Hochschule Bremerhaven

Fachbereich II
Infrastruktur
Informatik B.Sc.

Modul

Entwicklung einer Webanwendung zur Visualisierung von Schiffspositionen

Projekt im Modul Infrastruktur, Sommersemester 2024

Vorgelegt von: Mustafa Thamer (39628) .. ..
Kasem Rashrash (41398)
Ahmad Almohammad (38222)
Shiwan Ali Shahen (39604)

Vorgelegt am: 14. September 2024
Dozent:in: Prof. Dr. Olivar Radfelder

Inhaltsverzeichnis

1 Einleitung

Kombiniert mit dem Wissen aus dem Modul „Einführung in die Informatik" und den im Modul „Infrastruktur-2024" erlernten Werkzeugen, setzten wir uns zu viert (Ahmad Almohammad, Kasem Rashrash, Mustafa Thamer, Shiwan Al Shahen) zusammen, um eine Webanwendung zu entwickeln, die es den Benutzern ermöglicht, sich mit ihrer E-Mail und einem Passwort anzumelden. Nach dem Login können die Benutzer die Position von Schiffen in Echtzeit sowohl auf einer interaktiven Karte als auch in einer Tabelle anzeigen lassen. Die Positionsdaten der Schiffe werden in Echtzeit vom RHODES-Server erfasst, verarbeitet und visuell dargestellt. Die sogenannten AIS-Daten werden gesammelt, gefiltert und in einer Datenbank gespeichert. Von dort werden sie über ein AJAX-Skript abgerufen und auf einer interaktiven Karte mithilfe der Leaflet-Bibliothek dargestellt. Das Ziel des Projekts ist es, die Konzepte und Werkzeuge, die wir im Unterricht gelernt haben, in einem echten Projekt anzuwenden. Dabei arbeiten wir ohne fertige Programmbibliotheken (Frameworks), um die einzelnen Bestandteile der Anwendung besser zu verstehen. Wir lernen, wie man eine Datenbank, einen Webserver und die Benutzeroberfläche (Frontend) zusammenbringt, damit sie miteinander arbeiten. Ein wichtiger Teilaspekt des Projekts ist vor allem die Arbeit im Team. Wir haben gemeinsam gelernt, miteinander zu arbeiten, mit Problemen und Herausforderungen umzugehen und uns zu organisieren. Für eine gute Zusammenarbeit nutzten wir das Programm „Git", um unsere Arbeit zu organisieren.
Mit Git können wir die Änderungen am Code speichern und verfolgen, was jedes Mitglied im Team gemacht hat. Der genaue Prozess wird in den folgenden Abschnitten detaillierter erklärt.

Am Ende des Projekts haben wir eine funktionierende Webanwendung und eine schriftliche Ausarbeitung, die den gesamten Entwicklungsprozess erklärt.

2 Grundlagen

In diesem Abschnitt werden die grundlegenden Voraussetzungen und Vorgaben für die Entwicklung der Webanwendung beschrieben. Es wird erläutert, welche Tools und Technologien verwendet wurden, welche Einschränkungen bei der Umsetzung zu beachten waren und welche Anforderungen an die Systemumgebung gestellt wurden.
Außerdem wird auf die eingesetzten Datenbank- und Webserver-Lösungen eingegangen.

2.1 Benutzte Werkzeuge und Technologien

Für die Entwicklung der Webanwendung wurden einfache Werkzeuge und Technologien genutzt:

-Programmiersprachen: JavaScript (für das Frontend) und Bash (für das Backend).
-Datenbank: MariaDB zur Speicherung der Benutzer- und Schiffsdaten.
-Webserver: Apache2 für die Bereitstellung der Anwendung.
-Versionskontrolle: Git wurde für die Zusammenarbeit im Team und die Verwaltung des Codes verwendet.

2.2 Begrenzungen und Anforderungen

-Keine Frameworks: Es durften keine zusätzlichen Frameworks wie Bootstrap, jQuery oder React verwendet werden. Nur „Vanilla-Javascript“ und „Vanilla-CSS“ waren erlaubt.
-Datenbank: Es war vorgegeben, MariaDB zu verwenden, keine anderen Datenbanksysteme.
-Backend: Das Backend musste mit Bash-Skripten arbeiten, andere Programmiersprachen waren nicht erlaubt.

2.3 Datenbankmanagementsystem und Webserver

-MariaDB: Ein relationales Datenbanksystem, das zur Speicherung von Benutzern, Schiffen und deren Positionen genutzt wurde. Es bietet Funktionen wie das Anlegen von Tabellen, das Speichern und Abfragen von Daten sowie das Löschen alter Positionsmeldungen.

-Apache2: Ein Webserver, der die Webanwendung ausliefert und die Kommunikation zwischen Frontend und Backend ermöglicht. Der Server wurde für die Bearbeitung von Benutzeranfragen und die Verwaltung der Datenbank verwendet.

-Virtualisierung: Die Entwicklung erfolgte lokal auf den eigenen Geräten. Dafür wurden entweder VM (Virtual Machine) oder WSL (Windows Subsystem for Linux) genutzt, um eine Linux-Umgebung bereitzustellen. Diese Umgebungen dienten lediglich dazu, die Anwendung lokal zu entwickeln und zu testen.

-Git für Versionskontrolle: Git wurde für die Verwaltung des Quellcodes und die Zusammenarbeit im Team eingesetzt. Alle Teammitglieder arbeiteten über ein gemeinsames Git-Repository, um Änderungen nachzuverfolgen und verschiedene Versionen des Projekts zu verwalten.

2.4 Überblick über die Aufgabenstellung

In diesem Projekt geht es darum, eine Webanwendung zu entwickeln, die verschiedene Funktionen bietet, wie z.B. Benutzeranmeldungen, die Anzeige von Schiffspositionen und die Simulation eines E-Mail-Versands.
Die Anwendung soll in der Lage sein, Positionsdaten von Schiffen in Echtzeit zu zeigen und regelmäßig aktualisierte Informationen anzuzeigen.
Im Folgenden werden die einzelnen Anforderungen des Projekts beschrieben:

2.4.1 1. Benutzeranmeldungen

- **E-Mail und Passwort Anmeldung**: Benutzer können sich mit ihrer E-Mail und einem Passwort anmelden, um Zugriff auf die Anwendung zu erhalten.
- **Akkordeon mit drei Gruppen**: Die Benutzeroberfläche besteht aus drei Gruppen: Login, Karte und einer HTML-Tabelle, die sich wie ein Akkordeon verhält. Das bedeutet, dass immer nur die Login-Gruppe offen sein kann.
- **Login-Anzeige**: Nach erfolgreichem Login kann optional der Name des Benutzers in der Überschrift angezeigt werden.

2.4.2 2. Kartenansicht

- **Leaflet-Karte**: Die Karte zeigt Schiffe an, von denen in den letzten 5 Minuten eine Positionsmeldung empfangen wurde.
- **Wege der Schiffe**: Die Route der Schiffe der letzten Stunde wird auf der Karte als Linie angezeigt.
- **HTML-Tabelle**: Eine Tabelle zeigt die Schiffe an, die derzeit auf der Karte zu sehen sind.

2.4.3 3. E-Mail-Simulation

- **E-Mail-Simulation**: Nach jeder Anmeldung wird eine E-Mail-Simulation durchgeführt. Es wird jedoch keine echte E-Mail versendet, sondern der Vorgang wird simuliert.
- **Eigenes Skript**: Für diese Simulation wird ein spezielles Skript geschrieben, das den E-Mail-Versand simuliert.
- **Separate Benutzerrolle**: Ein separater Benutzer („Workuser“) agiert als TCP-Server, der die Aufträge für den E-Mail-Versand entgegennimmt.

- **Erweiterbares Protokoll**: Das Protokoll zwischen der Webanwendung und dem E-Mail-Simulationsskript ist einfach, aber erweiterbar für zukünftige Aufgaben.
- **Warteschlange für Aufträge**: Alle E-Mail-Aufträge werden in eine Warteschlange gestellt, und nach der erfolgreichen Annahme erhält der Benutzer eine Bestätigung.

2.4.4 4. Datenbank

- **Benutzerdaten in eigener Tabelle**: Die Benutzerdaten (E-Mail und Passwort) werden in einer separaten Tabelle in der Datenbank gespeichert.
- **Schiffe in eigener Tabelle**: Schiffsinformationen wie Name und MMSI (Maritime Mobile Service Identity) werden in einer eigenen Tabelle gespeichert, wobei die MMSI als eindeutiger Schlüssel (Primary Key) dient.
- **AIS-Daten in eigener Tabelle**: Die von den Schiffen gesendeten Positionsmeldungen (AIS-Daten) werden in einer separaten Tabelle gespeichert.
- **Löschen alter Daten**: Ältere Positionsmeldungen, die älter als sieben Tage sind, werden regelmäßig aus der Datenbank gelöscht.

2.4.5 5. Frontend-Aktualisierung

- **Regelmäßige AJAX-Aufrufe**: Die Webanwendung aktualisiert sich automatisch einmal pro Sekunde durch AJAX-Aufrufe, um die neuesten Positionsdaten zu laden.
- **Nur neueste Daten laden**: Bei jeder Aktualisierung werden nur die neuesten Schiffspositionen geladen, um die Datenmenge zu reduzieren.
- **Ältere Positionen entfernen**: Positionen, die älter als eine Stunde sind, sowie Schiffe, die inaktiv sind, werden automatisch aus der Karte und der Tabelle entfernt.

2.4.6 6. Austauschbare CGI-Implementierungen

- **Mindestens eine austauschbare Aktion**: Eine Aktion der Webanwendung wird so implementiert, dass sie austauschbar ist, um verschiedene Lösungsansätze zu ermöglichen.
- **Einfache und optimierte Implementierung**: Es wird sowohl eine einfache als auch eine optimierte Version der Implementierung entwickelt, um die Unterschiede zu testen.

2.4.7 7. Tests und Monitoring

- **Komponententests**: Jede Komponente der Anwendung (wie z.B. Benutzeranmeldung, Kartendarstellung) kann einzeln getestet werden.
- **Belastbarkeitstest der Warteschlange**: Es werden Tests durchgeführt, um zu überprüfen, wie viele Anfragen die E-Mail-Warteschlange bewältigen kann.
- **Frontend-Tests ohne Backend**: Es ist möglich, das Frontend zu testen, ohne dass das Backend verbunden ist, um die Funktionalität der Benutzeroberfläche zu prüfen.
- **Monitoring mit Gnuplot**: Mit dem Programm Gnuplot werden Grafiken erstellt, die den Zustand der Anwendung überwachen, z.B. wie viele Benutzer aktuell eingeloggt sind oder wie viele Positionsmeldungen pro Sekunde verarbeitet werden.
- **Lasttests für die Benutzeranzahl**: Es werden Lasttests durchgeführt, um zu testen, wie viele Benutzer die Anwendung gleichzeitig verwenden können, ohne dass sie langsamer wird.

3 Projektplanung

Vorab, bevor wir mit dem Projekt begonnen haben, haben wir uns in den ersten Wochen regelmäßig an der Hochschule sowie online getroffen, um gemeinsam die Hausaufgaben zu bearbeiten, zu üben und Ideen auszutauschen.

Dabei haben wir uns auf einen Zeitplan geeinigt, da jedes Mitglied aus einem anderen Semester kommt und jeder individuelle Zeitpläne hat.

Daher haben wir einen Zeitplan erstellt, der für alle passt. Da wir mit der virtuellen Maschine arbeiten sollten, begannen wir, die Implementierungen dort vorzunehmen. Einige von uns hatten dabei Probleme mit bestimmten Werkzeugen und Einstellungen. Wir halfen einander und beseitigten diese Probleme gemeinsam, sodass wir gut starten konnten.

Von Anfang an war uns bewusst, dass eine systematische und strukturierte Vorgehensweise entscheidend für eine erfolgreiche Umsetzung des Projekts ist. Deshalb tauschten wir unsere Ideen aus und begannen mit der Erstellung von Diagrammen.

Die Modellierung diente als Diskussionsgrundlage und wurde während der Entwicklung entweder angepasst oder beibehalten. Für die technische Umsetzung der Programmierung verwendeten wir das Versionsverwaltungssystem Git.

Das ermöglichte uns eine effiziente Softwareentwicklung und Zusammenarbeit. Zudem nutzten wir HedgeDoc zum Protokollieren, und am Ende des Tages gaben wir uns immer kurze Updates, tauschten weiter Ideen aus und diskutierten über die committen Nachrichten in Git.

3.1 Die Verzeichnisstruktur

In unserem Projekt haben wir viele Ordner und Dateien. Die Struktur zeigt uns, wie alles geordnet ist, damit wir leicht finden, was wir brauchen und um schneller und besser zu arbeiten.

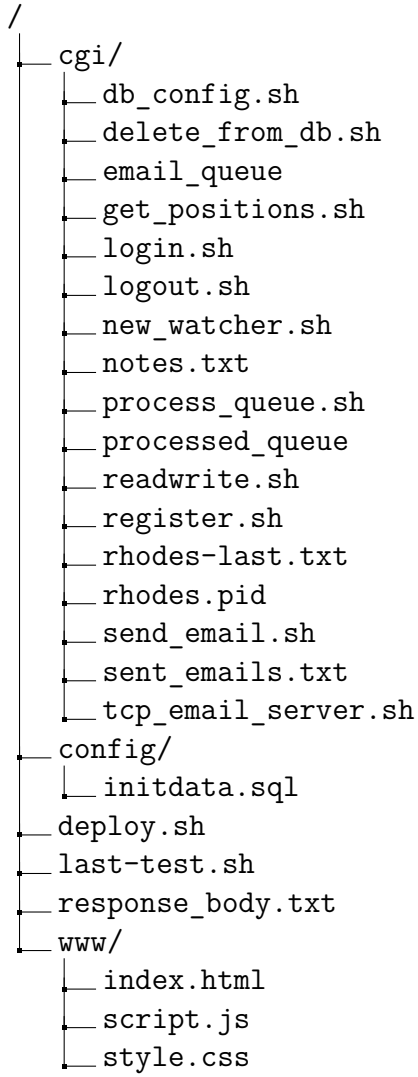


Abbildung 3.1: Verzeichnisstruktur

3.2 Modellierung

Für die UML-Diagramme haben wir app.creately.com benutzt. Am Anfang vom Projekt haben wir erstmal mit einem Use-Case-Diagramm die wichtigsten Funktionen von unserer Anwendung besprochen. Für die anderen Diagramme haben wir online.visual-paradigm.com verwendet. Diese Tools haben uns geholfen, die Abläufe und wie alles im System zusammenpasst, einfach zu zeigen.

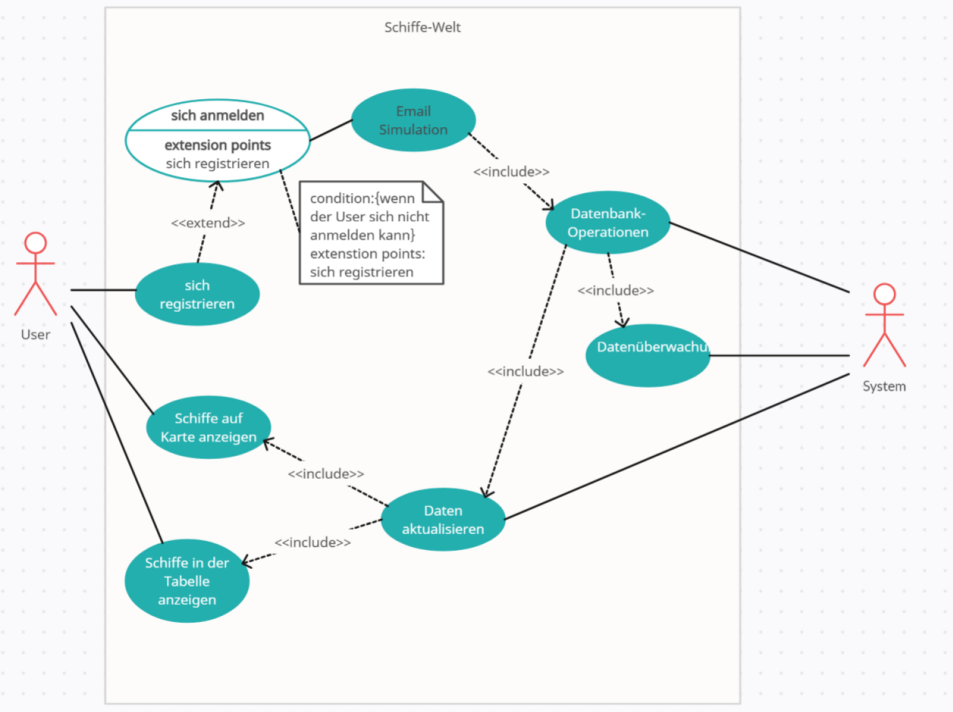


Abbildung 3.2: Use-Case

Das Use-Case zeigt, wie verschiedene Aktionen und Prozesse in einem System zusammenarbeiten. Es gibt zwei Akteure: der Benutzer (User) und das System.

Benutzeraktionen:

Der Benutzer kann sich im System anmelden. Wenn das nicht klappt, gibt es eine Möglichkeit zur Registrierung (sich registrieren). Dies ist ein Ëxtension Point also eine Erweiterung, die genutzt wird, wenn der Hauptprozess (Anmeldung) fehlschlägt. Sobald der Benutzer angemeldet ist, kann er Schiffe auf einer Karte anzeigen lassen oder sie in einer Tabelle sehen. Systemaktionen:

Es gibt einen Prozess namens ”Daten aktualisieren”, der für das Anzeigen der Schiffe auf der Karte und in der Tabelle wichtig ist. Das System kümmert sich auch um Datenbank-

Operationen und überwacht die Daten. Diese Prozesse werden durch Ëmail Simulationünd ”Datenüberwachungünterstützt. Zusammenhänge:

Das Anmeldeverfahren beinhaltet die Option, eine Registrierung durchzuführen, wenn nötig. Wenn der Benutzer sich registriert, wird eine E-Mail-Simulation durchgeführt (um die E-Mail-Adresse zu bestätigen). Verschiedene Prozesse hängen zusammen: Zum Beispiel werden ”Datenbank-Operationenünd ”Datenüberwachung”durch die Aktualisierung der Daten ausgelöst.

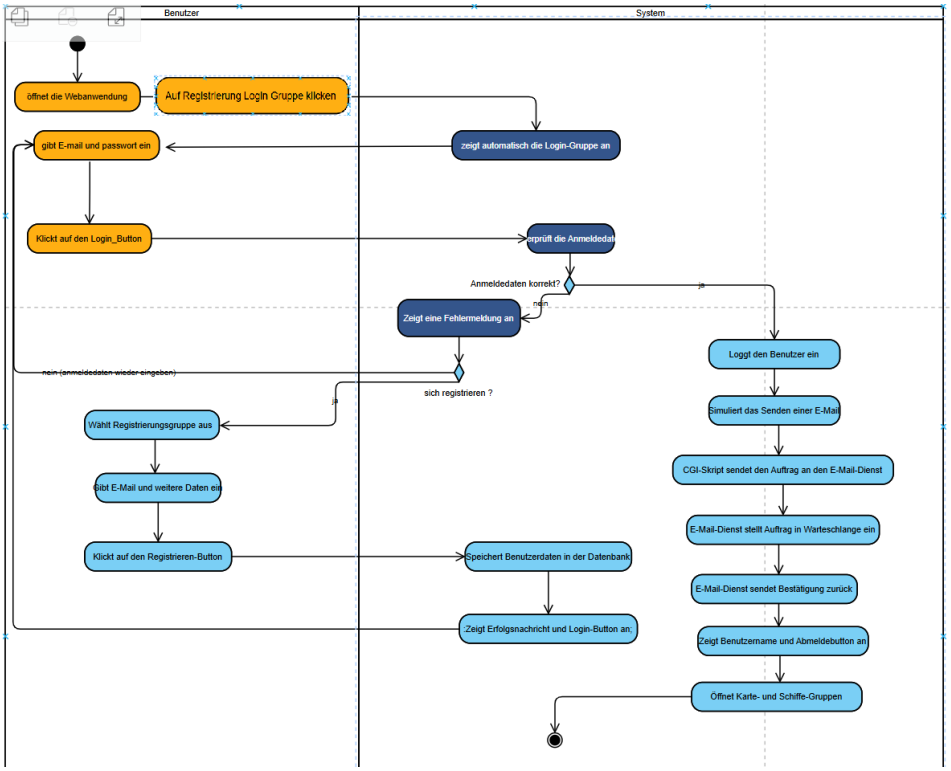


Abbildung 3.3: aktivitätsdiagramm

In unserem Team haben wir zusammen überlegt, wie der Anmelde- und Registrierungsprozess in unserer Webanwendung ablaufen soll. Das haben wir dann in diesem Diagramm festgehalten, damit wir einen Überblick haben.

Anmeldung:
Der Benutzer öffnet als erstes die Webseite und klickt auf die Login-Gruppe. Dann gibt er seine E-Mail und das Passwort ein und drückt auf den Login-Button. Jetzt ist das System dran und checkt, ob die Anmeldedaten stimmen. Wenn alles passt, wird der Benutzer eingeloggt. Im Hintergrund simuliert das System dann das Verschicken einer E-Mail. Das haben wir so eingerichtet: Ein CGI-Skript schickt die Anfrage an den E-Mail-Dienst, dieser stellt die Anfrage in eine Warteschlange und schickt danach eine Bestätigung zurück. Zum Schluss wird der

Benutzername und der Abmeldebutton angezeigt, und die Gruppen für die Karte und die Schiffe werden geöffnet. Registrierung:
Wenn die Anmeldedaten falsch sind, zeigt das System eine Fehlermeldung an. Dann haben wir uns überlegt, dass der Benutzer gefragt wird, ob er sich registrieren will. Wenn ja, klickt er auf die Registrierungsgruppe, gibt seine E-Mail und andere Daten ein und drückt auf den Registrieren-Button. Das System speichert dann die Daten in der Datenbank ab. Danach bekommt der Benutzer eine Erfolgsnachricht und sieht direkt den Login-Button.

3.3 Deploy-Skripte

In unserem Projekt haben wir verschiedene Deploy-Skripte erstellt, um unsere Dateien und Veränderungen zwischen dem Hopper-Server und der Docker-Umgebung effizient hin und her zu übertragen.
Diese Skripte helfen uns, die Dateien schnell und automatisch an den richtigen Ort zu bringen, ohne jedes Mal alles manuell erledigen zu müssen.
Hier eine Übersicht der Skripte, die wir verwenden:

3.3.1 deploy_local.sh

```
1 #!/bin/bash
2
3 sudo cp -a www/* /var/www/html/$USER-web/project_2024
4
5 sudo cp -a cgi/* /usr/lib/cgi-bin/project_2024
```

Listing 3.1: Kopiert Web- und CGI-Dateien auf den Server

Dieses Skript kopiert unsere Web- und CGI-Dateien vom lokalen Ordner an die richtigen Stellen auf dem Server.
www/: Es kopiert alle Webdateien aus dem Ordner www/ in das Verzeichnis /var/www/html/$USER-web/project_2024, wo unsere Webseite öffentlich zugänglich ist.
cgi/: Es kopiert alle CGI-Dateien aus dem Ordner cgi/ in das Verzeichnis /usr/lib/cgi-bin/project_2024, damit sie auf dem Server ausführbar sind.

3.3.2 deploy-docker.sh

```
1 #!/bin/bash
2
3 scp -r mydocker:/home/docker-$USER/* ./test
```

Listing 3.2: Kopiert Dateien von der Docker-Umgebung lokal

Dieses Skript befindet sich im sudo infra-2024-l und kopiert die Dateien aus unserer Docker-Umgebung lokal in den Ordner ./test.
So haben wir jedes Mal die Sachen zu uns rübergeholt und mit GIT hin und her übertragen, damit wir Veränderungen vornehmen können.
Mit dem scp-Befehl werden alle Dateien von mydocker:/home/docker-$USER/* auf unseren Computer in den Ordner ./test übertragen.

3.3.3 deploy-project.sh

```
1 #!/bin/bash
2
3 scp -r mydocker:/home/docker-$USER/project/* ./infra-2024-l
```

Listing 3.3: Kopiert Projektdateien von Docker zu lokalem GIT-Ordner

Dieses Skript befindet sich auch lokal im sudo infra-2024-l und kopiert alle Dateien unseres Projekts von der Docker-Umgebung mydocker:/home/docker-$USER/project/* zu uns lokal in den GIT-Ordner ./infra-2024-l.

3.3.4 deploy-hopper-to-docker.sh

```
1 #!/bin/bash
2
3 scp -q -r ./infra-2024-l/* mydocker:infra_2024_l_from_hopper
```

Listing 3.4: Kopiert Dateien vom lokalen Verzeichnis auf Docker

Mit diesem Skript kopieren wir die Dateien aus dem lokalen Verzeichnis ./infra-2024-l (in dem Fall unser Repository) vom Hopper-Server zu unserer Docker-Umgebung.
Es kopiert den Inhalt des Ordners ./infra-2024-l/* auf den Docker-Server in das Verzeichnis mydocker:infra$_2024_{lfrom_hopper}$.

Allgemein kann man sagen, dass diese Skripte den Datentransfer zwischen unserem Hopper-Server und Docker erleichtern und automatisieren.
Das ist besonders praktisch, wenn wir Änderungen auf dem Hopper vornehmen und diese dann auf Docker übertragen möchten, oder wenn wir Daten von Docker zurück auf den Hopper holen müssen.
Durch die Nutzung von scp können wir sicher und effizient Dateien kopieren, ohne das manuell machen zu müssen.
Die Skripte vereinfachen unsere Arbeit und sorgen dafür, dass wir Zeit sparen und mögliche Fehler beim Kopieren von Dateien vermeiden.

4 Die Anwendung

4.1 Funktionen und Benutzeroberfläche (Anmelden, Karte, Schiffe)

Login und Registrierung: Die Benutzer können sich mit ihrer E-Mail-Adresse und einem Passwort anmelden. Alternativ können sie ein Konto erstellen, wenn sie noch keines haben. Beide Funktionen gehören zur **Login-Gruppe**, die standardmäßig angezeigt wird.
Nach erfolgreicher Anmeldung erscheint der Name des Benutzers in der Kopfzeile, und ein Logout-Button wird sichtbar. Vor der Anmeldung ist nur die Login-Gruppe verfügbar, die anderen Gruppen sind gesperrt.

Karte: Nach der Anmeldung zeigt die Anwendung eine interaktive Karte mit den aktuellen Positionen der Schiffe an, die in den letzten fünf Minuten ein Signal gesendet haben.
Die Schiffe werden als Marker auf der Karte angezeigt, und ihre Bewegungen der letzten Stunde werden als Linien dargestellt.
Wenn der Benutzer auf einen Schiff-Marker klickt, werden detaillierte Informationen zum Schiff, wie Name, MMSI-Nummer und letzte gemeldete Position, angezeigt.

Schiffe: Neben der Karte gibt es eine HTML-Tabelle, die alle Schiffe auflistet, die aktuell auf der Karte zu sehen sind, einschließlich ihrer Namen und MMSI-Nummern.
Diese Tabelle wird ebenfalls erst nach erfolgreicher Anmeldung angezeigt.

4.2 Aufbau der Webseite (Front- und Backend)

Frontend: Das Frontend wurde mit HTML, CSS und JavaScript erstellt. Es verwendet ein Akkordeon-Layout, das in drei Abschnitte unterteilt ist: Login (mit Registrierung), Karte und Schiffe.
Standardmäßig wird nur die Login-Gruppe angezeigt. Die Karte und die Schiffstabelle werden erst nach erfolgreicher Anmeldung freigeschaltet.
Die Daten der Schiffe werden durch regelmäßige Ajax-Aufrufe aktualisiert, sodass die Seite nicht neu geladen werden muss.
Die interaktive Karte wurde mit der Leaflet-Bibliothek implementiert. Beim Klick auf einen Schiff-Marker auf der Karte öffnet sich ein Popup mit detaillierten Informationen zum ausgewählten Schiff.

Backend: Das Backend besteht aus Bash-Skripten, die als CGI-Skripte über den Apache-Webserver laufen.
Die Benutzer- und Schiffsdaten werden in einer MariaDB-Datenbank gespeichert. Das Frontend kommuniziert über HTTP-Anfragen mit dem Backend.
Eine zusätzliche Backend-Funktion simuliert den Versand von E-Mails bei der Registrierung oder Anmeldung.

4.3 Besonderheiten bei der Entwicklung

E-Mail-Simulation: Statt echte E-Mails zu versenden, wurde ein Dienst implementiert, der E-Mails simuliert.
Die Anfragen werden in eine Warteschlange gestellt und nach und nach verarbeitet.

Datenaktualisierung: Die Schiffsdaten werden durch Ajax-Aufrufe effizient aktualisiert. Dabei werden nur neue Daten abgerufen, um die Datenübertragung und Serverlast zu reduzieren.

Verarbeitungseffizienz: Es wurde darauf geachtet, dass wichtige Aufgaben wie der E-Mail-Versand und die Datenbankabfragen parallel und effizient ablaufen, um die Leistung der Anwendung zu optimieren.

Watcher-Überwachung: Der Watcher sorgt dafür, dass die Verbindung zur Datenquelle immer aktiv bleibt. Bei Problemen wird der Prozess automatisch neu gestartet, ohne dass Benutzereingriffe nötig sind.

Zugriffssteuerung: Vor der Anmeldung sind nur die Login- und Registrierungsfunktionen verfügbar. Erst nach erfolgreichem Login werden die Karte und die Schiffstabelle angezeigt, sodass nur angemeldete Benutzer auf diese sensiblen Daten zugreifen können.

Schiffsinformationen bei Klick: Ein zusätzlicher Vorteil der Kartenansicht ist, dass der Benutzer beim Klick auf ein Schiff direkt dessen spezifische Informationen (wie Name, MMSI-Nummer und Position) angezeigt bekommt, was die Interaktivität und Benutzerfreundlichkeit erhöht.

5 Datenbankstruktur

Die Anwendung nutzt eine Datenbank, um wichtige Informationen zu speichern. Diese Informationen umfassen Daten zu Schiffen, Benutzern und deren Positionen. Die Daten werden in Tabellen organisiert, die wie Listen aufgebaut sind.

5.1 Userstabelle

Die Userstabelle speichert alle Informationen über die Benutzer, die sich in die Anwendung einloggen. Die wichtigsten Felder in der Tabelle sind:

- **id**: Eine eindeutige Nummer, die jedem Benutzer automatisch zugewiesen wird. Diese Nummer wird verwendet, um jeden Benutzer eindeutig zu identifizieren.
- **firstname**: Der Vorname des Benutzers. Dieser Wert darf nicht leer sein.
- **lastname**: Der Nachname des Benutzers. Dieser Wert darf ebenfalls nicht leer sein.

- **email**: Die E-Mail-Adresse des Benutzers. Sie ist eindeutig, das heißt, keine zwei Benutzer dürfen dieselbe E-Mail-Adresse haben. Die E-Mail-Adresse dient auch als Hauptlogin für den Benutzer.
- **password**: Das Passwort des Benutzers, das sicher und verschlüsselt in der Datenbank gespeichert wird. Dieser Wert darf nicht leer sein.
- **phone**: Die Telefonnummer des Benutzers. Dieses Feld ist optional und kann leer bleiben.
- **created_at**: Das Datum und die Uhrzeit, zu der der Benutzer erstellt wurde. Dieser Wert wird automatisch gesetzt, wenn ein neuer Benutzer hinzugefügt wird.

Erstellung der Userstabelle

```
CREATE TABLE IF NOT EXISTS users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    firstname VARCHAR(255) NOT NULL,
    lastname VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    phone VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

5.2 Session-Tabelle

Die Session-Tabelle speichert Informationen über die Anmeldesitzungen der Benutzer. Jedes Mal, wenn sich ein Benutzer in die Anwendung einloggt, wird eine Sitzung erstellt und in dieser Tabelle gespeichert. Diese Tabelle hilft dabei, festzuhalten, welcher Benutzer sich wann angemeldet hat.

Die wichtigsten Felder in der Session-Tabelle sind:

- **session_id**: Eine eindeutige Kennung für jede Sitzung. Diese Kennung wird verwendet, um jede Sitzung eindeutig zu identifizieren.
- **user_id**: Dies ist die ID des Benutzers, der zur Sitzung gehört. Sie verweist auf die ID des Benutzers in der Benutzertabelle.
- **created_at**: Das Datum und die Uhrzeit, wann die Sitzung gestartet wurde. Dieser Wert wird automatisch gesetzt, wenn die Sitzung erstellt wird.

Wenn ein Benutzer gelöscht wird, werden auch alle zugehörigen Sitzungen automatisch aus der Session-Tabelle entfernt.

Erstellung der Session-Tabelle

```
CREATE TABLE IF NOT EXISTS sessions (
    session_id VARCHAR(64) NOT NULL PRIMARY KEY,
    user_id INT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```

5.3 Schiffstabelle

Die Schiffstabelle speichert Informationen über die Schiffe, die in der Anwendung angezeigt werden. Jedes Schiff hat eine eindeutige Nummer, die sogenannte MMSI (Maritime Mobile Service Identity), die weltweit einzigartig ist. Diese Nummer wird verwendet, um jedes Schiff eindeutig zu identifizieren.

Die wichtigsten Felder in der Schiffstabelle sind:

- **mmsi**: Die eindeutige Identifikationsnummer für jedes Schiff. Diese Nummer ist der Primärschlüssel der Tabelle, das bedeutet, dass jede MMSI nur einmal vorkommen darf.
- **timestamp**: Das Datum und die Uhrzeit der letzten Meldung des Schiffs. Dieser Wert zeigt, wann das Schiff zuletzt eine Positionsmeldung gesendet hat.
- **ship_name**: Der Name des Schiffs. Dieser Wert ist optional und kann leer bleiben.

Erstellung der Schiffstabelle

```
CREATE TABLE IF NOT EXISTS ships (
    mmsi VARCHAR(255) NOT NULL PRIMARY KEY,
    timestamp TIMESTAMP NOT NULL,
    ship_name VARCHAR(255)
);
```

5.4 Positionsmeldungen

Die Tabelle für Positionsmeldungen speichert die aktuellen Positionen der Schiffe. Jedes Mal, wenn ein Schiff eine neue Position sendet, wird diese in der Datenbank gespeichert. Diese Positionsdaten sind wichtig, um die Bewegung der Schiffe auf der Karte anzeigen zu können.

Die wichtigsten Felder in der Tabelle für Positionsmeldungen sind:

- **id**: Eine eindeutige Nummer, die automatisch für jede Positionsmeldung erstellt wird. Dies ist der Primärschlüssel der Tabelle.
- **ship_mmsi**: Die MMSI-Nummer des Schiffs, zu dem die Positionsmeldung gehört. Sie ist ein Fremdschlüssel, der auf die MMSI in der Schiffstabelle verweist.
- **latitude**: Der Breitengrad des Schiffs. Dieser Wert zeigt an, wie weit nördlich oder südlich sich das Schiff befindet.
- **longitude**: Der Längengrad des Schiffs. Dieser Wert zeigt an, wie weit östlich oder westlich sich das Schiff befindet.
- **timestamp**: Das Datum und die Uhrzeit, wann die Positionsmeldung gemacht wurde. Dieser Wert wird automatisch auf den aktuellen Zeitpunkt gesetzt, wenn die Meldung gespeichert wird.

Wenn ein Schiff in der Schiffstabelle gelöscht wird, werden auch alle zugehörigen Positionsmeldungen aus dieser Tabelle automatisch gelöscht.

Erstellung der Positionsmeldungen-Tabelle

```
CREATE TABLE IF NOT EXISTS positions (
    id INT AUTO_INCREMENT PRIMARY KEY,
    ship_mmsi VARCHAR(255) NOT NULL,
    latitude DECIMAL(9,6) NOT NULL,
    longitude DECIMAL(9,6) NOT NULL,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (ship_mmsi) REFERENCES ships(mmsi) ON DELETE CASCADE
);
```

6 Datenüberwachung und -verarbeitung

6.1 Watcher-Skript: New_watcher.sh

Das Watcher-Skript überwacht eine CSV-Datei mit Schiffspositionsdaten und aktualisiert die Datenbank. Es stellt auch sicher, dass der Rhodes-Dienst aktiv bleibt und neu gestartet wird, wenn nötig.

Hier ist der Inhalt des Skripts:

Bash-Skript für Datenüberwachung und -verarbeitungnew-watcher-script

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

rhodescheck=$(cat /usr/lib/cgi-bin/project_2024/rhodes-last.txt)

rhodesvergleich=$(wc -l < /usr/lib/cgi-bin/project_2024/data.csv)

if test "$rhodescheck" -eq "$rhodesvergleich" ; then

  kill "$(cat /usr/lib/cgi-bin/project_2024/rhodes.pid)" &>/dev/null

  nohup ncat -e /usr/lib/cgi-bin/project_2024/readwrite.sh rhodes 8082 &>/dev/null &

  echo "$!" > /usr/lib/cgi-bin/project_2024/rhodes.pid

fi

echo "$rhodesvergleich" > /usr/lib/cgi-bin/project_2024/rhodes-last.txt

source /usr/lib/cgi-bin/project_2024/db_config.sh

CSV_FILE="/usr/lib/cgi-bin/project_2024/data.csv"
TEMP_FILE="/usr/lib/cgi-bin/project_2024/temp_data.csv"

while true; do

    if [ -s "$CSV_FILE" ]; then

        while IFS="|" read -r timestamp typ mmsi spalte4 ship_name spalte6 spalte7
        ↪  spalte8 longitude latitude spalte11 spalte12 spalte13 spalte14 spalte15
        ↪  spalte16 spalte17
```

```
29
30      do
31
32          if [[ "$typ" == "1" || "$typ" == "3" ]]; then
33
34              EXISTING_SHIP=$(mysql -u "$DB_USER" -p"$DB_PASS" -h "$DB_HOST" -D
                ↪  "$DB_NAME" -se "SELECT COUNT(*) FROM ships WHERE mmsi='$mmsi';")
35
36              if [[ "$EXISTING_SHIP" -eq 0 ]]; then
37
38                  SQL_QUERY_SHIP="INSERT INTO ships (mmsi, timestamp, ship_name)
                    ↪  VALUES ('$mmsi', '$timestamp', '$ship_name');"
39
40              else
41
42                  SQL_QUERY_SHIP="UPDATE ships SET timestamp='$timestamp',
                    ↪  ship_name='$ship_name' WHERE mmsi='$mmsi';"
43
44              fi
45
46              mysql -u "$DB_USER" -p"$DB_PASS" -h "$DB_HOST" "$DB_NAME" -e
                ↪  "$SQL_QUERY_SHIP"
47
48              SQL_QUERY_POSITION="INSERT INTO positions (ship_mmsi, latitude,
                ↪  longitude, timestamp) VALUES ('$mmsi', '$latitude',
                ↪  '$longitude', '$timestamp');"
49
50              mysql -u "$DB_USER" -p"$DB_PASS" -h "$DB_HOST" "$DB_NAME" -e
                ↪  "$SQL_QUERY_POSITION"
51
52          elif [[ "$typ" == "18" ]]; then
53
54              latitude=$spalte5
55
56              longitude=$spalte4
57
58              EXISTING_SHIP=$(mysql -u "$DB_USER" -p"$DB_PASS" -h "$DB_HOST" -D
                ↪  "$DB_NAME" -se "SELECT COUNT(*) FROM ships WHERE mmsi='$mmsi';")
59
60              if [[ "$EXISTING_SHIP" -eq 0 ]]; then
61
```

```
62                      SQL_QUERY_SHIP="INSERT INTO ships (mmsi, timestamp, ship_name)
                        ↪  VALUES ('$mmsi', '$timestamp', '');"
63
64                  else
65
66                      SQL_QUERY_SHIP="UPDATE ships SET timestamp='$timestamp' WHERE
                        ↪  mmsi='$mmsi';"
67
68                  fi
69
70                  mysql -u "$DB_USER" -p"$DB_PASS" -h "$DB_HOST" "$DB_NAME" -e
                    ↪  "$SQL_QUERY_SHIP"
71
72                  SQL_QUERY_POSITION="INSERT INTO positions (ship_mmsi, latitude,
                    ↪  longitude, timestamp) VALUES ('$mmsi', '$latitude',
                    ↪  '$longitude', '$timestamp');"
73
74                  mysql -u "$DB_USER" -p"$DB_PASS" -h "$DB_HOST" "$DB_NAME" -e
                    ↪  "$SQL_QUERY_POSITION"
75
76              fi
77
78              tail -n +2 "$CSV_FILE" > "$TEMP_FILE" && mv "$TEMP_FILE" "$CSV_FILE"
79
80          done < "$CSV_FILE"
81
82      else
83
84          sleep 5
85
86          mariadb -u $DB_USER -p$DB_PASS -h $DB_HOST $DB_NAME -e "
87          DELETE FROM positions WHERE timestamp < NOW() - INTERVAL 7 DAY;
88          "
89
90      fi
91
92  done
```

Erklärung: Unser Watcher-Skript überwacht regelmäßig eine CSV-Datei, um Schiffspositionsdaten zu verarbeiten und die Datenbank zu aktualisieren. Es sorgt auch dafür, dass der Rhodes-Dienst kontinuierlich läuft. Bei Bedarf wird der Dienst neu gestartet, wenn keine Datenänderung festgestellt wird. Datenbankverbindungsdaten werden aus einer Konfigurationsdatei geladen. Die CSV-Datei wird kontinuierlich überwacht und verarbeitet, und alte Positionsdaten werden regelmäßig aus der Datenbank gelöscht.

6.2 Cronjobs

Die Cronjobs sorgen dafür, dass die verschiedenen Skripte regelmäßig ausgeführt werden. Hier sind die Cronjob-Einträge:

Cronjob für Watcher-Skriptcronjob-watcher

```
* * * * * /usr/bin/pgrep -f ~/new_watcher.sh || /usr/bin/nohup ~/new_watcher.sh
↪  &>/dev/null &
```

Cronjob für TCP-Email-Servercronjob-email-server

```
* * * * * /usr/bin/pgrep -f ~/tcp_email_server.sh || /usr/bin/nohup
↪  ~/tcp_email_server.sh &>/dev/null &
```

Cronjob für E-Mail-Verarbeitungcronjob-email-processing

```
* * * * * /usr/bin/pgrep -f ~/process_queue.sh || /usr/bin/nohup ~/process_queue.sh
↪  &>/dev/null &
```

Erklärung der Cronjobs: Die Cronjobs stellen sicher, dass die verschiedenen Skripte kontinuierlich laufen. Der erste Cronjob überprüft jede Minute, ob das Watcher-Skript ... new_watcher.sh .... aktiv ist, und startet es bei Bedarf neu. Der zweite Cronjob stellt sicher, dass der TCP-Email-Server-Skript ...tcp$_{e}mail_{s}erver.sh....kontinuierlichE-Mailsempfngt.DerdritteCronjobberwach$
$Mail-Warteschlangeundverarbeiteteingegangene E-Mails.$

7 Webseite

Die Darstellung der Webseite basiert auf einem Akkordeon-Layout. Das bedeutet, dass immer nur ein Abschnitt gleichzeitig geöffnet sein kann, während die anderen geschlossen bleiben. Es gibt drei Abschnitte:

Registrierung und Login: Hier kann sich der Benutzer registrieren oder mit E-Mail und Passwort anmelden.
Karte: Nach erfolgreichem Login wird die Karte sichtbar, auf der die Schiffspositionen angezeigt werden.
Schiffe: Dieser Bereich zeigt eine Liste der Schiffe, die auf der Karte sichtbar sind.

7.1 Startwebseite

Das Akkordeon-Konzept auf unserer Website zeigt, dass alle Gruppen zuerst geschlossen sind, außer den Bereichen Registrierung und Login. Wenn der Benutzer auf eine Gruppe klickt, öffnet sich diese, und der Inhalt wird sichtbar. Andere Gruppen bleiben weiterhin geschlossen. Dadurch wird die Webseite übersichtlicher, da immer nur eine Gruppe auf einmal geöffnet ist.

7.1.1 Darstellung

Abbildung 7.1: Startwebseite

7.1.2 Frontend

JavaScript für Akkordeon script.js

```
window.onload = function() {
    const accordion = document.getElementById('accordion');
    const groups = accordion.getElementsByClassName('group');

    for (let i = 0; i < groups.length; i++) {
```

```
6           groups[i].querySelector('h3').addEventListener('click', function() {
7               for (let j = 0; j < groups.length; j++) {
8                   if (j !== i) {
9                       groups[j].querySelector('.content').style.display = 'none';
10                  }
11              }
12              const content = groups[i].querySelector('.content');
13              content.style.display = content.style.display === 'block' ? 'none' :
                ↪  'block';
14          });
15      }
16  };
```

Der Code wird ausgeführt, wenn die Webseite komplett geladen ist. Das bedeutet, dass der Browser wartet, bis die Seite bereit ist, bevor er diesen Code startet. Zuerst wird das Akkordeon-Element aus dem HTML-Dokument geholt. Das ist der Bereich, der alle Gruppen oder Abschnitte enthält, wie zum Beispiel ”Registrierung und Login”. Dann sucht der Code nach allen Elementen im Akkordeon, die die Klasse ”group”haben. Diese Elemente sind die verschiedenen Abschnitte im Akkordeon.
Der Code benutzt eine Schleife, um durch alle Gruppen im Akkordeon zu gehen und ihnen eine Funktion zu geben. Diese Funktion wird jedes Mal ausgeführt, wenn der Benutzer auf eine Überschrift klickt, also auf einen Abschnitt. Es gibt noch eine zweite Schleife, die dafür sorgt, dass alle anderen Abschnitte, außer dem, auf den geklickt wurde, geschlossen werden. Das bedeutet, dass nur ein Abschnitt auf einmal sichtbar ist.
Wenn der Benutzer auf eine Überschrift klickt, wird der Inhalt des angeklickten Abschnitts ausgewählt. Dann prüft der Code, ob der angeklickte Abschnitt schon offen oder geschlossen ist. Wenn er offen ist, wird er geschlossen. Wenn er geschlossen ist, wird er geöffnet. So wird immer nur ein Bereich zur Zeit angezeigt, damit die Seite übersichtlich bleibt.

7.2 Registrierung-Login

Im Bereich zum Einloggen oder Registrieren muss ein bestehender Benutzer seine E-Mail und das Passwort eingeben, um sich anzumelden. Falls man noch keine Login-Daten hat, soll man sich ein Konto erstellen, indem man die folgenden Daten eingibt: Vorname, Nachname, E-Mail-Adresse, Telefonnummer und ein Passwort. Die Daten sind wichtig, damit der neue Benutzer sicher identifiziert und die Karte und die Tabelle sehen kann. Bei der Eingabe der Daten im Login- und Registrierungsbereich müssen bestimmte Informationen in festgelegten Formaten eingegeben werden.

7.2.1 Darstellung

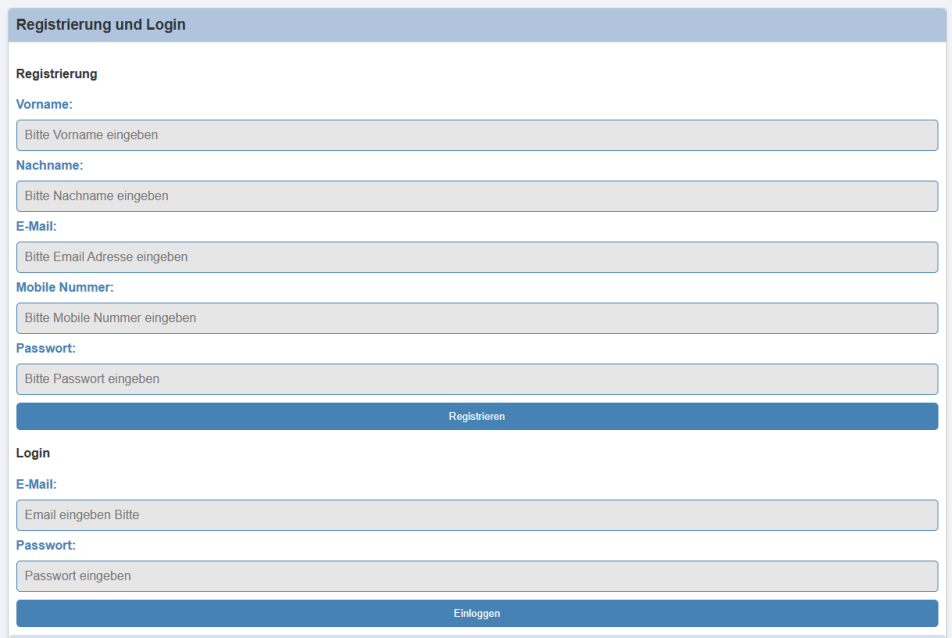


Abbildung 7.2: Registrierung-Login

7.2.2 Frontend

Registrierung

HTML für Registrierung und Login index.html

```
<div id="accordion">
    <div class="group">
        <h3>Registrierung und Login</h3>
        <div class="content">
            <h4>Registrierung</h4>
            <form id="register-form">
                <label for="firstname">Vorname:</label>
                <input type="text" name="firstname" id="firstname"
                ↪  placeholder="Bitte Vorname eingeben" required>

                <label for="lastname">Nachname:</label>
                <input type="text" name="lastname" id="lastname" placeholder="Bitte
                ↪  Nachname eingeben" required>

```

```
                <label for="email">E-Mail:</label>
                <input type="email" name="email" id="email" placeholder="Bitte Email
                ↪  Adresse eingeben" required>

                <label for="phonenumber">Mobile Nummer:</label>
                <input type="text" name="phonenumber" id="phonenumber"
                ↪  placeholder="Bitte Mobile Nummer eingeben" required>

                <label for="password">Passwort:</label>
                <input type="password" name="password" id="password"
                ↪  placeholder="Bitte Passwort eingeben" required>

                <input type="submit" value="Registrieren">
            </form>
        </div>
    </div>
</div>
```

Das <div id=äccordion» ist das Akkordeon, das alle verschiedenen Gruppen oder Abschnitte enthält. In diesem Fall ist es der Bereich ”Registrierung und Login”. Dann gibt es <div class=”group», das ist eine Gruppe im Akkordeon und beinhaltet den Bereich ”Registrierung und Login”.
Die Überschrift für diesen Abschnitt ist <h3>Registrierung und Login</h3>. Darunter befindet sich <div class=”content», das den Inhalt der Gruppe enthält. Hier geht es um das Formular, in dem der Benutzer sich registrieren kann.
Das Formular selbst wird durch <form id=”register-form» erstellt. Es hat mehrere Eingabefelder, in denen der Benutzer seinen Vornamen, Nachnamen, die E-Mail, die mobile Nummer und ein Passwort eingeben muss. Jedes dieser Eingabefelder hat ein placeholder, das ist ein Text, der dem Benutzer zeigt, was er in das Feld eingeben soll. Am Ende gibt es einen ”RegistrierenButton. Wenn der Benutzer darauf klickt, wird das Formular abgeschickt.

JavaScript für Formular-Registrierung script.js

```
const baseURL = '/docker-infra-2024-l-web/cgi-bin/project_2024';

document.getElementById('register-form').addEventListener('submit', function(e) {
    e.preventDefault();
    const formData = new FormData(this);
    const params = new URLSearchParams(formData);
```

```
7
8       let xhr = new XMLHttpRequest();
9       xhr.open('GET', `${baseURL}/register.sh?` + params.toString(), true);
10      xhr.onload = function() {
11          if (xhr.status === 200) {
12              if (xhr.responseText.includes('Registrierung abgeschlossen')) {
13                  alert("Registrierung Abgeschlossen");
14              } else {
15                  alert("Registrierung Fehlgeschlagen");
16              }
17          } else {
18              alert("Registrierung Fehlgeschlagen: " + xhr.responseText);
19          }
20      };
21      xhr.send();
22  });
```

Der Code beginnt mit const baseURL = ’/docker-infra-2024-l-web/cgi-bin/project_2024’;. Das ist die Basis-Adresse, die später im Code benutzt wird. Diese Adresse zeigt auf den Ort, wo das Skript register.sh liegt, das für die Registrierung zuständig ist.
Dann gibt es document.getElementById(’register-form’).addEventListener(’submit’, function(e) ...);. Das fügt dem Registrierungsformular eine Funktion hinzu, die immer dann ausgeführt wird, wenn der Benutzer auf ”Registrieren”klickt und das Formular abschickt.
Der Code e.preventDefault(); sorgt dafür, dass die Seite nicht neu geladen wird, wenn der Benutzer das Formular abschickt. So bleibt der Benutzer auf der gleichen Seite, und die Registrierung läuft im Hintergrund ab.
Mit const formData = new FormData(this); werden alle Daten gesammelt, die der Benutzer in das Formular eingetragen hat. this bezieht sich auf das Formular, das gerade abgeschickt wurde.

Dann wird mit const params = new URLSearchParams(formData); die gesammelten Daten in ein Format umgewandelt, das in einer URL verwendet werden kann. So können die Daten an den Server geschickt werden.

Mit let xhr = new XMLHttpRequest(); wird ein neues Objekt erstellt, das dafür da ist, Daten an den Server zu senden und eine Antwort zu bekommen, ohne die Seite neu zu laden. Das ist so, als würde man eine Nachricht verschicken und auf eine Antwort warten, ohne wegzulaufen.

Der Code xhr.open(’GET’, $baseURL/register.sh¿ + params.toString(), true);bereitet die Anfrage vor, die an den Server geschickt wird. Die Daten aus dem Formular werden dabei an das Skriptregister.shgesendet, undGET‘ zeigt, dass wir Daten senden möchten.

Wenn der Server antwortet, wird die Funktion xhr.onload = function() ... ; ausgeführt. Sie entscheidet, was passieren soll, je nachdem, ob die Registrierung erfolgreich war oder nicht.

Login

HTML für das Login-Formular index.html

```
<div id="accordion">
    <div class="group">
        <h3>Registrierung und Login</h3>
        <div class="content">
            <h4>Login</h4>
            <form id="login-form">
                <label for="emailLogin">E-Mail:</label>
                <input type="email" id="emailLogin" name="emailLogin"
                ↪  placeholder="Email eingeben Bitte" required>

                <label for="passwordLogin">Passwort:</label>
                <input type="password" id="passwordLogin" name="passwordLogin"
                ↪  placeholder="Passwort eingeben" required>

                <input type="submit" value="Einloggen">
            </form>
        </div>
    </div>
</div>
```

Der Code hier ist fast gleich wie bei der Registrierung. Ein Abschnitt ist für den Login, mit der Überschrift ”Login”. Der Benutzer gibt seine E-Mail und sein Passwort ein.
Für die E-Mail gibt es ein Feld, das sagt Ëmail eingeben Bitte”. Das Passwortfeld zeigt Punkte an, damit keiner das Passwort sieht. Unten ist ein ËinloggenButton. Wenn der Benutzer darauf klickt, versucht er sich einzuloggen.

JavaScript für die Login-Verarbeitung script.js

```
document.getElementById('login-form').addEventListener('submit', function(event) {
    event.preventDefault();
    const formData = new FormData(this);
    const params = new URLSearchParams(formData);

    let xhr = new XMLHttpRequest();
```

```
    xhr.open('GET', `${baseURL}/login.sh?` + params.toString(), true);
    xhr.onload = function() {
        if (xhr.status === 200) {
            if (xhr.responseText.includes('Login erfolgreich')) {
                alert("Login erfolgreich!");
                document.getElementById('username').textContent =
                ↪  formData.get('emailLogin');
                document.getElementById('user-info').classList.remove('hidden');
                document.getElementById('map-group').classList.remove('disabled');
                document.getElementById('ships-group').classList.remove('disabled');
            } else {
                alert("Login Fehlgeschlagen");
            }
        } else {
            alert("Login Fehlgeschlagen: " + xhr.responseText);
        }
    };
    xhr.send();
});
```

Der Code wird ausgeführt, wenn der Benutzer das Login-Formular absendet. Er fügt dem Formular eine Funktion hinzu, die im Hintergrund arbeitet. Normalerweise lädt die Seite neu, wenn man ein Formular absendet, aber hier wird das verhindert, damit der Benutzer auf der gleichen Seite bleibt. Das passiert durch event.preventDefault().
Der Code sammelt die Daten, die der Benutzer ins Formular eingegeben hat, wie E-Mail und Passwort, und speichert sie in einem FormData-Objekt. Dann werden diese Daten in ein Format umgewandelt, das in der URL verwendet werden kann. Das macht const params = new URLSearchParams(formData).
Jetzt erstellt der Code ein neues Objekt, let xhr = new XMLHttpRequest(), um die Anfrage an den Server zu senden. Es bereitet eine GET-Anfrage vor, die die Login-Daten an ein Skript auf dem Server (login.sh) schickt. Danach wird eine Funktion definiert, die ausgeführt wird, wenn der Server eine Antwort schickt.
Der Code überprüft, ob die Antwort vom Server erfolgreich war, also ob der Statuscode 200 ist, was ÖK"bedeutet. Er schaut auch, ob die Antwort "Login erfolgreichënthält. Wenn ja, zeigt er dem Benutzer eine Erfolgsmeldung an und macht bestimmte Bereiche der Webseite sichtbar, wie die Karte und die Schiffsliste. Am Ende sendet der Code die Anfrage mit den eingegebenen Login-Daten an den Server.

7.2.3 Backend

Registrierung

Bash-Skript für die Registrierung register.sh

```
#!/bin/bash

echo "Content-type: text/plain"
echo ""
source db_config.sh
IFS='&' read -r -a pairs <<< "$QUERY_STRING"
for pair in "${pairs[@]}"; do
    IFS='=' read -r key value <<< "$pair"
    key=$(echo "$key" | sed 's/%20/ /g' | sed 's/+/ /g')
    value=$(echo "$value" | sed 's/%20/ /g' | sed 's/+/ /g' | sed 's/%40/@/g')

    case $key in
        firstname) firstname=$value ;;
        lastname) lastname=$value ;;
        email) email=$value ;;
        password) password=$value ;;
        phonenumber) phonenumber=$value ;;
    esac
done
password_hash=$(echo -n "$password" | sha256sum | sed 's/ .*//')
insert_result=$(mysql -h "$DB_HOST" -u "$DB_USER" -p"$DB_PASS" "$DB_NAME" -sse "
INSERT INTO users (firstname, lastname, email, password, phone)
VALUES ('$firstname', '$lastname', '$email', '$password_hash', '$phonenumber');
" 2>&1)

if [ $? -eq 0 ]; then
    echo "success: Registrierung abgeschlossen"
else
    echo "error: Fehler bei der Registrierung - $insert_result"
fi
```

wenn sich ein Benutzer neu auf der Webseite registrieren möchte.
Es nimmt die Daten aus dem Registrierungsformular, speichert sie sicher in der Datenbank und gibt eine Rückmeldung an den Benutzer.
Das Skript holt sich die Daten, die der Benutzer ins Registrierungsformular eingegeben hat. Es

zerlegt diese Daten in kleinere Teile. Dafür benutzt es IFS, um die Daten in kleine Paare wie SSchlüssel=Wertäufzuteilen .
Dann wird der Code etwas aufräumen, um sicherzustellen, dass die Daten korrekt gelesen werden, indem er einige Zeichen wie %20 oder in Leerzeichen und umwandelt.
Das Skript speichert diese Daten in Variablen (firstname, lastname, email, password, phonenumber) , die später verwendet werden .Das Passwort wird in einen ”Hashümgewandelt, damit es sicher gespeichert werden kann.
Ein Hash ist eine Art verschlüsselter Code, der das Passwort schützt. Dann fügt das Skript die neuen Daten des Benutzers in die Datenbank ein. Es benutzt dafür den Befehl mysql und wird ein neuer Benutzer in der Datenbank erstellt.
Am Ende überprüft das Skript, ob das Einfügen der Daten erfolgreich war. Wenn alles geklappt hat, gibt es eine Erfolgsmeldung zurück und wenn etwas schiefgegangen ist, bekommt der Benutzer eine Fehlermeldung.

Login Bash-Skript für die Anmeldung login.sh

```
#!/bin/bash

echo "Content-Type: text/plain"
source db_config.sh

IFS='&' read -r -a pairs <<< "$QUERY_STRING"
for pair in "${pairs[@]}"; do
    IFS='=' read -r key value <<< "$pair"
    key=$(echo "$key" | sed 's/%20/ /g' | sed 's/+/ /g')
    value=$(echo "$value" | sed 's/%20/ /g' | sed 's/+/ /g' | sed 's/%40/@/g')

    case $key in
        emailLogin) emailLogin=$value ;;
        passwordLogin) passwordLogin=$value ;;
    esac
done

password_hash=$(echo -n "$passwordLogin" | sha256sum | sed 's/ .*//')
result=$(mysql -h "$DB_HOST" -u "$DB_USER" -p"$DB_PASS" "$DB_NAME" -sse "
SELECT id, firstname FROM users WHERE email='$emailLogin' AND
↪  password='$password_hash';
")

if [ "$result" ]; then
  user_id=$(echo $result | sed 's/\([0-9]*\).*/\1/' )
  session_id=$(pwgen 40 1)
```

```
26
27      insert_session=$(mysql -h "$DB_HOST" -u "$DB_USER" -p"$DB_PASS" "$DB_NAME" -sse
        ↪  "
28      INSERT INTO sessions (session_id, user_id) VALUES ('$session_id', '$user_id');
29      ")
30
31      if [ $? -eq 0 ]; then
32          echo "Set-Cookie: session_id=$session_id; Path=/docker-infra-2024-l-web/;
            ↪  HttpOnly"
33          echo ""
34          email_subject="Hello from there :)"
35          email_body="User $emailLogin hat sich erfolgreich angemeldet."
36          echo -e "$emailLogin, $email_subject, $email_body" | ncat localhost 4000
37          echo "Login erfolgreich"
38      else
39          echo ""
40          echo "Fehler bei der Erstellung der Sitzung"
41      fi
42  else
43      echo ""
44      echo "Fehler bei der Anmeldung"
45  fi
```

Das Bash-Skript wird auf dem Server ausgeführt, wenn sich ein Benutzer auf der Webseite einloggen möchte. Es überprüft die Login-Daten des Benutzers und kümmert sich darum, dass der Benutzer eingeloggt wird.
Das Skript bekommt die Daten vom Login-Formular, also die E-Mail und das Passwort. Es teilt die Daten auf, damit es sie verarbeiten kann. Es benutzt dafür den IFS (Interne Feldtrenner), um die Daten in kleine Teile zu zerlegen: `IFS='' read -r -a pairs <<< "$QUERY_STRING"`.
Das Passwort wird nicht einfach so gespeichert. Stattdessen wird es in einen speziellen ”Codeümgewandelt, den man nicht mehr zurück in das eigentliche Passwort umwandeln kann. Dieser Code heißt ”Hash: `password_hash=$(echo -n "$passwordLogin sha256sum | sed 's/ .*//')`.
Das Passwort wird so sicher gemacht, damit niemand es einfach auslesen kann.
Das Skript schaut jetzt in der Datenbank nach, ob es einen Benutzer mit dieser E-Mail und diesem Passwort-Hash gibt. Wenn ein Benutzer mit diesen Daten gefunden wird, geht es weiter, ansonsten gibt es eine Fehlermeldung.
Wenn die Login-Daten korrekt sind, erstellt das Skript eine neue SSessionöder Sitzung. Eine Sitzung ist wie ein Ausweis, der zeigt, dass der Benutzer eingeloggt ist. Das Skript benutzt dafür den Befehl `pwgen`, um eine zufällige und sichere Sitzungs-ID zu erstellen: `session_id=$(pwgen 40 1)`. Diese ID wird dann in der Datenbank gespeichert, damit man später weiß, welcher

Benutzer eingeloggt ist.
Ein ”Cookieïst eine kleine Datei, die auf dem Computer des Benutzers gespeichert wird. Damit kann die Webseite später erkennen, dass der Benutzer schon eingeloggt ist. Das Skript erstellt so ein Cookie und schickt es an den Browser: `echo SSet-Cookie: session_id=$session_id; Path=/docker-infra-2024-l-web/; HttpOnly"`. Das Cookie wird sicher gemacht, damit niemand es einfach auslesen oder benutzen kann.
Am Ende gibt das Skript eine Nachricht an den Benutzer zurück. Wenn alles gut gegangen ist, steht dort ”Login erfolgreich”. Wenn etwas schiefgelaufen ist, steht dort ”Fehler bei der Anmeldung”.

7.3 Die Karte

7.3.1 Darstellung

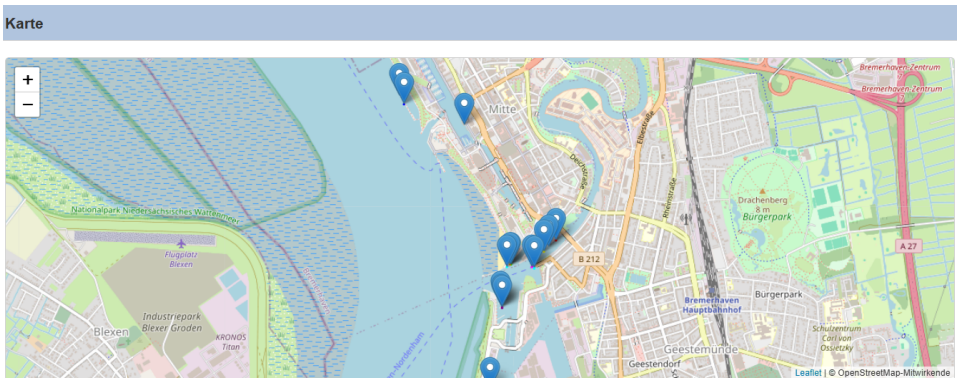


Abbildung 7.3: Die Karte

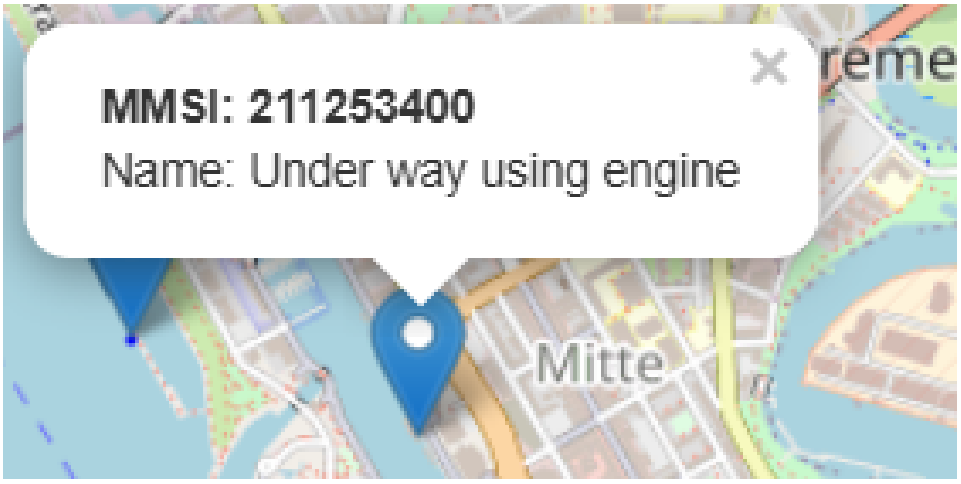


Abbildung 7.4: Informationen eines Schiffes auf Karte

7.3.2 Frontend

Die Karte zeigt nicht nur die aktuelle Position der Rhodes-Schiffe in Echtzeit an, sondern auch den Weg, den die Schiffe in der letzten Stunde zurückgelegt haben. Dieser Weg wird in verschiedenen Farben dargestellt, damit man die Bewegungen der Schiffe besser nachvollziehen kann. Jede Farbe steht für einen bestimmten Abschnitt des Weges, damit man sehen kann, wo das Schiff zu welcher Zeit war. Die Daten kommen als json-Format aus Bash-Skript ... get-postitions.sh ... und werden regelmäßig aktualisiert, um die Karte immer auf dem neuesten Stand zu halten.

Hier ist der HTML-Code für die Karte:

HTML für Karte index.html

```
<div class="group disabled" id="map-group">
    <h3>Karte</h3>
    <div class="content">
        <div id="map" style="height:400px;"></div>
    </div>
</div>
```

Die Höhe der Karte wurde auf 400 Pixel festgelegt, damit sie gut sichtbar ist.

Im Frontend wird die Karte mithilfe der Leaflet-Bibliothek initialisiert. Sobald der Benutzer den Kartenbereich öffnet, wird eine Verbindung zu einem Kartenserver hergestellt und die Basiskarte geladen. Die Schiffsdaten werden durch wiederholte Abfragen (alle 1.000 Millisekunden) vom

Rhodes-Server abgerufen. Diese Daten besitzen die Position und den Kurs der Schiffe, die in der letzten Stunde aktiv waren.

Lesen von JSON-Daten mit AJAX Im Backend werden die Schiffsdaten von Rhodes über eine AJAX-Anfrage abgerufen. Die Daten werden aus der Datenbank geholt und im JSON-Format geschickt, die dann verarbeitet und auf der Karte angezeigt werden. Diese Daten besitzen Informationen wie die MMSI, Breiten- und Längengrade sowie den Zeitstempel der Schiffe.

JavaScript zum Abrufen von Schiffsdaten script.js

```
function fetchShipPositions() {
    let xhr = new XMLHttpRequest();
    xhr.open('GET',
    ↪  `${baseURL}/get_positions.sh?last_received_id=${lastReceivedId}`, true);

    xhr.onload = function() {
        if (xhr.status === 200) {
            const data = JSON.parse(xhr.responseText);
            if (data.length > 0) {
                lastReceivedId = data[data.length - 1].id;
                processData(data);
            }
        }
    };

    xhr.onerror = function() {
        console.error('Fehler beim Abrufen der Schiffsdaten.');
    };

    xhr.send();
}
```

Die JSON-Daten werden mit der Funktion `processData()` verarbeitet, um die Schiffspositionen auf der Karte darzustellen und ihre Bewegungen zu verfolgen. Die Daten werden in Gruppen (nach MMSI) aufgeteilt, und der Weg des Schiffes wird auf der Karte als farbige Linie angezeigt, obwohl wir Probleme hatten.

7.4 Schiffstabelle

Die Tabelle zeigt die aktuellen Daten, die Schiffe an, wie MMSI, Name, Breiten- und Längengrad sowie den Zeitstempel. Die Daten kommen vom Server und werden regelmäßig aktualisiert, damit die neuesten Informationen in der Tabelle erscheinen, genau wie auf dem Bild steht.

7.4.1 Darstellung

Schiffe

MMSI	Name	Breite	Länge	Zeitstempel
218709000	Moored	53.53818	8.58276	2024-09-14 18:04:44
211590050	Under way using engine	53.52873	8.57569	2024-09-14 18:03:11
245586000	Under way using engine	53.54678	8.56654	2024-09-14 18:03:10
211593560	Under way using engine	53.55151	8.56574	2024-09-14 18:03:07
211228220	Under way using engine	53.52500	8.57801	2024-09-14 18:03:01
211536810	Under way using engine	53.53646	8.57748	2024-09-14 18:03:01
211253400	Under way using engine	53.54547	8.57292	2024-09-14 18:03:00
211225920	Moored	53.53744	8.58135	2024-09-14 18:02:55
211599340	Under way using engine	53.53637	8.58037	2024-09-14 18:02:44
211595610	Under way using engine	53.53658	8.58030	2024-09-14 18:02:42
211540110	Under way using engine	53.53663	8.57790	2024-09-14 18:02:37
211172030	Moored	53.53398	8.57687	2024-09-14 18:02:32
211233660	Not defined	53.53662	8.57778	2024-09-14 18:02:32

Abbildung 7.5: Schiffstabelle

7.4.2 Frontend

Im Frontend wird eine HTML-Tabelle verwendet. Diese Tabelle wird mit AJAX regelmäßig mit den neuen Schiffsdaten gefüllt. Wenn neue Daten ankommen, werden sie in die Tabelle eingefügt und sortiert. Hier ist ein Beispielcode, der die Tabelle darstellt:

HTML-Tabelle für Schiffspositionen index.html

```
<table id="ship-table">
    <thead>
        <tr>
            <th>MMSI</th>
            <th>Name</th>
            <th>Breite</th>
            <th>Länge</th>
            <th>Zeitstempel</th>
```

```
        </tr>
    </thead>
    <tbody id="ship-table-body">
    </tbody>
</table>
```

Mit AJAX holen wir regelmäßig neue Daten vom Server, ohne die gesamte Seite neu laden zu müssen. Die Daten werden dann in die Tabelle eingefügt.

AJAX-Funktion zum Abrufen der Schiffspositionen script.js

```
function fetchShipPositions() {
    let xhr = new XMLHttpRequest();
    xhr.open('GET', '/get_positions.sh', true);
    xhr.onload = function() {
        if (xhr.status === 200) {
            const data = JSON.parse(xhr.responseText);
            updateTable(data);
        }
    };
    xhr.send();
}
```

Daten in die Tabelle einfügen Sobald die Daten mit AJAX abgerufen wurden, wird die Tabelle aktualisiert. Hier ist die Funktion, die die neuen Daten in die Tabelle einfügt:

JavaScript-Funktion zur Aktualisierung der Tabelle..

```
function updateTable(data) {
    const tableBody = document.getElementById('ship-table-body');
    tableBody.innerHTML = ''; // Löscht alte Daten

    data.forEach(ship => {
        const row = document.createElement('tr');
        row.innerHTML = `
            <td>${ship.mmsi}</td>
            <td>${ship.ship_name || 'Nicht definiert'}</td>
            <td>${ship.latitude.toFixed(5)}</td>
```

```
            <td>${ship.longitude.toFixed(5)}</td>
            <td>${ship.timestamp}</td>
        `;
        tableBody.appendChild(row);
    });
}
```

7.4.3 Backend

Bash-Skript für Datenabfrage get-positions.sh

```
#!/bin/bash
last_received_id=${1:-0}
echo "Content-type: application/json"
echo ""
source db_config.sh
SQL_QUERY="
SELECT p.id, s.mmsi, s.ship_name, p.latitude, p.longitude, p.timestamp
FROM positions p
INNER JOIN ships s ON p.ship_mmsi = s.mmsi
WHERE p.ship_mmsi IN (
    SELECT DISTINCT p2.ship_mmsi
    FROM positions p2
    WHERE p2.timestamp >= NOW() - INTERVAL 5 MINUTE
)
AND p.timestamp >= NOW() - INTERVAL 1 HOUR
AND p.id > $last_received_id
ORDER BY p.id;
"
RESULT=$(mysql -h "$DB_HOST" -u "$DB_USER" -p"$DB_PASS" -D "$DB_NAME" -B -e
↪  "$SQL_QUERY")
if [ -z "$RESULT" ]; then
    echo "[]"
    exit 0
fi
```

```
24
25 RESULT=$(echo "$RESULT" | sed -e '1d')
26 echo "["
27 first_record=true
28 while IFS=$'\t' read -r id mmsi ship_name latitude longitude timestamp; do
29     if [ -z "$mmsi" ] || [ -z "$latitude" ] || [ -z "$longitude" ]; then
30         continue
31     fi
32     if [ "$first_record" = false ]; then
33         echo ","
34     fi
35     first_record=false
36     echo -n "  {"
37     echo -n "\"id\": \"$id\","
38     echo -n "\"mmsi\": \"$mmsi\","
39     echo -n "\"ship_name\": \"$ship_name\","
40     echo -n "\"latitude\": $latitude,"
41     echo -n "\"longitude\": $longitude,"
42     echo -n "\"timestamp\": \"$timestamp\""
43     echo -n "}"
44 done <<< "$RESULT"
45 echo
46 echo "]"
```

Unser Skript ist ein wichtiger Teil unseres Projekts, da es aktuelle Schiffspositionsdaten aus der Datenbank abruft und als JSON-Format ausgibt. Dieses Format nutzen wir, um die Schiffspositionen auf unserer Website oder in einer Anwendung darzustellen.
Das Skript sucht nach Schiffen, die in den letzten 5 Minuten aktiv waren und deren Positionsdaten in der letzten Stunde aktualisiert wurden. Es gibt die ID, den Schiffsnamen, die Position (Breitengrad und Längengrad) sowie den Zeitstempel zurück.
Diese Informationen helfen uns, die Positionen der Schiffe in Echtzeit anzuzeigen und unser System auf dem neuesten Stand zu halten. Wenn keine neuen Daten gefunden werden, zeigt das Skript ein leeres Ergebnis.

7.5 Abmelden

Im oberen rechten Bereich des Frontends gibt es einen ’Abmelden’-Button. Dieser Button ist dafür da, damit sich der Benutzer sicher vom System abmelden kann.

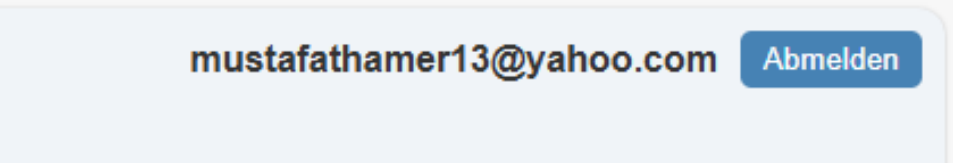


Abbildung 7.6: AbmeldeButton

7.5.1 Frontend

JavaScript für Abmeldung script.js

```
1  document.getElementById('logout-button').addEventListener('click',
   ↪  function() {
2      let xhr = new XMLHttpRequest();
3      xhr.open('GET', `${baseURL}/logout.sh`, true);
4      xhr.onload = function() {
5          if (xhr.status === 200) {
6              if (xhr.responseText.includes('Abmeldung erfolgreich')) {
7                  alert("Sie wurden erfolgreich abgemeldet.");
8                  document.getElementById('user-info').classList.add('hidden'
                   ↪  );
9                  document.getElementById('map-group').classList.add('disable
                   ↪  d');
10                 document.getElementById('ships-group').classList.add('disab
                   ↪  led');
11
12                 document.getElementById('username').textContent = '';
13                 for (let i = 1; i < groups.length; i++) {
14                     groups[i].getElementsByClassName('content')[0].style.di
                       ↪  splay =
                       ↪  'none';
15                 }
16                 groups[0].getElementsByClassName('content')[0].style.displa
                   ↪  y =
                   ↪  'block';
```

```
17              } else {
18                  alert("Fehler bei der Abmeldung: " + xhr.responseText);
19              }
20          } else {
21              alert("Fehler bei der Abmeldung: " + xhr.responseText);
22          }
23      };
24      xhr.send();
25  });
```

Dieser Code meldet den Benutzer ab, wenn auf den Logout-Button geklickt wird. Wenn der Button gedrückt wird, schickt der Code eine Anfrage an den Server, um das Logout-Skript (logout.sh) auszuführen.

Wenn die Abmeldung erfolgreich ist, bekommt der Benutzer eine Nachricht, dass er abgemeldet wurde. Teile der Webseite, die nur für angemeldete Benutzer sichtbar sind, werden ausgeblendet, wie die Benutzerinformationen und bestimmte Gruppen.

Wenn es einen Fehler gibt, wird dem Benutzer eine Fehlermeldung angezeigt.

Kurz gesagt: Der Code meldet den Benutzer ab und passt die Webseite entsprechend an. Bei Fehlern gibt es eine Warnung.

7.5.2 Backend

Das Abmelden-Skript verarbeitet die Abmeldung des Benutzers und entfernt die entsprechende Sitzung aus der Datenbank.

Session-Verarbeitung Zunächst wird die Session-ID aus dem Cookie extrahiert, das im HTTP-Header mitgesendet wird. Falls eine Session-ID vorhanden ist, wird sie zur Identifikation der Sitzung verwendet.

Bash-Skript zur Extraktion der Session-ID logout.sh

```
1  cookieline=$(echo "$HTTP_COOKIE" | tr ";" "\n" | grep "^session_id=")
2  session_id=$(echo "$cookieline" | cut -d "=" -f 2)
```

Löschen der Session Die Session wird aus der MySQL-Datenbank gelöscht. Wenn das erfolgreich ist, wird das Session-Cookie auf einen ungültigen Status gesetzt.

Bash-Skript zum Löschen der Session logout.sh

```
mysql -h "$DB_HOST" -u "$DB_USER" -p"$DB_PASS" "$DB_NAME" -e "
DELETE FROM sessions WHERE session_id = '$session_id';
"

echo "Set-Cookie: session_id=; Path=/docker-infra-2024-l-web/project_2024/;
↪  Expires=Thu, 01 Jan 1970 00:00:00 GMT; HttpOnly"
```

Falls keine gültige Sitzung gefunden wird, gibt das Skript eine Fehlermeldung aus.

8 Arbeitsweise mit GIT

8.1 Was ist GIT?

Für unser Projekt benutzten wir Git als Tool, das uns dabei unterstützt, Änderungen an unserem Code nachzuverfolgen und gemeinsam an einem Projekt zu arbeiten.
Es ermöglicht jedem Entwickler, eine eigene Kopie des Projekts zu haben, an der man unabhängig arbeiten kann, ohne den Fortschritt anderer zu stören.
Durch Funktionen wie Branches (zum Aufteilen von Aufgaben) und Merges (zum Zusammenführen von Änderungen) wird der Entwicklungsprozess einfacher und strukturierter.

Ein großer Vorteil von Git ist, dass es dezentral arbeitet. Das heißt in unserem Fall, dass wir keinen zentralen Server brauchen, sondern jeder hat das komplette Projekt lokal auf seinem Rechner oder in der VM.
So können wir jederzeit Änderungen zurückverfolgen, alte Versionen wiederherstellen und Dateien miteinander vergleichen.
Git lässt sich über einfache Befehle in der Kommandozeile steuern und funktioniert nicht nur mit Textdateien, sondern auch mit anderen Dateitypen wie Bildern.

Für unser Projekt war Git extrem nützlich, weil es uns ermöglicht hat, parallel zu arbeiten und trotzdem den Überblick über alle Änderungen zu behalten.
So konnten wir effizient zusammenarbeiten und das Projekt Schritt für Schritt voranbringen.

Folgende Schritte waren für uns wichtig bei der Arbeit mit GIT, wenn wir eine Datei geändert haben:

1. Wenn wir an unserem Projekt gearbeitet und Dateien erstellt oder geändert haben, müssen diese GIT mitgeteilt werden, damit es die Änderungen verfolgen kann. Um eine bestimmte Datei hinzuzufügen, benutzen wir:

```
git add datei_name
```

Listing 8.1: Hinzufügen einer Datei

   Wenn wir beispielsweise mehrere oder alle Dateien auf einmal hinzufügen wollten, die geändert wurden, verwendeten wir:

```
git add *
```

Listing 8.2: Hinzufügen aller Dateien

   Mit diesem Befehl werden alle geänderten Dateien, die sich im Projektordner befinden, zur nächsten Speicherung (Commit) vorbereitet. Nachdem die Änderungen hinzugefügt wurden, müssen sie gespeichert werden. Das nennt man ”Commit”. Dabei hinterlassen wir eine kurze Beschreibung, die erklärt, was wir geändert haben. So bleibt für das Team nachvollziehbar, was gemacht wurde und wer es gemacht hat. Der Befehl sieht so aus:

```
git commit -m "Beschreibung der Änderungen"
```

Listing 8.3: Commit-Nachricht

   Ein Beispiel für eine Commit-Nachricht könnte sein: `git commit -m "Fehlerbehebung in der Login-Funktion"`.

2. Sobald wir mit den Änderungen zufrieden sind und die mit dem Team diskutiert haben, möchten wir sie mit dem gesamten Team teilen. Dafür müssen die Änderungen in das zentrale Repository auf dem Server hochgeladen werden. Das tun wir mit:

```
git push
```

Listing 8.4: Änderungen hochladen

Jetzt sind unsere Änderungen auf dem Server und alle im Team können darauf zugreifen. Doch bevor wir mit der Arbeit beginnen, sollten wir immer sicherstellen, dass wir die neuesten Änderungen von anderen Teammitgliedern haben. Dafür holen wir uns die aktuelle Version des Projekts vom Server mit:

```
1 git pull
```

Listing 8.5: Änderungen vom Server holen

Das stellt sicher, dass wir mit der aktuellen Version arbeiten und keine Konflikte mit den Änderungen anderer entstehen. Um zu sehen, welche Dateien wir geändert haben und welche schon für den nächsten Commit bereit sind, nutzen wir den Befehl:

```
1 git status
```

Listing 8.6: Status überprüfen

Das gibt uns einen Überblick über den aktuellen Stand der Dinge: Welche Dateien wurden geändert, was wurde schon zu Git hinzugefügt, und was muss noch gespeichert werden. Das ist außerdem wichtig, wenn Sachen nicht richtig bearbeitet wurden. Für eine Übersicht über alle bisherigen Änderungen im Projekt haben wir:

```
1 git log
```

Listing 8.7: Änderungsverlauf anzeigen

Das zeigt uns eine Liste aller Commits, wann sie gemacht wurden, wer sie gemacht hat, und welche Nachricht dabei hinterlassen wurde. So können wir die gesamte Projektentwicklung nachvollziehen. Hierfür haben wir, wie gefordert, eine Tabelle mit unseren Logs erstellt:

Diese Arbeitsweise haben wir auf infra-2024-l verwendet, um als Team effizient an unserem Projekt zu arbeiten. Jeder hatte seine eigenen Arbeitskopien, konnte unabhängig arbeiten, und durch Git wurde sichergestellt, dass alle Änderungen sauber und gut nachvollziehbar ins Projekt integriert wurden.

Tabelle 8.1: Commit-Logs

Autor	Commit-Nachricht	Beschreibung
Kasem Rashrash	JSON-Skript erstellt	Skript zur Verarbeitung und Ausgabe von Daten im JSON-Format erstellt.
Kasem Rashrash	Ausortieren der CGI-Skripte ausgeführt	Bereinigung und Optimierung der CGI-Skripte im Projekt.
Ahmad Al Mohammad	Register script erstellt	Skript zum Registrieren hinzugefügt.
Kasem Rashrash	AJAX für Schiffe erstellt	AJAX-Funktionalitäten hinzugefügt, um Schiffspositionen dynamisch zu laden.
Musthfa Thamer	Neue Veränderungen	Änderungen und Fehlerbehebungen in Skripte vorgenommen.
Shiwan Ali Shahen	Schiffe auf die Karte angezeigt	Schiffe wurden erfolgreich auf der Leaflet-Karte visualisiert.
Ahmad Almohammad	Initial setup for frontend	Frontend mit HTML, CSS und JavaScript eingerichtet (Akkordion).
Kasem Rashrash	5 Min-Skript+import-MySQL-Datenbank	Skript zur Datenaktualisierung alle 5 Minuten sowie Import der Daten in MySQL erstellt.
Ahmad Al Mohammad	Add login last-test script	Ein finales Test-Skript für das Login-System hinzugefügt.
Shiwan Ali Shahen	Leaflet-Test erstellt	Erster Test mit Leaflet zur Anzeige von Schiffen basierend auf einer CSV-Datei durchgeführt.
Kasem Rashrash	Vorgehen vom Watcher	Skript zur Überwachung und Automatisierung von Prozessen im Projekt implementiert.
Mustafa Thamer	CSS-Style angepasst	Designanpassungen für die Website / ...
Shiwan Ali Shahen	Leaflet-Marker und Pop-up	Marker und Popup-Funktionen angepasst, um Schiffsinformationen korrekt anzuzeigen.
Mustafa Thamer	Filter-Skripte für Schiffstypen	Die Schiffstypen 1, 3, 18 von der data.csv werden mithilfe vom Skripten gefiltert.

9 Tests and Monitoring

9.1 Schnittstelle zur Datenbank testen

Unsere Anwendung hat eine eigenständige Schnittstelle zur Datenbank, in der Schiffe und ihre Positionen gespeichert werden.
Diese Schnittstelle kann unabhängig getestet werden, indem wir überprüfen, ob Daten korrekt importiert, gespeichert und abgerufen werden.
Wenn die Datenbank die Daten nicht richtig importiert, wird im Frontend keine aktuelle Anzeige der Schiffe und ihrer Positionen erfolgen.
Der AJAX-Aufruf im Frontend zeigt nur die neuesten Daten an, die über diese Schnittstelle abgerufen werden.
Wir testeten mehrfach immer wieder den AJAX-Aufruf, als keine Daten in die Datenbank importiert wurden und wussten sofort, woran das Problem lag und zwar, dass die Daten nicht direkt in die Datenbank importiert wurden.

9.2 Watcher-Komponente testen

Der Watcher überwacht die Verbindung zu Rhodes und ist ebenfalls als eigenständige Komponente testbar.
Zahlreiche Tests haben gezeigt, dass der Watcher ohne Crontab nach einem Neustart nicht automatisch startet.
Mit einem Crontab-Eintrag wird jedoch sichergestellt, dass der Watcher bei jedem Neustart wieder aktiviert wird.
Eingesetzte falsche Pfade bei uns führten oft dazu, dass der Watcher nicht ausgeführt wurde, was leicht durch Log-Überprüfung behoben werden könnte.

9.3 E-Mail-Simulation und Warteschlange testen

Das Worker-Skript ist dafür verantwortlich, E-Mail-Aufträge zu verarbeiten, die in einem Ordner gespeichert sind. Dieser Ordner wird als Warteschlange verwendet. Das Skript läuft in einer Endlosschleife, in der es regelmäßig nach neuen E-Mails sucht. Sobald es eine Textdatei in diesem Ordner findet, liest es den Inhalt der Datei. Jede Datei enthält drei Informationen, die durch Kommata getrennt sind: die E-Mail-Adresse des Empfängers, den Betreff der E-Mail und den Text der E-Mail.
Nachdem diese Informationen ausgelesen wurden, führt das Skript ein anderes Skript aus, das das Senden der E-Mail simuliert. Wenn dieses Skript erfolgreich ist, wird die bearbeitete Datei in einen separaten Ordner verschoben, der für verarbeitete E-Mails vorgesehen ist. Sollte jedoch ein Fehler auftreten, wird die Datei in einen Fehlerordner verschoben, und der Fehler wird in einer Protokolldatei festgehalten. Zwischen der Verarbeitung jeder E-Mail gibt es eine kurze Pause von 0,3 Sekunden, um sicherzustellen, dass das System nicht überlastet wird. Außerdem wird nach jeder Runde, in der nach neuen E-Mails gesucht wurde, eine Sekunde gewartet, bevor das Skript wieder mit der Suche beginnt.

Das TCP-Server-Skript ist dafür zuständig, Aufträge von anderen Diensten entgegenzunehmen. Es wartet in einer Endlosschleife auf Nachrichten, die über das Netzwerk gesendet werden. Der Server hört auf einem bestimmten Port, in diesem Fall Port 4000. Wenn eine Nachricht ankommt, wird der Inhalt der Nachricht geprüft. Falls die Nachricht leer ist, ignoriert der Server diese. Ansonsten speichert der Server die Nachricht als Textdatei in einem speziellen Ordner, der als Warteschlange für E-Mail-Aufträge dient. Jede Datei bekommt dabei einen eindeutigen Namen, der auf einem Zeitstempel basiert. Dies stellt sicher, dass jede Datei im System eindeutig ist und keine Verwechslungen passieren können. Zusätzlich sendet der TCP-Server eine Bestätigung an den Absender der Nachricht zurück, sobald die Nachricht erfolgreich in der Warteschlange gespeichert wurde.
Nachdem der Benutzer sich erfolgreich angemeldet hat, sendet dieses Skript eine Nachricht an den TCP-Server. Diese Nachricht enthält die E-Mail-Adresse des Benutzers, einen Betreff und einen kurzen Text, der beschreibt, dass sich der Benutzer erfolgreich angemeldet hat.
Das findet ihr in cgi unter login.sh ab zeile 34 Das Login-Skript bekommt dann auf eine Bestätigung vom TCP-Server. Zusammengefasst arbeitet das System in drei Schritten: Erstens, das Login-Skript sendet eine Nachricht an den TCP-Server, wenn sich ein Benutzer anmeldet. Zweitens, der TCP-Server empfängt diese Nachricht und speichert sie als Auftrag in der Warteschlange. Drittens, der Worker verarbeitet diese Aufträge und simuliert das Senden der E-Mail.

9.4 Schnittstelle zum Backend mit curl testen

Die Schnittstelle zum Backend kann als separate Komponente betrachtet werden.
Mit curl können wir die Anwendungsfälle wie Login, mehrmaliges Abfragen der aktuellen Schiffe und Positionen sowie den Logout automatisiert testen.
Dies ermöglicht uns, die Funktionalität der Backend zu prüfen, ohne dass das Frontend beteiligt ist.

9.5 Frontend-Test mit Ajax

Im Frontend wird jede Sekunde ein AJAX-Aufruf abgesendet, um die aktuellen Positionen der Schiffe zu aktualisieren.
Anstatt auf reale Datenänderungen zu warten, können wir definierte Zustandsänderungen simulieren und den erwarteten Ergebniszustand mit dem Istzustand vergleichen.
Dieser Test kann durchgeführt werden, ohne dass das Backend verbunden ist, was besonders hilfreich ist, wenn mehrere Personen im Team arbeiten.
Alternativ könnten wir den Server so einstellen, dass er eine bestimmte Reihenfolge von Ereignissen sendet, um das erwartete Ergebnis zu testen.

9.6 Monitoring und Visualisierung

Wir sollten ein Monitoring in die Anwendung einbauen, damit wir den aktuellen Zustand des Systems und die letzten 24 Stunden im Blick haben.
Mit gnuplot haben wir klare Grafiken erstellt, die zeigen, wie viele Benutzer eingeloggt sind, wie stark das System ausgelastet ist, und wie viele Nachrichten pro Sekunde von Rhodes kommen.
Diese Visualisierungen helfen uns dabei, den Zustand des Systems besser zu verstehen und mögliche Probleme frühzeitig zu erkennen.
So können wir sicherstellen, dass das System gut funktioniert und überwacht wird.

9.7 Lasttest und Systemperformance

Ein Lasttest sollte durchgeführt werden, um herauszufinden, wie viele Benutzer das System gleichzeitig nutzen können, ohne dass es langsamer wird.
Mit gnuplot können wir die Anzahl der Benutzer darstellen, die sich gleichzeitig einloggen und mit dem System arbeiten, ohne dass es zu einer merklichen Verzögerung kommt.

Diese Tests sind wichtig, um die Grenzen des Systems unter realistischen Bedingungen zu verstehen und sicherzustellen, dass es auch bei starker Nutzung stabil bleibt.

Diese detaillierten Tests geben uns ein klares Bild davon, wie belastbar und zuverlässig unser System ist.
So können wir sicherstellen, dass alle Teile – vom Backend bis zur Datenbankschnittstelle – stabil und effizient arbeiten.

9.8 Last-Test-Skript

```
#!/bin/bash

num_requests=100

# URL der Login-Seite
url="http://informatik.hs-bremerhaven.de/docker-infra-2024-1-web/cgi-bin/project_20_24/login.sh"

email="ahmad@gmail.com"
password="1111"

echo "Starting login load test with $num_requests parallel requests to $url"

send_login_request() {
    start_time=$(date +%s%3N)
    response=$(curl -s -L -w "%{http_code}" -o response_body.txt "$url?emailLogin=$email&passwordLogin=$password")
    end_time=$(date +%s%3N)
    response_time=$((end_time - start_time))
    http_code=$(tail -n1 <<< "$response")
    response_body=$(<response_body.txt)
    echo "Response code: ${http_code}, Response time: ${response_time}ms, Response body: ${response_body}"
}

for ((i=1; i<=$num_requests; i++))
do
    send_login_request &
```

```
26 done
27
28 wait
29 echo "Login load test completed."
```

Listing 9.1: Last-Test-Skript

Wir simulieren 100 gleichzeitige Login-Anfragen an unsere Seite und messen die Antwortzeiten des Servers.
Die Ausgabe gibt uns Aufschluss darüber, ob unsere Anfragen erfolgreich waren, wie lange der Server für die Antworten benötigt hat, und wie der Inhalt der Antwort aussieht.
Dies hilft uns zu verstehen, wie gut unser Server mit einer hohen Anzahl gleichzeitiger Anfragen umgehen kann und ob unsere Login-Seite stabil und schnell genug ist.
Unser ausgeführter Test zeigte uns, dass der Server die 100 parallelen Anfragen erfolgreich verarbeitet hat (alle mit dem Statuscode 200).
Die Antwortzeiten variieren, und bei höherer Last werden die Antworten langsamer, was darauf hinweist, dass der Server mit der steigenden Anzahl an gleichzeitigen Anfragen belastet wird.
Einige leere Response bodies könnten darauf hindeuten, dass der Server unter Last nicht alle Antworten vollständig senden konnte.

9.8.1 Last-Test Ergebnisse

Wofür ist der Lasttest? Ein Lasttest wird durchgeführt, um die Leistung einer Website oder Anwendung unter realistischen Bedingungen zu überprüfen, insbesondere wenn viele Benutzer gleichzeitig darauf zugreifen. Hier ist der Vorgang eines Lasttests, den wir für die Website mit Login-Funktion durchgeführt haben. Zunächst zeigen wir unsere Ergebnisse in Form von Diagrammen, die wir mit Gnuplot erstellt haben.

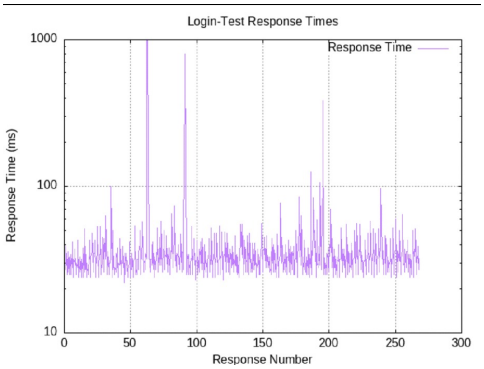


Abbildung 9.1: Last-Test mit 500 Anfragen: Antwortzeiten des Login-Systems

Last-Test mit 500 Anfragen Wir haben mehrere Tests durchgeführt. Der erste Test wurde mit einer großen Zahl von Anfragen durchgeführt, und zwar von 500 Anfragen. Das Lasttest-Diagramm zeigt die Antwortzeiten des Login-Systems unserer Anwendung, die über mehrere Stunden durchgeführt wurde. Auf der Y-Achse sind die Antwortzeiten in Millisekunden dargestellt, während die X-Achse die Anzahl der Anfragen während des Tests repräsentiert.

Die meisten Anfragen haben eine stabile Antwortzeit von unter 100 ms, was bedeutet, dass das Login-System unter normaler Last schnell und zuverlässig reagiert. Allerdings gibt es einige markante Spitzen, bei denen die Antwortzeit kurzfristig auf bis zu 1000 ms ansteigt. Diese Spitzen könnten auf Momente hinweisen, in denen das System stark belastet war durch viele gleichzeitige Login-Versuche. Solche Performance-Einbrüche könnten die Benutzererfahrung beeinträchtigen, da längere Ladezeiten beim Login zu Frustration führen können.

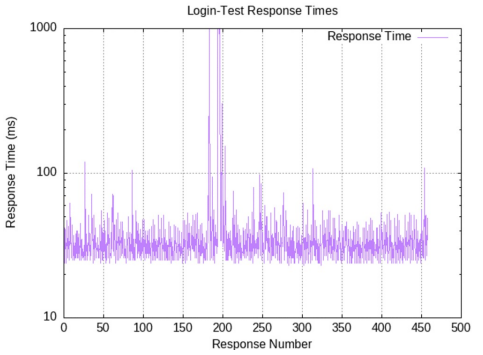


Abbildung 9.2: Last-Test mit 300 Anfragen: Antwortzeiten des Login-Systems

Lasttest für ungefähr 300 Anfragen Das zweite Diagramm zeigt die Antwortzeiten unseres Lasttests für die Login-Funktion unserer Website, dieses Mal mit 300 Anfragen. Die meisten Antwortzeiten liegen unter 100 Millisekunden, was bedeutet, dass das System in den

meisten Fällen sehr schnell reagiert und Anfragen effizient verarbeitet. Allerdings gibt es einige deutliche Ausreißer, bei denen die Antwortzeit auf bis zu 1000 Millisekunden ansteigt. Diese Spitzen treten um die Anfrage-Nummern 50 und 200 auf und könnten auf eine temporäre Überlastung im System hindeuten. Trotz dieser wenigen Spitzen bleibt unser System insgesamt stabil, da die Mehrheit der Anfragen schnell beantwortet wird. Die Verzögerungen sollten jedoch weiter untersucht werden, um potenzielle Probleme zu identifizieren und zu beheben, damit das System auch unter hoher Last zuverlässig bleibt.

10 Herausforderungen & Probleme

Während unseres Projekts sind wir auf verschiedene technische und organisatorische Herausforderungen gestoßen, die uns viel Zeit und Geduld abverlangt haben. Hier sind die Probleme aufgelistet:

10.1 Virtuelle Maschine (VM)

Ein großes Problem war die Arbeit in der virtuellen Maschine, insbesondere bei der Installation der Datenbank. Es gab immer wieder kleine Fehler, die uns daran hinderten, die Datenbank korrekt zu starten und damit zu arbeiten. Deshalb haben wir oft auch mit Hopper gearbeitet, da die Zeit uns davonrannte, während wir versuchten, die Probleme zu lösen, die immer wieder auftauchten, beispielsweise nach Updates. Ein weiteres Problem war das Vergeben von Rechten in Bash. Oft mussten wir manuell Rechte setzen, um die Skripte auszuführen oder den Server korrekt einzurichten. Auch das Speichern der IP-Adresse von Rhodes unter ihrem Domain-Namen war eine Herausforderung, da die Konfiguration häufig verschwand und wir sie neu einstellen mussten.

10.2 GIT

Die Arbeit mit Git war eine weitere Herausforderung. Manchmal wurden Probleme verursacht, insbesondere wenn jemand vergessen hatte, seine Änderungen zu pushen oder falsch gepusht hat. Dies führte häufig zu Konflikten, die mühsam gelöst werden mussten. Rechteprobleme beim Zugriff auf die Repositories haben den Prozess zusätzlich verkompliziert, insbesondere nach Updates von Git.

10.3 Leaflet-Karte

Beim Arbeiten mit Leaflet hatten wir Probleme, Polylines korrekt darzustellen. Oft wurden sie an falschen Standorten gezeichnet oder sie bewegten sich, was sie ungenau machte. Der Startpunkt der Polyline verschob sich manchmal ans Ende oder änderte sogar seine Position, was zu unklaren Visualisierungen führte. Dies machte es schwer, die tatsächliche Position und Route der Schiffe zu verfolgen.

10.4 Schiffstabelle

Ein weiteres Problem war die Aktualisierung der Tabelle. Wir wollten, dass die neuesten Daten immer oben angezeigt werden und die ältesten nach unten rutschen. Doch manchmal wurden die Daten durcheinander angezeigt, was die Übersicht erschwerte. Dies machte es schwieriger, den aktuellsten Stand der Daten im Blick zu behalten, was jedoch im Nachhinein gelöst wurde.

10.5 AJAX

Bei der Arbeit mit AJAX stießen wir auf viele Fehler durch falsche Pfade oder Content-Formate. Dies führte zu häufigen 404-Fehlern oder anderen Fehlermeldungen. Die Debugging-Prozesse nahmen oft viel Zeit in Anspruch, bis die Probleme behoben waren.

10.6 Prozesse und Serverbelastung

Manchmal haben wir zu viele Prozesse gleichzeitig gestartet und vergessen, sie zu beenden, was die Serverlast stark erhöhte. Dies führte zu Verzögerungen und Problemen bei den Skripten. Wir experimentierten mit `while`, `wait` und `sleep` in unseren Automatisierungsskripten, was sich jedoch als nicht effizient herausstellte. Der Ressourcenverbrauch war zu hoch, daher entschieden wir uns, die Skripte zu optimieren und unnötige Prozesse zusammenzuführen.

10.7 Separate Skripte für Schiffstypen

Ein weiteres Problem war, dass wir separate Skripte für die verschiedenen Schiffstypen hatten. Diese Schiffstypen hatten unterschiedliche Datenstrukturen, was die Reihenfolge der Daten komplizierter machte. Das führte zu Fehlern bei der Verarbeitung, da wir oft unterschiedliche Anpassungen vornehmen mussten, um die Skripte korrekt auszuführen. Am Ende haben wir jedoch die Skripte zusammengeführt in einem Skript und mit einfachen `if`-Anweisungen das Problem gelöst.

10.8 Cronjob-Probleme

Wir wollten einen Cronjob, beispielsweise für den Watcher für regelmäßige Aufgaben, einrichten, aber dieser wurde nicht ausgeführt, weil die Pfade nicht optimal waren. Es dauerte, bis wir den Fehler gefunden und den Cronjob korrekt konfiguriert haben.

10.9 LaTeX

Auch bei der Arbeit mit LaTeX hatten wir unerwartete Schwierigkeiten. Oft wurden Bilder nicht korrekt angezeigt oder Texte erschienen anders, als sie geschrieben wurden. Diese Probleme haben viel Zeit gekostet, da wir die Fehler nur durch ständiges Testen und Anpassen beheben konnten. Wegen dieser Verzögerungen gerieten wir manchmal in Zeitnot, da die Arbeit mit LaTeX aufwendiger war als geplant.

11 Reflexion

Unser Projekt hat viel Spaß gemacht und uns geholfen, unser Wissen zu erweitern. Wir haben viel praktisch geübt und dabei gelernt, wie man Probleme löst. Besonders haben wir gelernt, wie wichtig gute Kommunikation in einem Team ist. Durch den ständigen Austausch konnten wir Herausforderungen gemeinsam bewältigen. Manchmal waren die Probleme im Skript klein, aber sie haben uns viel Zeit gekostet. Wir haben festgestellt, dass nicht immer alles reibungslos läuft, und das hat manchmal zu Zeitdruck geführt. Vor allem, weil wir manches anders machen wollten und einige Dinge nicht sofort geklappt haben. Durch die Zusammenarbeit haben wir verstanden, wie wichtig es ist, regelmäßig und klar zu kommunizieren. Ohne gute Kommunikation wäre es schwierig gewesen, unsere Aufgaben zu koordinieren. Wir haben gemerkt, dass es im Team leichter ist, auch schwierige Probleme zu lösen. Während des Projekts mussten wir manchmal schnell handeln und unsere Pläne anpassen. Das hat uns gezeigt, wie wichtig es ist, flexibel zu bleiben und uns auf das Wesentliche zu konzentrieren, wenn die Zeit knapp wird. Durch das Projekt haben wir gelernt, wie mächtig Bash-Skripte sind und wie sie mit der Datenbank, AJAX und JavaScript kombiniert werden können. Besonders schön war es, die Schiffspositionen in Echtzeit auf der Karte zu sehen und zu wissen, dass alles automatisiert funktioniert. Es war ein tolles Gefühl, das Ergebnis zu sehen, besonders nach den vielen Herausforderungen. Insgesamt hat uns das Projekt sehr viel gebracht. Wir haben neue technische Fähigkeiten gelernt und verstanden, wie wichtig es ist, als Team gut zusammenzuarbeiten. Die Herausforderungen haben uns stärker gemacht, und am Ende konnten wir stolz auf das Ergebnis sein.

11.1 Kasem Rashrash

Am Anfang des Semesters war mir die Schnittstelle zwischen AJAX und der Datenbank noch nicht vollständig klar. In den Infrastruktur-Vorlesungen hatte ich zwar das Gefühl, viel Input zu bekommen, aber das praktische Einüben hätte intensiver sein können. Durch andere Fächer wie Rechnerarchitektur und zusätzliche Veranstaltungen war es schwierig, meine Zeit optimal einzuteilen. Trotzdem lernte ich, wie mächtig JavaScript sein kann und welche Vor- und Nachteile AJAX mit sich bringt. Besonders gefallen hat mir die Arbeit in Umgebungen wie virtuellen Maschinen (VMs), die viel Freiheit beim Entwickeln bieten.
Im Laufe des Semesters, nachdem ich mich intensiver mit den verschiedenen Technologien und Tools auseinandergesetzt hatte, bemerkte ich deutliche Fortschritte in meinen Fähigkeiten. Es wurde spürbar, wie schnell sich das Wissen verfestigt, wenn man Technologien aus dem Stegreif beherrscht und sie zur Routine werden. Am Ende erschienen viele Aufgaben, die zu Beginn kompliziert wirkten, nach erfolgreicher Umsetzung viel einfacher – ein klarer Teil des Lernprozesses.
Zu den Technologien, die ich inzwischen sicher anwenden kann, gehören: Git, das Schreiben von Watcher-Skripten, das Verwalten von Linux-Prozessen in der VM, AJAX und CGI. Diese setze ich mittlerweile routiniert ein. Um meine Fähigkeiten weiter zu festigen, habe ich regelmäßig kleine Testprojekte durchgeführt und gezielt geübt, was mir half, das Gelernte zu vertiefen und praktisch umzusetzen.
Besonders prägend für meinen Arbeitsprozess war Git. Durch den regelmäßigen Einsatz von Git habe ich verstanden, wie wichtig es ist, die Arbeit in kleine, nachvollziehbare Einheiten aufzuteilen und diese sauber zu versionieren. Das kontinuierliche Testen und Anpassen von Code hat mir zudem gezeigt, wie essenziell eine gute Kommunikation im Team ist. Absprachen und klare Strukturen sind entscheidend, um effizient und fehlerfrei zu arbeiten.
Die größte Herausforderung für mich und mein Team bestand in der Integration von Datenbanken und der Datenverarbeitung. Es gab eine deutliche Lücke zwischen dem theoretischen Verständnis und der praktischen Umsetzung, insbesondere bei der Lösung komplexer Probleme und der Einführung neuer Technologien. Während des Semesters habe ich viel Zeit investiert, um diese Lücke zu schließen und meine Handlungskompetenz zu verbessern.
Auch meine Teamarbeit hat sich deutlich weiterentwickelt. Ich habe gelernt, Aufgaben besser zu strukturieren, konstruktives Feedback zu geben und zu empfangen sowie klarer und gezielter zu kommunizieren. Insgesamt hat mir das Semester gezeigt, dass man geduldig sein muss und wie man effektiv im Team arbeitet, Technologien schnell erlernt und Probleme löst.

11.2 Ahmad Almohammad

Während des Semesters habe ich ein Bash-Buch gekauft, um mein Wissen aufzufrischen. Da ich im achten Semester bin und lange nicht mit Bash gearbeitet habe, musste ich vieles neu lernen. Wie übe ich und welche Methode funktioniert gut? Um das Gelernte zu üben, setze

ich es in der Praxis um. Dabei probiere ich unterschiedliche Ideen aus. Eine Zeit lang habe ich Kurse auf Plattformen wie Udemy gekauft und mir vorgenommen, sie bis zu einem bestimmten Zeitpunkt zu beenden. Leider hat das oft nicht gut funktioniert. Es gab zwei Probleme: Erstens lernt man zwar viele neue Dinge, aber manchmal fehlt der Überblick, wie alles zusammenpasst. Zweitens verliere ich oft schnell das Interesse, einen Kurs bis zum Ende durchzuarbeiten. Diese Methode habe ich deshalb nicht mehr verwendet. Ich habe gemerkt, dass ich am besten durch praktisches Arbeiten lerne. Ich muss versuchen, das Gelernte in kleine Schritte aufzuteilen und immer wieder Neues hinzuzufügen. Wie viel Zeit habe ich im Semester investiert? Insgesamt habe ich zwischen 120 und 150 Stunden investiert, um das Gelernte zu üben und zu vertiefen.
Wo habe ich Schwierigkeiten?
Manchmal verstehe ich eine neue Sache, aber ich weiß nicht sofort, wie ich sie in der Praxis einsetzen kann. Trotzdem habe ich meistens wenig Probleme, das Gelernte auch umzusetzen. Leider habe ich im Bereich der Teamarbeit nicht viel Neues gelernt. Wir haben zwar oft zusammengearbeitet und dabei immer gute Ergebnisse erzielt, aber ich habe oft das Gefühl, dass ich schneller vorankomme, wenn ich allein an einer Aufgabe arbeite. Hoffentlich ändert sich das im Berufsleben.

11.3 Shiwan

Im Laufe des Semesters habe ich viele neue Technologien und Methoden kennengelernt. Besonders wichtig war für mich das Arbeiten mit JavaScript und der Leaflet-Bibliothek, um Karten auf unserer Website darzustellen. Mit AJAX habe ich es geschafft, die Inhalte zu aktualisieren, ohne die Seite neu zu laden. Eine große Herausforderung war der Umgang mit MySQL-Datenbanken. Ich musste lernen, wie man die Positionen der Schiffe speichert und später wieder abruft. Ein weiteres wichtiges Werkzeug war Git. Mit Git war die Teamarbeit leichter und jede Änderung war nachzuverfolgen. Die größte Schwierigkeit für mich war immer, den Code auswendig zu lernen, besonders bei komplexen Teilen wie dem Accordion. Auch bei CGI-Skripten hatte ich am Anfang große Probleme. Viele Konzepte waren mir nicht klar, und ich musste viel Zeit investieren, um sie wirklich zu verstehen.ich musste oft auf Beispiele und Dokumentationen die auf Element oder auf der Tutorial Seite mir geben. Erst nach mehreren Wiederholungen und Übungen wurde es für mich einfacher. Diese Zusammenarbeit war sehr hilfreich, um den Überblick zu behalten und sicherzustellen, dass wir auf dem richtigen Weg sind. Besonders wichtig war es, dass wir uns gegenseitig bei Problemen geholfen haben. So konnte ich viele Schwierigkeiten schneller bewältigen und weiter im Projekt arbeiten.
Ich habe auch gelernt, wie wichtig es ist, geduldig zu bleiben, wenn es mal nicht so gut läuft, und auf die Unterstützung des Teams zu zählen. Insgesamt hat mir die Teamarbeit gezeigt, wie man gemeinsam Herausforderungen meistert. Diese Erfahrung hat mir geholfen, besser im Team zu arbeiten und schneller Lösungen für Probleme zu finden.

11.4 Mustafa Thamer

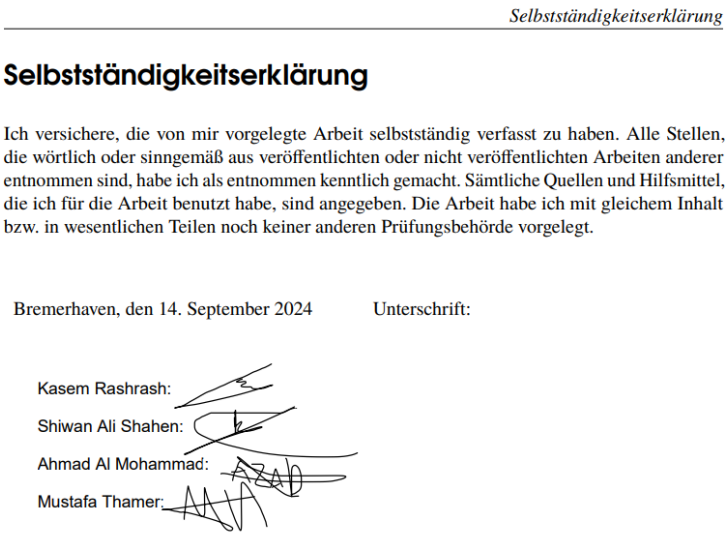
In diesem Semester habe ich viele neue Dinge gelernt. Ich habe zum ersten Mal AJAX, JavaScript, Bash und virtuelle Maschinen miteinander verbunden. Diese Schnittstellen waren mir bisher nicht so bekannt. Diese neuen Inhalte waren spannend und haben mir Spaß gemacht. Allerdings gab es auch viele kleine Probleme, die mehr Zeit gekostet haben als erwartet. Manchmal hatte ich das Gefühl, dass ich viel Zeit für einfache Aufgaben brauche, weil vieles neu für mich war. Trotzdem habe ich dabei viel gelernt. Die Teamarbeit war sehr wichtig für mich. Durch die vielen Meetings mit meinem Team konnte ich meine Kommunikationsfähigkeiten verbessern. Es war hilfreich, regelmäßig Rückmeldungen zu geben und zu bekommen. Die Arbeit im Team hat uns geholfen, viele Probleme schneller zu lösen. Besonders das Arbeiten mit Git hat mir gezeigt, wie wichtig es ist, kleine Arbeitsschritte zu dokumentieren und zu teilen.
Leider hatte ich während des Semesters auch andere Prüfungen, die mich unter Druck gesetzt haben. Dadurch geriet ich oft in Zeitnot, und die Zeitplanung war schwierig für mich. Ich konnte nicht genug Zeit für andere Fächer wie C++ finden, weil ich mich auf unser Projekt konzentrieren musste. Das hat meine Psyche manchmal belastet, weil ich nicht alles geschafft habe, was ich wollte.
Die Arbeit an der Infrastruktur und das Lernen der neuen Technologien war für mich sehr zeitintensiv. Es hat eine Weile gedauert, bis ich die neuen Inhalte verstanden und umsetzen konnte. Trotz der Schwierigkeiten bin ich froh, dass ich am Ende so viel gelernt habe. Insgesamt war das Projekt eine gute Erfahrung, auch wenn es herausfordernd war.

12 Fazit

In unserem Projekt haben wir mehrere wichtige Ziele erreicht.
Wir haben es geschafft, die Schiffspositionen automatisch zu erfassen, in der Datenbank zu speichern und im Frontend in Echtzeit über AJAX anzuzeigen.
Der Watcher, der die Verbindung zu Rhodes überwacht, funktioniert auch gut und wird dank Crontab nach einem Neustart automatisch wieder aktiviert.
Außerdem haben wir Lasttests gemacht, um zu sehen, wie viele Benutzer sich gleichzeitig einloggen können, ohne dass das System langsamer wird.
Diese Tests haben uns gezeigt, wie gut unser System unter Belastung funktioniert.
Ein weiteres Highlight war die Darstellung der Schiffspositionen auf der Karte in Echtzeit, was besonders schön war, nachdem wir so viel Arbeit investiert hatten.
Insgesamt haben wir eine stabile Verbindung zwischen Backend, Datenbank und Frontend geschaffen.
Die Daten werden richtig erfasst und live im Frontend angezeigt.
Mit den Bash-Skripten und Crontab haben wir eine Automatisierung umgesetzt, die ohne unser Eingreifen zuverlässig läuft.
Die Lasttests haben gezeigt, dass das System auch viele Benutzer gleichzeitig gut verarbeiten

kann, obwohl wir auch die Grenzen der Belastbarkeit erkannt haben.

Zustandselbsständigkeiterklärung



Selbstständigkeitserklärung

Selbstständigkeitserklärung

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

Bremerhaven, den 14. September 2024 Unterschrift:

Kasem Rashrash:
Shiwan Ali Shahen:
Ahmad Al Mohammad:
Mustafa Thamer:

Abbildung 12.1: Dies ist eine Bildunterschrift

Quellenverzeichnis

1. https://s3-eu-west-1.amazonaws.com/gxmedia.galileo-press.de/leseproben/5507/leseprobe_rheinwerk_shell-programmierung.pdf
2. https://www.w3schools.com/sql/func_mysql_concat.asp
3. https://stackoverflow.com/questions/16101495/how-can-i-suppress-column-header-output-for-a-single-sql-statement
4. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
5. https://api.vesselfinder.com/docs/response-ais.html
6. https://wiki.openstreetmap.org/wiki/OpenSeaMap

7. https://leafletjs.com/examples.html
8. https://www.igismap.com/leafletjs-point-polyline-polygon-rectangle-circle/
9. https://www.w3schools.com/js/js_ajax_http.asp
10. https://attacomsian.com/blog/xhr-post-request
11. https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest_API/Using_XMLHttpRequest
12. https://www.codeproject.com/articles/26335/append-rows-and-controls-into-a-table-element-with
13. https://www.slingacademy.com/article/javascript-displaying-json-data-as-a-table-in-html/
14. https://earthly.dev/blog/automate-tasks-shell-scripts/
15. https://thelinuxcode.com/add-row-to-html-table-using-javascript/
16. https://github.com/PiresTiago/MySQL-To-JSON
17. https://www.baeldung.com/linux/jq-command-json

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis