <div id="ship-table"></div>
43      </div>
44    </body>
45  </html>
```

III main.css

```
1   body {
2     display: flex;
3     flex-direction: column;
4     font-family: Verdana, sans-serif;
5     background-color: #f5f3f4;
6     align-items: center;
7   }
8
9   .hidden {
```

```
10    display: none;
11  }
12
13  .accordion {
14    width: 80%;
15    padding: 0.8rem;
16    margin: 0.75rem 0;
17    align-items: center;
18    background-color: #00b4d8;
19    color: #ffffff;
20    cursor: pointer;
21    font-size: 1.5rem;
22    outline: none;
23    border: none;
24    border-radius: 20px;
25  }
26
27  .accordion:hover {
28    background-color: #0096c7;
29  }
30
31  .accordion:disabled {
32    cursor: not-allowed;
33    background: rgba(0, 0, 0, .1);
34    color: rgba(0, 0, 0, .3);
35  }
36
37  .panel {
38    display: none;
39    padding: 3vh;
40    overflow: hidden;
41    width: 80%;
42  }
43
44  .panelactive {
45    display: block;
46  }
47
48  input {
49    font-size: 1.25rem;
50    padding: 0.8rem;
51    outline: none;
```

```
52    border: 2px solid #d3d3d3;
53    background-color: transparent;
54    border-radius: 20px;
55  }
56
57  input:focus {
58    border-color: #00b4d8;
59  }
60
61  #loginBtn, #logoutBtn {
62    background-color: #00b4d8;
63    width: 7rem;
64    padding: 0.8rem;
65    border: 0;
66    border-radius: 20px;
67    color: #ffffff;
68    cursor: pointer;
69    font-size: 1.25rem;
70  }
71
72  #loginBtn:hover,
73  #logoutBtn:hover {
74    background-color: #0096c7;
75  }
76
77  #login,
78  #logout {
79    text-align: center;
80  }
81
82  #logout p {
83    font-size: 1.25rem;
84  }
85
86  /* on small screens align underneath */
87  @media (max-width: 600px) {
88    form {
89      display: flex;
90      flex-direction: column;
91      align-items: center;
92      gap: 1rem;
93    }
```

```
94
95    #login input,
96    #login button {
97      width: 100%;
98      max-width: 220px;
99    }
100 }
101
102 .error {
103   color: red;
104   font-weight: bold;
105   margin-top: 8px;
106   text-align: center;
107 }
108
109 #map {
110   height: 70vh;
111 }
112
113 #ship-table {
114   overflow-y: scroll;
115 }
116
117 table {
118   width: 100%;
119   border: 1px solid black;
120   border-collapse: collapse;
121   background-color: #f0f8ff;
122   table-layout: auto;
123 }
124
125 table th, table td {
126   border: 1px solid black;
127   padding: 4px;
128   text-align: left;
129 }
130
131 table th {
132   background-color: #00b4d8;
133   color: white;
134 }
135
```

```
/* Alternate row coloring */
table tr:nth-child(even) {
  background-color: #f9fcff;
}

.highlighted {
  background-color: #f48c06 !important;
}

table tr:hover {
  background-color: #dceeff;
}
```

Selbstständigkeitserklärung

Wir versichern, die von uns vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, haben wir als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die wir für die Arbeit benutzt haben, sind angegeben. Die Arbeit haben wir mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegt.

Bremerhaven, den 30. August 2025

Unterschrift:

Bastian Buch

Luca Focken

Leon Stüve

Lukas Weber