Hochschule Bremerhaven

Fachbereich II
Management und Informationssysteme
Informatik B.Sc.

Modul
Infrastruktur

Ausarbeitung Infrastruktur 2024

infra-2024-e

Vorgelegt von: Laura Halbeck MatNr. 37578
Fabian Tober MatNr. 40607

Vorgelegt am: 13. September 2024

Dozent:in: Prof. Dr. Oliver Radfelder

Inhaltsverzeichnis

1 Aufgabenstellung

Unsere Aufgabenstellung umfasst zwei Hauptteile: die Programmierung einer Webanwendung und die Erstellung eines Berichts.

Programmierteil:

Es soll eine Webanwendung entwickelt werden, bei der sich Nutzer*innen mit E-Mail und Passwort anmelden können. Die Anwendung hat drei Hauptbestandteile:

1. **Login**: Nutzer*innen melden sich mit E-Mail und Passwort an.
2. **Karte**: Eine Leaflet-Karte zeigt Schiffe an, deren Daten in den letzten 5 Minuten empfangen wurden.
3. **Schiffe**: Eine Tabelle listet alle auf der Karte sichtbaren Schiffe auf.

- E-Mail-Versand wird simuliert, nicht real durchgeführt.
- Eine Warteschlange verwaltet Aufträge, z.B. das Versenden von E-Mails.
- Die Anwendung läuft unter Linux, verwendet Mariadb für die Datenbank und Apache2 als Webserver, wobei für Backend-Skripte die Bash und für das Frontend „Vanilla-JavaScript“ genutzt werden.

Bericht:

- Der Bericht wird in LaTeX verfasst und umfasst etwa zehn Seiten pro Person.
- Es sollen alle wesentlichen Technologien und der Entwicklungsprozess dokumentiert werden, inklusive der Nutzung von Git, Tests, Monitoring und Lasttests.
- Der Bericht soll keine Arbeitsteilung widerspiegeln, sondern sicherstellen, dass alle Teammitglieder die gesamte Anwendung und den Code verstehen.
- Tests, Ressourcenverbrauch und Performance der Anwendung müssen ebenfalls analysiert und dokumentiert werden.

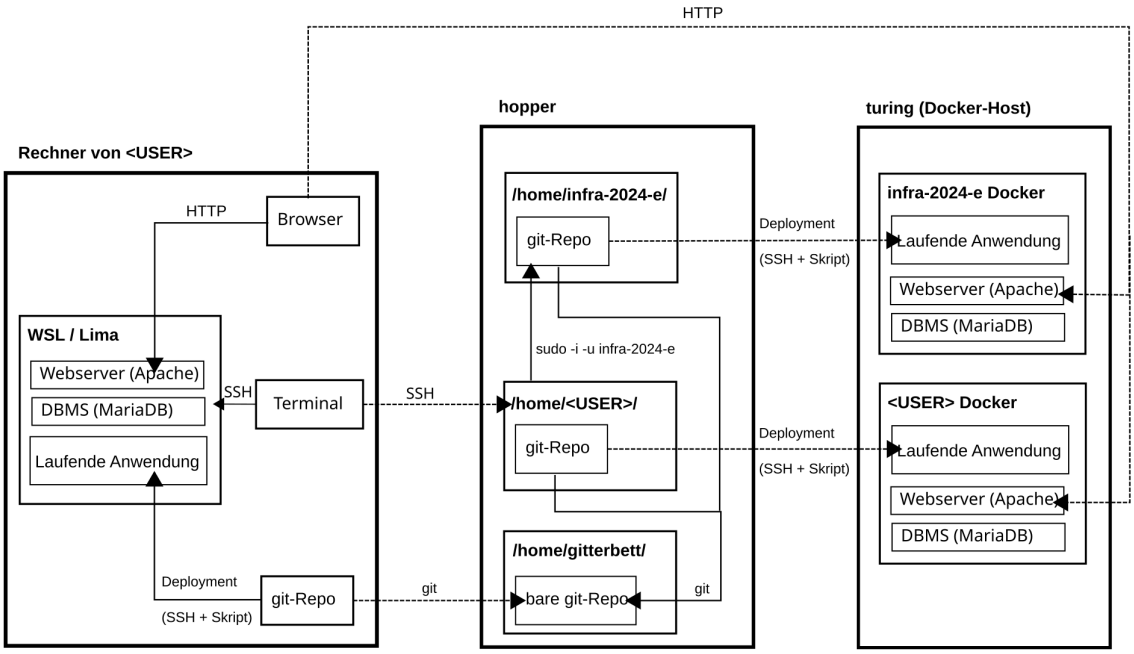

Abbildung 1.1: Ablaufumgebung | infra-2024-e | Schiffdaten HS Bremerahven

Ablaufumgebung

Die Ablaufumgebung in der Grafik zeigt, wie eine Webanwendung auf verschiedenen Ebenen entwickelt, getestet und bereitgestellt wird. Diese Umgebung besteht aus mehreren miteinander verbundenen Systemen, die alle unterschiedliche Aufgaben erfüllen. Dabei spielen lokale Entwicklung, Versionskontrolle und die Bereitstellung auf entfernten Servern eine entscheidende Rolle. Im Folgenden wird jede Komponente und ihre Funktion detailliert beschrieben.

1. Rechner von <USER>

Der Rechner des Nutzers stellt die lokale Entwicklungsumgebung dar. Hier arbeitet der Nutzer direkt an der Webanwendung in einer Umgebung, die durch WSL (Windows Subsystem for Linux) oder Lima (virtuelle Maschine für Linux-Distributionen) simuliert wird. In dieser Umgebung laufen alle wesentlichen Dienste, um die Webanwendung lokal zu entwickeln, zu testen und zu betreiben. Diese Dienste umfassen:

- **Webserver (Apache):** Der Apache-Webserver wird unter WSL/Lima lokal auf dem Rechner des Nutzers betrieben. Er ist dafür zuständig, die Webanwendung lokal zu hosten und über den Browser zugänglich zu machen. Dies ermöglicht es dem Nutzer, die Anwendung in einer Testumgebung auszuführen, bevor sie auf entfernte Server übertragen wird.
- **Datenbank (MariaDB):** Die MariaDB-Datenbank, die ebenfalls lokal unter WSL/Lima läuft, speichert die für die Webanwendung erforderlichen Daten. In dieser Entwicklungsphase greift die Anwendung auf die lokale Datenbank zu, um alle Funktionen vollständig zu testen.
- **Laufende Anwendung:** Die Webanwendung selbst wird auf dem lokalen Rechner unter WSL/Lima ausgeführt. Hier kann der Nutzer sämtliche Änderungen am Quellcode direkt in einer realen Laufzeitumgebung testen und sicherstellen, dass die Anwendung korrekt funktioniert, bevor sie auf entfernte Server bereitgestellt wird.
- **Terminal:** Das Terminal auf dem lokalen Rechner ist das Hauptwerkzeug, um mit entfernten Servern wie „hopper“ und „turing“ zu kommunizieren. Über **SSH** (Secure Shell) kann der Nutzer Dateien hochladen, Befehle auf den Servern ausführen und die Anwendung remote deployen.
- **Git-Repository:** Das lokale Git-Repository verwaltet alle Änderungen am Quellcode der Anwendung. Git ist ein Versionskontrollsystem, das es dem Nutzer ermöglicht, Änderungen nachzuverfolgen, zu speichern und mit anderen Entwicklern zu teilen. Sobald Änderungen lokal abgeschlossen sind, werden sie per **Git** auf den Server „hopper“ übertragen.

Die lokale Umgebung, unterstützt durch WSL/Lima, bietet somit eine vollständige Testumgebung, in der die Webanwendung entwickelt und vorab getestet werden kann.

2. Server „hopper“

„Hopper“ ist der zentrale Server, der als Schnittstelle zwischen der lokalen Entwicklungsumgebung auf dem Rechner des Nutzers und dem endgültigen Produktionsserver „turing“ fungiert. Hier wird die Anwendung weiterentwickelt, der Quellcode verwaltet und der Bereitstellungsprozess koordiniert. Es gibt mehrere wichtige Verzeichnisse und Repositories auf „hopper“, die verschiedene Funktionen erfüllen:

- **/home/<USER>/:** In diesem Verzeichnis befindet sich das persönliche Git-Repository des Nutzers, in dem der Quellcode abgelegt wird. Über SSH werden Änderungen vom lokalen Rechner auf dieses Verzeichnis übertragen. Hier können die Änderungen vorübergehend gespeichert und getestet werden.

- **/home/infra-2024-e/:** Dieses Verzeichnis ist für die spezielle Version der Anwendung reserviert, die unter dem Nutzer „infra-2024-e“ läuft. Über Skripte wird der Quellcode aus dem Nutzerverzeichnis in dieses Verzeichnis übertragen, um die Anwendung dort zu testen und für die Bereitstellung auf „turing“ vorzubereiten.
- **Bare Git-Repository:** Im Verzeichnis **/home/gitterbett/** befindet sich ein sogenanntes „bare“ Git-Repository, das als zentraler Ort zur Verwaltung von Quellcode-Versionen dient. Ein „bare“ Repository enthält keinen Arbeitsbaum und speichert nur die Versionshistorie des Codes. Es ist der zentrale Punkt, an dem alle Änderungen gesammelt werden, bevor sie zur Bereitstellung auf den Server „turing“ weitergeleitet werden.

„Hopper“ spielt eine entscheidende Rolle im Bereitstellungsprozess, da der Quellcode von den Entwicklern auf diesem Server gesammelt, getestet und dann auf „turing“ übertragen wird.

3. Server „turing“ (Docker-Host)

„Turing“ ist der Server, auf dem die Webanwendung letztendlich bereitgestellt wird. Dieser Server verwendet **Docker**, um die Anwendung in einer stabilen, isolierten Umgebung auszuführen. Docker-Container ermöglichen es, Anwendungen samt ihrer Abhängigkeiten in einer sicheren Umgebung zu betreiben, die unabhängig vom Host-System funktioniert. Auf „turing“ gibt es zwei wichtige Docker-Container:

- **infra-2024-e Docker:** Dieser Docker-Container hostet die Version der Anwendung, die unter dem Nutzer „infra-2024-e“ läuft. Er enthält sowohl den Apache-Webserver als auch die MariaDB-Datenbank. Alle HTTP-Anfragen an diesen Container werden vom Webserver bearbeitet, während die Datenbank die notwendigen Anwendungsdaten verwaltet.
- **<USER> Docker:** Jeder Nutzer hat seinen eigenen Docker-Container, der eine isolierte Instanz der Webanwendung bereitstellt. Dieser Container läuft unabhängig vom „infra-2024-e“ Container, sodass der Nutzer seine eigene Version der Anwendung testen kann, ohne die Hauptinstanz zu beeinflussen. Auch hier gibt es einen Apache-Webserver und eine MariaDB-Datenbank.

Der Einsatz von Docker auf „turing“ ermöglicht eine klare Trennung zwischen den verschiedenen Instanzen der Anwendung und gewährleistet, dass verschiedene Nutzer unabhängig voneinander arbeiten können.

4. Verbindungen zwischen den Komponenten

Die verschiedenen Komponenten der Umgebung sind durch mehrere Protokolle und Technologien miteinander verbunden:

- **SSH-Verbindungen:** SSH wird verwendet, um sicher von einem System auf ein anderes zuzugreifen. Der Nutzer kann sich über SSH vom lokalen Rechner zu „hopper“ und „turing“ verbinden, um Dateien hochzuladen, Befehle auszuführen und die Anwendung remote zu verwalten.
- **Git:** Git ist das Versionskontrollsystem, das dafür sorgt, dass alle Änderungen am Quellcode verfolgt und synchronisiert werden. Der Nutzer überträgt seine Änderungen vom lokalen Git-Repository auf dem Rechner zu den Repositories auf „hopper“, von wo aus sie auf „turing“ weitergeleitet werden.
- **HTTP:** HTTP wird verwendet, um die Webanwendung im Browser anzuzeigen. Sowohl lokal auf dem Rechner des Nutzers als auch auf „turing“ kann die Anwendung über HTTP im Browser getestet und betrachtet werden.

Diese Verbindungen schaffen eine effiziente und flexible Umgebung, in der die Entwicklung lokal stattfinden kann, während die Anwendung remote auf Servern bereitgestellt und getestet wird.

2 Technologien

JavaScript mit DOM im Browser

JavaScript ermöglicht die dynamische Manipulation von Webseiten im Browser, insbesondere durch die Interaktion mit dem **Document Object Model (DOM)**. Das DOM repräsentiert die HTML-Struktur einer Webseite als eine Baumhierarchie, wobei jedes Element ein Knoten ist.

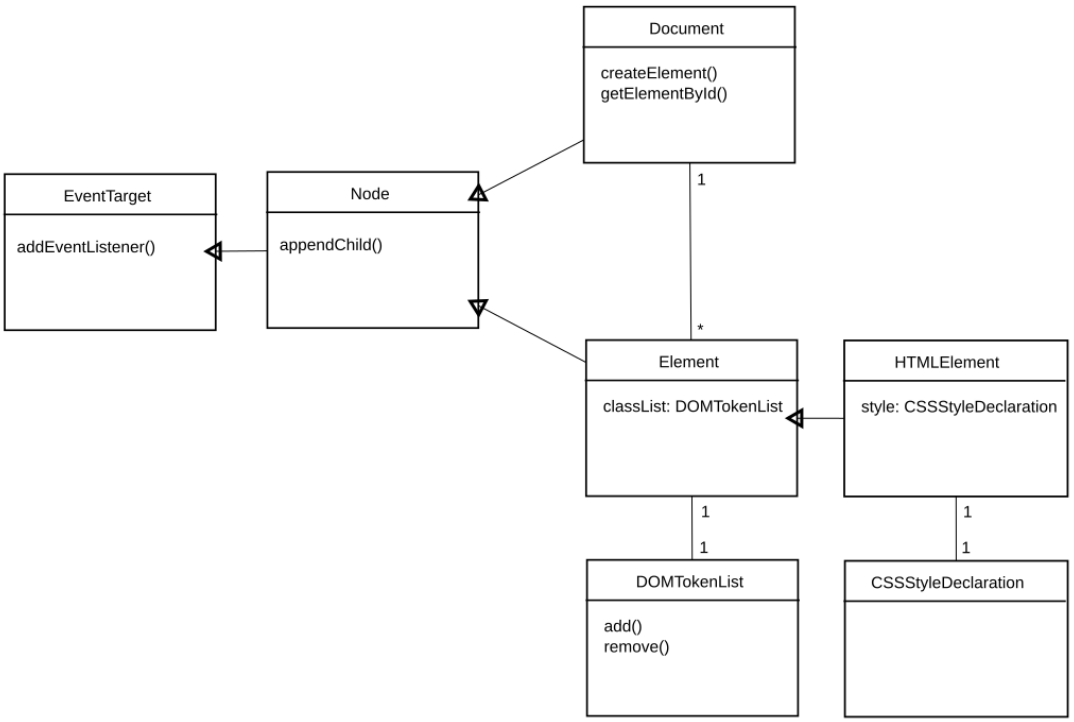

Abbildung 2.1: Klassendiagramm JavaScript

Klassendiagramm Beschreibung:

- **EventTarget**: Basisklasse, die es erlaubt, Ereignisse an DOM-Elemente zu binden, z. B. mit der Methode `addEventListener()`.
- **Node**: Repräsentiert ein einzelnes DOM-Element, das mit der Methode `appendChild()` andere Knoten hinzufügen kann.
- **Document**: Der Ausgangspunkt für die Manipulation des DOMs. Mit Methoden wie `createElement()` oder `getElementById()` können neue Elemente erzeugt und existierende abgerufen werden.
- **Element**: Eine spezielle Art von `Node`, die HTML-Elemente darstellt. Enthält Attribute wie `classList`, die eine Instanz von `DOMTokenList` sind und Methoden wie `add()` und `remove()` besitzen.
- **HTMLElement**: Eine spezifische Unterklasse von `Element`, die zusätzliche Attribute wie `style` (vom Typ `CSSStyleDeclaration`) für die direkte Manipulation von CSS-Eigenschaften bietet.

Zusammengefasst bietet dieses Klassendiagramm eine Abbildung der zentralen Schnittstelle von JavaScript zum DOM, die durch hierarchische Beziehungen und Methoden klar wird. Dies ist die Grundlage für die dynamische Manipulation von Webseiteninhalten durch JavaScript.

Leaflet

Leaflet ist eine leichtgewichtige Open-Source-JavaScript-Bibliothek zur Darstellung interaktiver Karten. Sie bietet die Möglichkeit, verschiedene Layer, Marker und geometrische Formen auf Karten zu rendern und diese dynamisch zu steuern.

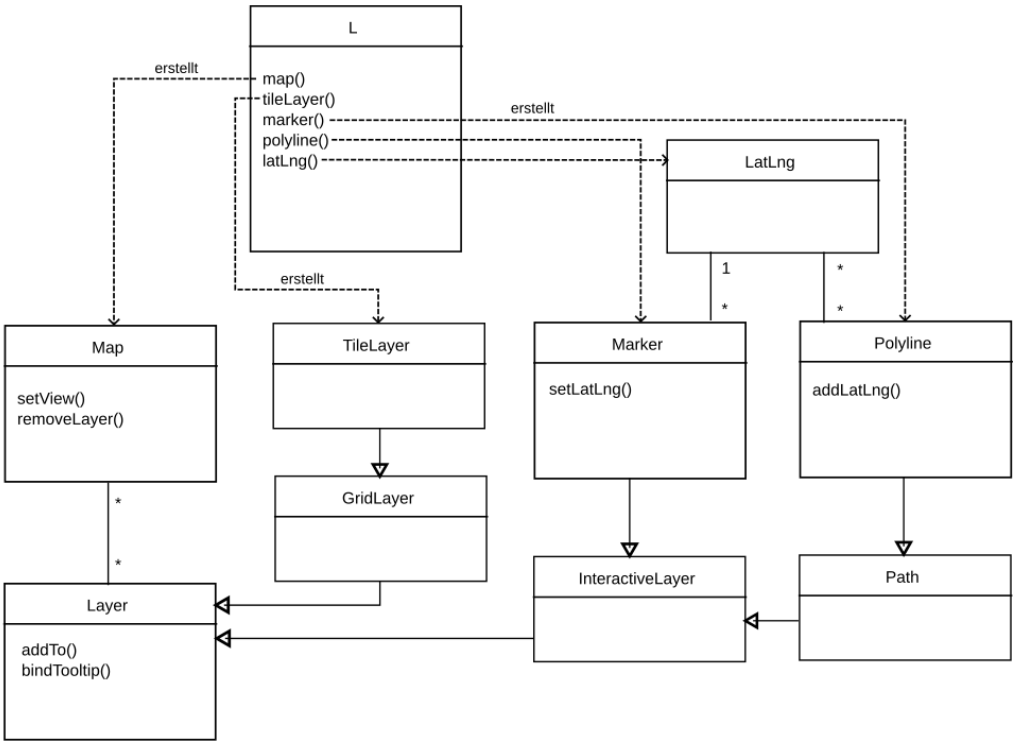

Abbildung 2.2: Klassendiagramm Leaflet

Klassendiagramm Beschreibung:

- **L**: Die Hauptschnittstelle der Leaflet-Bibliothek, welche Methoden wie `map()`, `tileLayer()`, `marker()`, `polyline()` und `latLng()` bietet. Diese Methoden ermöglichen die Erstellung von Karten, Markern und Linien.

- **Map**: Stellt eine Leaflet-Karte dar. Methoden wie `setView()` oder `removeLayer()` ermöglichen die Navigation und das dynamische Hinzufügen/Entfernen von Layern.
- **TileLayer**: Dient dazu, Kacheln (Tiles) aus verschiedenen Quellen (z. B. OpenStreetMap) als Layer auf einer Karte zu laden. Wird in `GridLayer` organisiert.
- **Marker**: Repräsentiert Markierungen auf der Karte. Die Methode `setLatLng()` erlaubt es, die Position eines Markers dynamisch zu ändern.
- **Polyline**: Zeichnet Linien auf der Karte, wobei mit der Methode `addLatLng()` zusätzliche Punkte zu einer Linie hinzugefügt werden können.
- **LatLng**: Einfache Klasse, die geografische Koordinaten (Breiten- und Längengrad) darstellt.

Das Leaflet-Klassendiagramm zeigt die Hierarchie und den Zusammenhang der verwendeten Klassen und Methoden, die für die Darstellung und Interaktion mit Karten verwendet werden.

Datenbankmanagementsystem

MariaDB ist ein relationales Datenbankmanagementsystem (DBMS), das auf SQL (Structured Query Language) basiert. Es wird verwendet, um Daten strukturiert zu speichern und abzurufen. Typische Anfragen wie `SELECT`, `INSERT`, `UPDATE` und `DELETE` ermöglichen den Zugriff auf und die Verwaltung von Daten.

Rolle in der Anwendung: In einer Webanwendung, die Leaflet nutzt, könnten geografische Daten wie Koordinaten von Markern oder Polylinien in einer MariaDB-Datenbank gespeichert und dynamisch abgerufen werden. Diese Daten werden dann mithilfe von JavaScript und Leaflet in der Benutzeroberfläche visualisiert.

Apache mit CGI als Webserver

Apache ist ein weit verbreiteter Webserver, der HTTP-Anfragen entgegennimmt und Webseiten sowie APIs bereitstellt. **CGI (Common Gateway Interface)** ermöglicht die Interaktion von Apache mit serverseitigen Programmen, um dynamische Inhalte wie Daten aus einer Datenbank bereitzustellen.

Rolle in der Anwendung: Apache könnte für die Bereitstellung einer Leaflet-Karte verwendet werden, wobei CGI-Skripte auf die Datenbank zugreifen und Daten in Echtzeit an den Client senden. So könnten Nutzer beispielsweise Marker auf einer Karte hinzufügen, deren Daten über Apache und CGI an die Datenbank weitergeleitet werden.

Linux als Betriebssystem

Linux ist ein Open-Source-Betriebssystem, das in der Regel als Serverumgebung genutzt wird. Es bildet die Grundlage für den Betrieb des Apache-Webservers, die Ausführung von CGI-Skripten und die Verwaltung der MariaDB-Datenbank.

HTTP-Protokoll

HTTP (Hypertext Transfer Protocol) ist das grundlegende Protokoll für die Kommunikation im Web. Es regelt den Austausch von Daten zwischen Client (z. B. Browser) und Server (z. B. Apache). In dieser Anwendung würde HTTP dazu verwendet, Anfragen an den Webserver zu senden (z. B. das Laden der Leaflet-Karte) und Antworten wie HTML, CSS oder JavaScript zu empfangen.

Zusammengefasst wird in dieser Architektur JavaScript verwendet, um über das DOM im Browser dynamische Inhalte zu manipulieren, Leaflet für die Anzeige und Interaktion mit Karten, und Apache, CGI, MariaDB und Linux, um Daten zu speichern und serverseitige Funktionen bereitzustellen.

3 Architektur

3.1 Beschreibung

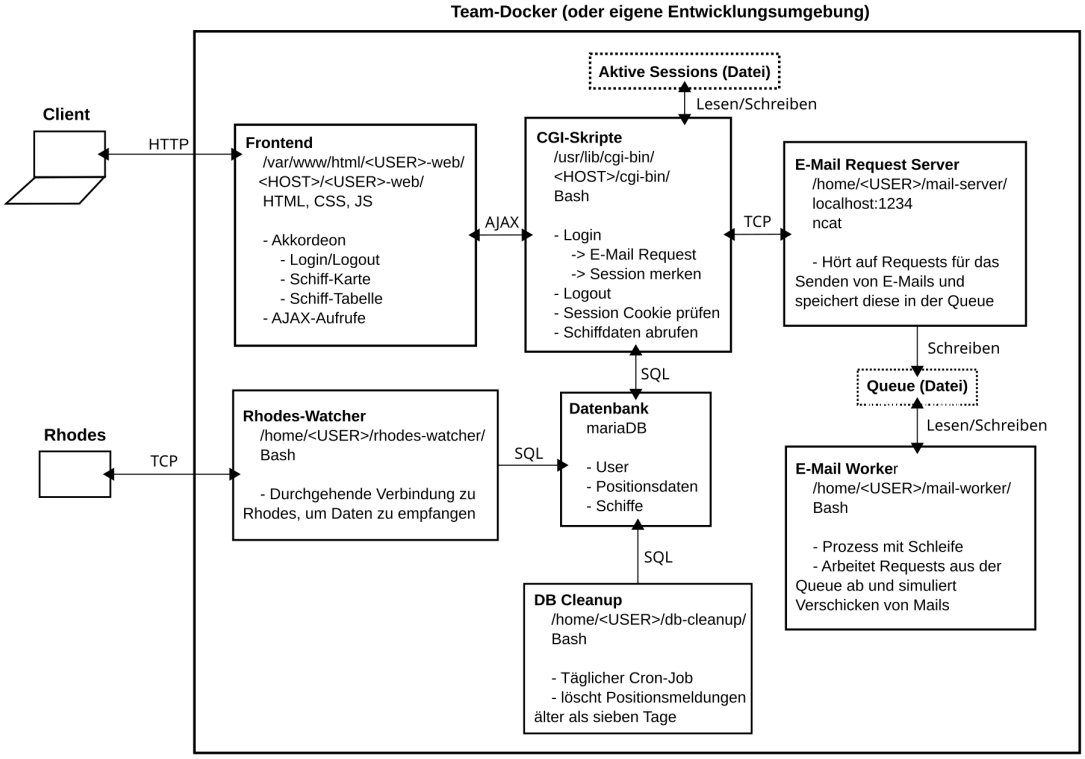

Abbildung 3.1: Darstellung unserer Architektur

Die Abbildung 3.1 stellt unsere Architektur grafisch dar. Unsere Anwendung besteht aus den einzelnen Komponenten Frontend, CGI-Skripte, Datenbank, DB-Cleanup, Rhodes-Watcher, Mail-Server und Mail-Worker. Diese kommunizieren auf verschiedene Weisen miteinander, zum Beispiel über TCP, SQL-Queries oder das Schreiben und Lesen aus derselben Datei. Das Frontend sowie die CGI-Skripte sind von außen über das Internet erreichbar, wobei vom Nutzer nur ein direkter Zugriff auf das Frontend angedacht ist. Der Rhodes-Watcher kommuniziert über das Internet ausschließlich mit Rhodes.

Ruft ein Nutzer die URL des Frontends auf, sendet er damit einen HTTP-Request an unseren Apache-Server. Als Antwort erhält er die Datei index.html. Diese beinhaltet Referenzen auf das CSS und JavaScript, welches der Browser des Nutzers über weitere HTTP-Requests an unseren Server nachlädt. Nun kann der Nutzer mit dem Frontend interagieren. Das JavaScript wird

clientseitig ausgeführt. Soll sich dieses zum Beispiel mit dem Login-CGI-Skript in Verbindung setzen, geschieht dies somit über einen weiteren HTTP-Request von der Maschine des Nutzers an unseren Server. Bevor unser CGI-Skript antworten kann, muss es zunächst einen SQL-Query an unsere Datenbank schicken. Die Datenbank antwortet und das CGI-Skript weiß nun, ob die Login-Daten korrekt sind oder nicht. Sind sie korrekt, arbeitet es einige Aufgaben ab: Zunächst schreibt es eine generierte Session-ID, welche später auch mittels Set-Cookie Header an den Nutzer gesendet wird, in eine Datei auf dem Docker. Dann öffnet es mittels ncat eine Verbindung zu unserem Mail-Request-Server. Dieser hört nur auf Verbindungen von derselben Maschine. Das Skript übermittelt die Daten, die als Mail geschickt werden sollen, und erhält eine Bestätigung. Dann kann die vollständige Antwort des CGI-Skripts an den Nutzer geschickt werden. Währenddessen schreibt der Mail-Request-Server die erhaltenen Daten in die Queue, aus der sie der Mail-Worker wieder herausholen kann, sobald er dafür verfügbar ist. Der Nutzer würde dann, wenn wir tatsächlich E-Mails verschicken würden, eine Mail erhalten. Die Antwort des CGI-Skripts wird wieder clientseitig im Browser verarbeitet und die HTML-Seite entsprechend angepasst.

Auch ohne Nutzer-Interaktion läuft durchgängig der Rhodes-Watcher, welcher eine durchgängige Verbindung zu Rhodes hält und jedes Mal, wenn er Daten von Rhodes empfängt, per SQL-Query die Datenbank aktualisiert. Außerdem wird per Cron-Job alle sieben Tage ein Skript zum Aufräumen der Datenbank gestartet.

3.2 Beispielcode

```
# Prüfen, ob Nutzer mit Email und Password existiert
email=$(echo $QUERY_STRING | grep -Eo 'email=[^&]+' | cut -d '=' -f 2)
password=$(echo $QUERY_STRING | grep -Eo 'password=[^&]+' | cut -d '=' -f 2)
query="SELECT id FROM User WHERE email = '$email' AND password = '$password';"
result=$(mariadb --defaults-file="private/.my.cnf" -e "$query")

if test "$result"; then
  # Cookie setzen in Browser und Datei
  session="$(pwgen 40 1)"
  echo "Set-Cookie:infra2024-e-session=$session; Secure; HttpOnly; SameSite=Strict;"
  echo "$email" > cookies/"$session"

  # Mail-Anfrage schicken
  ncat -e "send-mail-request.sh $email" 127.0.0.1 1234

  echo "Content-Type: text/plain"
  echo
```

```
18    echo "$email"
19  else
20    echo "Content-Type: text/plain"
21    echo
22  fi
```

Listing 3.1: Login-CGI-Skript

In Listing 3.1 ist das Login-CGI-Skript dargestellt. Es nimmt wie bereits beschrieben HTTP-Anfragen entgegen, die im Query-String die E-Mail und das Passwort enthalten sollen. Per grep und cut werden die Daten aus dem Query-String geholt (Zeile 6-7). grep sucht dabei nach dem Key gefolgt von einem Gleichheitszeichen gefolgt von beliebig vielen Zeichen, die kein Ampersand sind. Der Schalter *-E* schaltet die erweiterte Regex ein, die wir für die Negation benötigen. Durch den Schalter *-o* wird nicht die ganze Zeile, sondern nur der gefundene Text ausgegeben. Dabei wird der Key allerdings auch mit ausgegeben, daher pipen wir die Ausgabe weiter in cut und reduzieren ihn auf den zweiten Teil (*-f 2*) nach dem Gleichheitszeichen (*-d '='*).

Die Daten werden in ein SQL-Query eingesetzt und per *mariadb -e* an die Datenbank gesendet (Zeile 4-5). Dabei geben wir zur Authentifizierung die *.my.cnf*, die wir beim Deployment in */usr/lib/cgi-bin/private/* kopieren, als *defaults-file* mit. Die Antwort von MariaDB speichern wir in einer Variable. Ist sie leer, dann wurde kein Nutzer mit dieser E-Mail und Passwort-Kombination gefunden. Das prüfen wir mit *test*.

In jedem Fall wird im Header *Content-Type: text/plain* zurückgegeben (Zeile 20 bzw. 24). Im Fall, dass der Login fehlschlägt, ist die Antwort sonst leer. Bei erfolgreichen Login wird im Header noch per *Set-Cookie* der Session-Cookie gesetzt, welcher vorher mit pwgen generiert wurde (Zeile 13-14). Im eigentlichen Inhalt der Antwort wird dann als Bestätigung die E-Mail des Nutzers ausgegeben (Zeile 23).

Bei erfolgreichem Login wird zusätzlich noch via ncat die Verbindung zum Mail-Request-Server unter 127.0.0.1 auf Port 1234 aufgebaut (Zeile 18). Ein separates Skript *send-mail-request.sh* nimmt die E-Mail als Argument entgegen und sendet die Daten per echo an den Server.

3.3 Tests

```
1  response=$(curl -s
   ↪  "localhost/$USER-web/cgi-bin/login.sh?email=demo@user.de&password=demo")
2
3  if test "$response" = "demo@user.de"; then
```

```
4       exit 0
5   fi
6
7   echo "\"demo@user.de\" erwartet, aber \"$response\" erhalten"
8   exit 1
```

Listing 3.2: Test für Login

Listing 3.2 zeigt ein Beispiel eines Test-Skripts für den Login. In diesem Testfall prüfen wir, ob bei validen Login-Daten auch die E-Mail des Nutzers ausgegeben wird. Tests für den Negativ-Fall (invalide Daten), das Versenden der Mail und das Setzen des Cookies haben wir ebenfalls geschrieben. Die Tests terminieren bei Erfolg mit Exit Code 0. Bei Misserfolg geben sie die erwarteten und erhaltenen Daten aus und terminieren mit einem Fehler-Code. Wir ließen diese Tests nach jeder Änderung am Code noch einmal laufen, um sicherzustellen, das alles noch funktioniert.

4 Versionsgeschichte

Für unsere gemeinsame Entwicklung verwendeten wir Git. Dies erlaubte uns, eine gemeinsame Code-Basis zu haben, an der wir von unseren Rechnern oder von Hopper aus Änderungen machen konnten. Mithilfe der Historie konnten wir nachverfolgen, wann von wem welche Änderungen gemacht wurden und wenn nötig auch alte Stände wiederherstellen.

Wenn jemand eine Änderung am Code vornehmen möchte, holt er sich zunächst mit *git pull* den aktuellen Stand des Repos. Dann führt er die Änderungen durch und merkt die modifizierten Dateien mit *git add <Dateien>* zum Commit vor. Committet wird mit *git commit -m '<Nachricht>'* zunächst als lokaler Commit, der daraufhin mittels *git push* auf das Remote-Repository gepusht werden kann. Sind in der Zwischenzeit von einer anderen Person Änderungen gemacht worden, ist vorerst ein weiterer Pull nötig. Sind diese Änderungen an denselben Dateien passiert, die auch die erste Person verändert hat, dann treten Merge-Konflikte auf. Dies sollte daher vermieden werden. Durch eine gute Strukturierung des Repos und Aufteilung des Codes in viele Dateien sinkt die Wahrscheinlichkeit, dass meherere Personen an der gleichen Datei arbeiten wollen. Darüber hinaus sollte abgesprochen werden, wer wann wo Änderungen vornimmt.

Tabelle 4.1: Ausschnitt aus dem Git-Log

Nr	Datum	Autor	Nachricht	Kommentar
1	06.09.2024 16:42	Laura Halbeck	Erster Monitoring-Versuch hinzugefügt	
2	06.09.2024 18:39	Laura Halbeck	Monitoring mit Durchschnittswerten + Kleinere Probleme gefixt	
3	07.09.2024 00:26	Fabian Tober	frontend hinzugefügt ohne Karte ohne POI	
4	07.09.2024 01:31	Fabian Tober	erster Versuch schiffsdaten zu erhalten - frontend bezogen	
5	07.09.2024 01:37	Fabian Tober	eine karte ohne Funktion hinzugefügt	
6	07.09.2024 13:12	Laura Halbeck	Monitoring Probleme gefixt	Anpassungen, nachdem das Monitoring wegen falsch formatierter Daten über Nacht stoppte
7	07.09.2024 18:45	Laura Halbeck	Lasttest-Skripte hinzugefügt	
8	07.09.2024 20:05	Fabian Tober	pfad position_table.js geändert	Anpassung des CGI-Pfads, damit er universell für alle Docker funktioniert
9	08.09.2024 19:39	Fabian Tober	Schiffe als Tabelle dastellen - Javascript für verbindung von Backend und Frontend hinzugefügt	
10	08.09.2024 20:34	Laura Halbeck	Lasttest optimiert	Auswirkung auf Systemlast des Lasttest deutlich reduziert
11	08.09.2024 21:37	Fabian Tober	maps mit poliline eingefügt	

Tabelle 4.1 zeigt einen Ausschnitt aus unserer Git-Historie. Der Großteil des Backends war zu diesem Zeitpunkt schon vorhanden. Während dieses Wochenendes wurde dann das Frontend

vom Platzhalter zur funktionsfähigen Anwendung, an der später nur noch kleinere Anpassungen gemacht werden mussten. Außerdem wurde das Monitoring erstellt und erstmals beobachtet, und die Lasttests wurden durchgeführt. Da längere Treffen zeitlich nicht machbar waren, teilten wir Aufgaben auf und arbeiteten dann, wenn wir jeweils Zeit hatten. Die Änderungen wurden dann beim nächsten Treffen gemeinsam besprochen. Wenn dabei noch Änderungsvorschläge aufkamen, wurden diese während des Treffens von einer Person committet. Commit Nummer 8 und Commit Nummer 10 aus der Tabelle sind zum Beispiel Änderungen, die aus dem gemeinsamen Besprechen des Codes entstanden sind.

5 Beobachtungen

5.1 Monitoring

5.1.1 Aufbau

Wir entschieden uns, für unser Monitoring einmal pro Minute die folgenden Daten zu messen:

- Aktuell eingeloggte Nutzer (Anzahl Dateien in */usr/lib/cgi-bin/cookies/*)
- CPU-Last des Systems (aus dem Programm *uptime*)
- Nachrichten von Rhodes pro Minute (Differenz der Anzahl Zeilen in der Datenbank-Tabelle *Positionsmeldung*)
- HTTP-Requests an den Server pro Minute (Differenz der Anzahl Zeilen in */var/log/apache2/access.log*)

Diese Daten direkt zu plotten führte zu einem sehr unleserlichen Diagramm, daher entschieden wir uns, sie zunächst in 15-Minuten Blöcke einzuteilen und dort den Durchschnittswert zu nehmen. Das Mess-Skript und das Plot-Skript laufen regelmäßig als Cron-Jobs, und das generierte Diagramm wird laufend in */var/www/html/<USER>-web/monitoring/* kopiert.

5.1.2 Ergebnisse

In Abbildung 5.1 ist ein Beispiel eines 24-Stunden-Monitorings auf dem Docker zu sehen. Während dieser 24 Stunden führten wir keine Tests durch, also liegen die eingeloggten Nutzer bei null und die HTTP-Requests fast durchgehend bei null. Dies stellt also die geringste Last des Systems dar. Ein Monitoring während eines Lasttests zeigen wir im nächsten Abschnitt.

Anzumerken ist, dass zwischen 3:30 und 4:00 Uhr immer ein Datenpunkt der HTTP-Requests fehlt. Als wir dies untersuchten, fanden wir heraus, dass um diese Uhrzeit auf den Dockern die

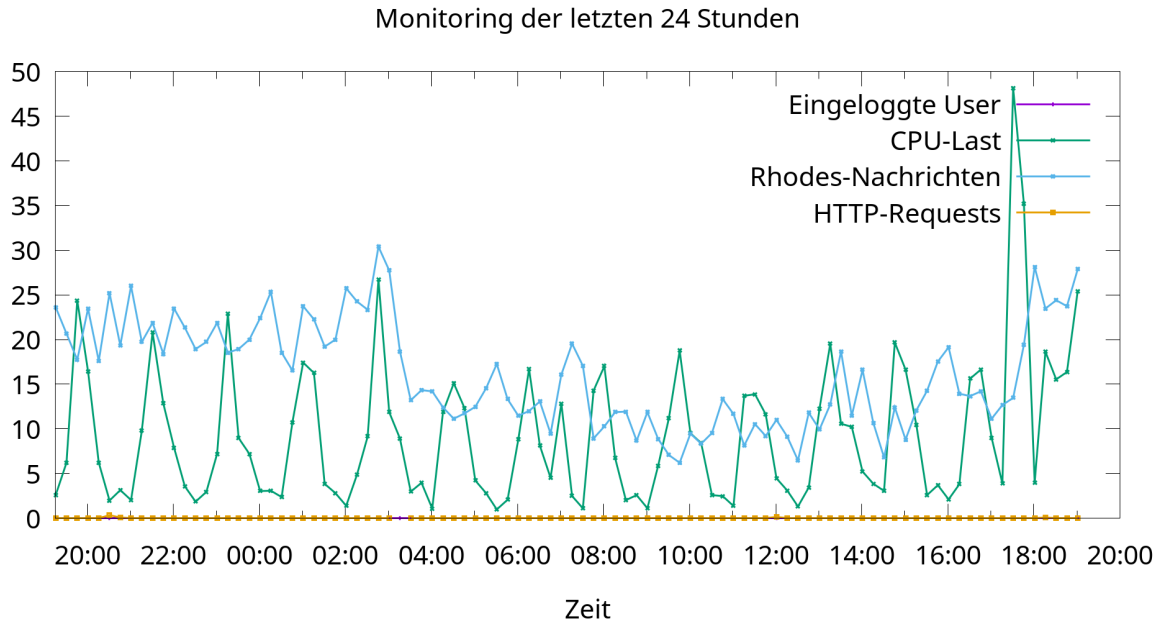

Abbildung 5.1: 24-Stunden-Monitoring bei geringer Last

Apache-Logs geleert werden und die Differenz der Zeilen damit negativ ist. Da die anderen Datenpunkt weiterhin korrekt sind, akzeptierten wir dieses eine fehlende Datum pro Tag.

5.2 Lasttest

5.2.1 Aufbau

Ziel unseres Lasttests ist es, die Wartezeit auf das Senden der Bestätigungs-Mail nach dem Login in Verhältnis zu der Höhe des Nutzeraufkommens zu messen. Dazu simulierten wir per Bash-Skript mehrere Nutzer, die sich einloggen und dann wieder ausloggen. Unser Test besitzt dabei zwei variable Parameter: Die Anzahl der Nutzer und die Verzögerung zwischen Nutzern. Mit Parametern 50 und 0,2 wird zum Beispiel alle 0,2 Sekunden eine Nutzer-Simulation gestartet, bis insgesamt 50 Nutzer simuliert wurden (vgl. Listing 5.1).

```
1 for i in $(seq $users); do
2   testing/lasttest/simulate-user.sh "$i" &
3   sleep $delay
4 done
```

Listing 5.1: Starten der Simulationen

```
while test ! "$(cat "/home/$USER/sent-mails.txt" \
  | grep "user$userid@lasttest.de")"; do
  sleep 0.3
done
```

Listing 5.2: Ressourcenaufwändiges Warten auf Mail

```
userid="$1"

echo "user$userid $EPOCHREALTIME" >> "/home/$USER/lasttest/started.dat"

cookiefile="/home/$USER/lasttest/cookie$userid.txt"
curl -b "$cookiefile" -c "$cookiefile" \
  "localhost/$USER-web/cgi-bin/login.sh?email=user$userid@lasttest.de&password=demo
  ↪ "
  ↪ &>/dev/null
echo "user$userid $(cat "/home/$USER/mail-queue" | wc -l)" \
  >> "/home/$USER/lasttest/queuesize.dat"
curl -b "$cookiefile" -c "$cookiefile" \
  "localhost/$USER-web/cgi-bin/logout.sh" &>/dev/null

echo "user$userid $EPOCHREALTIME" >> "/home/$USER/lasttest/finished.dat"
```

Listing 5.3: Finale Version von simulate-user.sh

In der ersten Version unseres Tests wartete jede Nutzer-Simulation darauf, dass die Bestätigungs-Email erhalten wurde (siehe Listing 5.2). Dies hatte den Vorteil, dass jede Simulation am Ende nur eine Zeile Daten in eine Datei schrieb, die dann direkt an gnuplot weitergegeben werden konnte. Es hatte allerdings den Nachteil, dass jeder Nutzer solange, bis er die Mail erhalten hat, als Prozess weiterläuft und mehrmals die Sekunde kurz zwei weitere Prozesse startet, was bei großer Nutzerzahl und kleiner Verzögerung eine sehr große Last für das System verursacht und damit auch die Daten verfälscht. Daher startet in der finalen Version (vgl. Listing 5.3) jeder Nutzer nur die zwei curl-Aufrufe und loggt einzeln die Zeiten sowie die Größe der Queue. Wir passten dafür außerdem den Mail-Worker an, so dass der Zeitpunkt des Senden der Mails mit in sent-mails.txt geschrieben wird. Damit mussten wir nur noch beim Erstellen des Diagramms die Daten für jeden Nutzer sammeln und in einer neuen Datei in je eine Zeile bringen, bevor diese an gnuplot übergeben wird. Dies verbesserte die Systemauslastung erheblich.

5.2.2 Ergebnisse

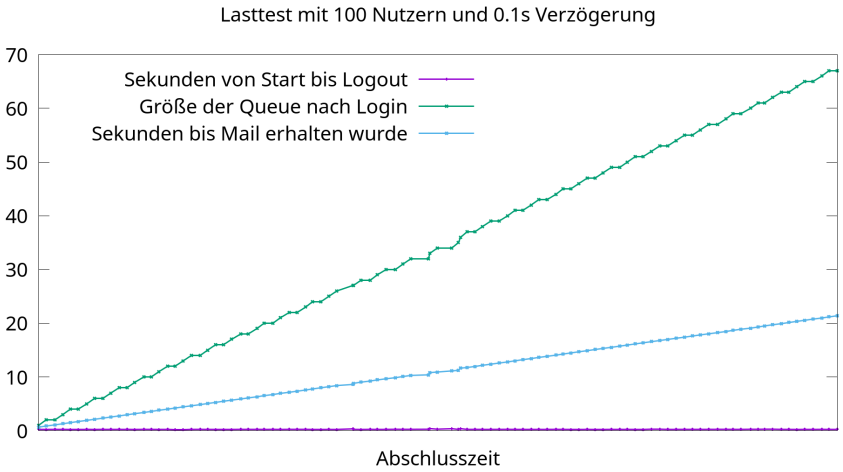

Abbildung 5.2: Lasttest mit 0,1 Sekunden Verzögerung

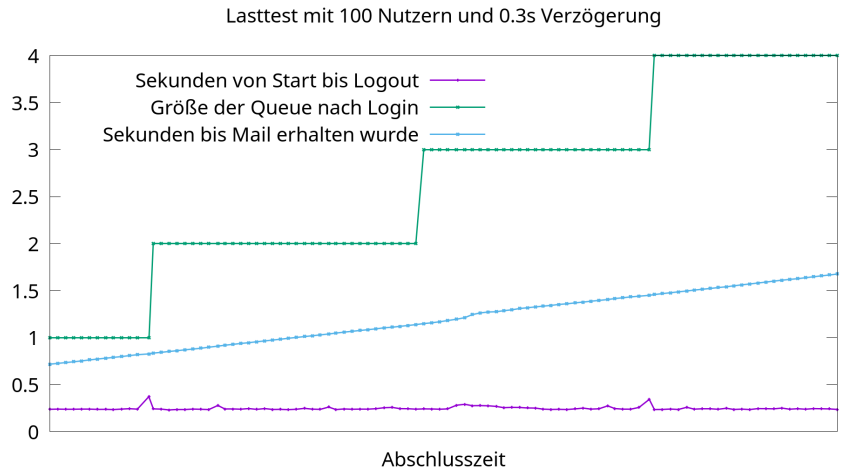

Abbildung 5.3: Lasttest mit 0,3 Sekunden Verzögerung

Die Abbildungen 5.2, 5.3 und 5.4 zeigen die drei aussagekräftigsten Tests, mit jeweils 100 Nutzern und verschiedenen Verzögerungen.

Aufgrund des *sleep 0.3* in unserem Mail-Worker dauert das Senden einer Mail bei leerer Queue etwas mehr als 0,3 Sekunden. Bei einer Verzögerung von 0,3 Sekunden oder weniger erhöht sich also mit der Zeit die Größe der Queue, wie auch in den Diagrammen zu sehen ist. Je kleiner die Verzögerung, desto schneller wächst die Wartezeit. Loggen sich wiederum alle 0,4 Sekunden Nutzer ein, muss keiner davon länger als eine Sekunde auf die Mail warten. Das sollte auch der

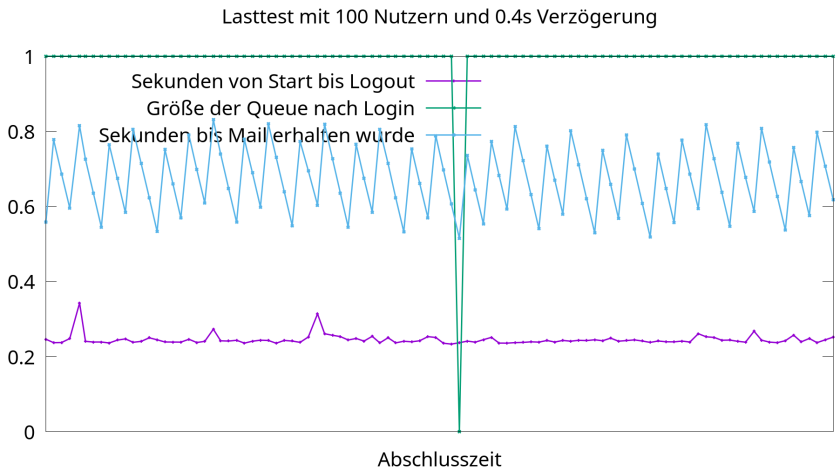

Abbildung 5.4: Lasttest mit 0,4 Sekunden Verzögerung

Fall sein, wenn diese Last den ganzen Tag über gleichmäßig besteht. Es lässt sich als schließen, dass das System für einen großen Andrang vieler Nutzer auf einmal nicht gut geeignet ist, wobei die höchste Wartezeit bei 100 Nutzern mit 0,1 Sekunden Verzögerung mit rund 20 Sekunden für das Erhalten einer Mail noch ertragbar ist. Führt man die Linie weiter mit der Annahme, dass sie der Funktion $f(x) = \frac{x}{5}$ gleicht, wären wir bei 10 000 Nutzern bei einer höchsten Wartezeit von rund 33 Minuten und bei 100 000 Nutzern wäre die höchste Verzögerung circa fünfeinhalb Stunden. Das Einsetzen mehrerer nebenläufiger Mail-Worker könnte bei diesem Problem Abhilfe schaffen.

Es sei anzumerken, dass ncat standardmäßig nur bis zu 20 gleichzeitige Verbindungen erlaubt. Diese Einstellung haben wir für unseren größten Lasttest auf 500 erhöht. Sollten sich mehr als 500 Nutzer in einem sehr kurzen Zeitraum einloggen, würde ncat jedoch weiterhin Verbindungen verweigern und somit würden manche Nutzer in diesem Fall überhaupt keine E-Mail erhalten.

Da in allen Diagrammen die Linie der Zeit von Start bis Logout, welche die Antwortzeit von zwei curl-Aufrufen beinhaltet, abgesehen von kleinen Variationen konstant zu sein scheint, dürfte es bei mindestens 0,1 Sekunden Verzögerung keine große Auswirkung auf die Systemlast geben, die sich in längeren Antwortzeiten zeigen würde. Trotzdem führten wir zusätzlich einen Lasttest mit 500 Nutzern und 0,2 Sekunden Verzögerung per Cron-Job alle zwei Stunden über einen ganzen Tag aus, um zu sehen, wie sich dies auf die CPU-Last auswirkt. Die Ergebnisse sind in Abbildung 5.5 zu sehen. Es lässt sich dort durch Vergleichen der Kurven von HTTP-Requests und CPU-Last keine sichtbare Auswirkung erkennen.

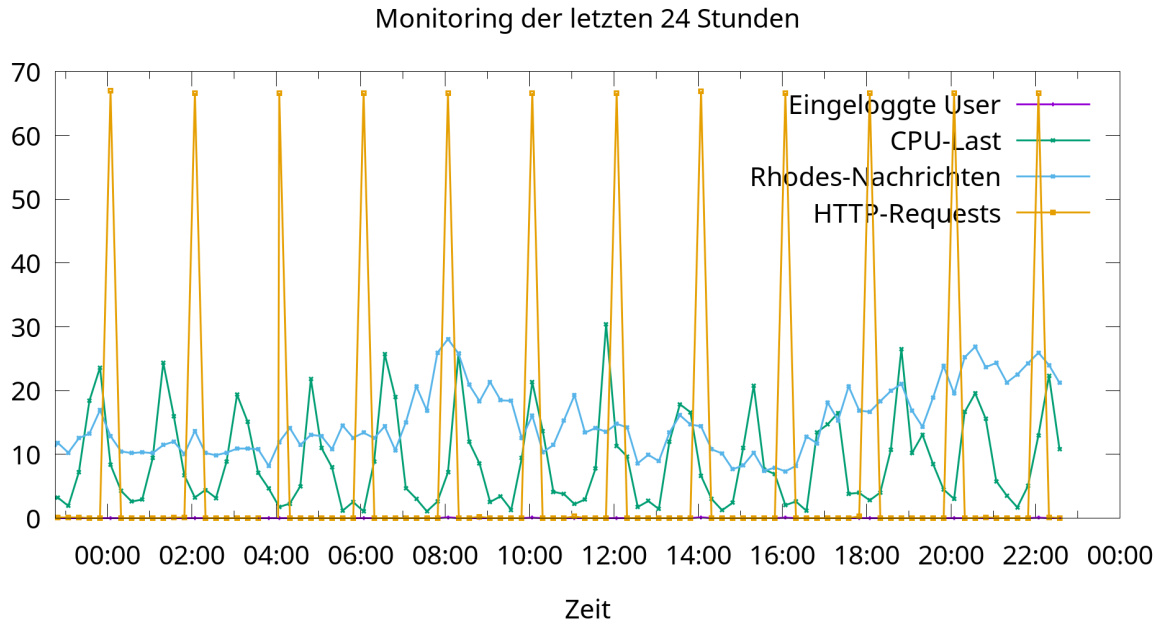

Abbildung 5.5: 24-Stunden-Monitoring bei wiederholt ausgeführtem Lasttest

6 Selbstreflexion

6.1 Laura Halbeck

Ich bin aktuell im 6. Semester meines Informatik-Studiums. Im 2. Semester nahm ich an den Vorlesungen zu diesem Modul bereits teil, kümmerte mich aber zu spät um die Gruppenfindung und konnte deshalb die Semesteraufgabe nicht mitmachen.

Meiner Einschätzung nach hatte ich schon im 2. Semester die Inhalte gut genug verstanden, dass ich die Prüfung zumindest bestanden hätte. Dieses Semester kam ich jedoch mit noch mal einer anderen Perspektive auf diese Inhalte zurück. Durch die Erfahrungen, die ich seitdem gemacht habe, kann ich sie nun in einen größeren Kontext einordnen. Insbesondere durch das Praxissemester und meiner darauf folgenden Werkstudententätigkeit im selben Unternehmen, bei der ich die Mitarbeit an einem großem und langlebigen Software-Projekt erlebe, sehe ich die Motivation hinter vielen Konzepten, die im Studium nur Theorie sind. So sehe ich zum Beispiel die Folgen wachsender technischer Schulden. Ich bin noch nicht soweit, dass ich in diesen Situationen klar sagen kann, wie diese Probleme zu lösen sind, kann nun aber anfangen, neu gelerntes als Tools zum Lösen solcher Probleme zu speichern. Das Bachelor-Projekt hat in dieser Hinsicht auch sehr geholfen, wobei ich dort auch Inhalte aus diesem Modul wie zum Beispiel gnuplot noch weiter üben konnte.

Die Wiederholung festigte außerdem viele Beispiele soweit, dass ich sie auswendig programmieren kann, was im 2. Semester noch nicht der Fall war. Auch wenn ich damals schon das Prinzip von Cron-Jobs verstanden hatte, muss ich heute nicht mehr nachschauen, mit welchen Schaltern man den crontab ausgibt oder in ihn schreibt, oder welche Zahlen ich wo schreiben muss, um eine Aufgabe alle 15 Minuten auszuführen. Dazu brauchte es zusätzlich zu den Vorlesungen auch das öftere Tippen über mehrere Tage während des Semesterprojekts. Es gibt aber immer noch andere Beispiele, die ich nicht auswendig kann, zum Beispiel die erste Zeile einer Gnuplot-Datei.

In Hinblick auf die Teamarbeit hat sich seit dem 2. Semester einiges verändert. Allgemein verbessern sich langsam aber sicher meine Kommunikationsfähigkeiten. Da ich schon vor dem Studienbeginn über viele Jahre das Reden mit anderen Menschen so gut wie möglich vermied, war der Ausgangspunkt nicht sonderlich gut. Durch teils private Veränderungen aber auch mehr Übung durch Studium und Arbeit geht es aber bergauf. Das größte Problem ist immer noch, meine Gedanken spontan in verständlichen Sätzen zu formulieren. Dort hilft zum Beispiel als Übung, zuhause laut das zu erklären, was ich gerade mache, allgemein sowie beim Coden.

Durch das öftere Reden merke ich auch, dass es das Verständnis weiter fördert, über die Themen frei zu sprechen. Ein gemeinsamer Austausch und Pair-Programming sind natürlich auch hilfreich und es werden Lösungen und Fehler so einfacher gefunden. Ich merke allerdings auch, dass ich mich während Gesprächen deutlich schlechter auf das Problem konzentrieren kann und mir viele Lösungen erst einfallen, wenn ich alleine drauf schaue. Dies könnte sich mit mehr Übung aber weiter verbessern.

Insgesamt habe ich im Hinblick auf mein Informatik-Studium inzwischen das Gefühl, auf einem guten Weg zu sein und diesen nur weiter gehen zu müssen.

6.2 Fabian Tober

Im Laufe dieses Semesters habe ich in vielen Bereichen Fortschritte gemacht, aber auch Herausforderungen erlebt. Im Folgenden möchte ich reflektieren, wie sich meine Fähigkeiten entwickelt haben, welche Methoden für mich besonders effektiv waren und wo ich noch Schwierigkeiten habe.

Zu Beginn des Semesters haben wir uns mit Technologien wie Git, Docker und MariaDB auseinandergesetzt. Während ich anfangs einige Zeit brauchte, um mich in die Nutzung von Git einzuarbeiten, beherrsche ich mittlerweile die grundlegenden Befehle wie `git add`, `commit`, `push` und `pull` aus dem Stegreif und stehe nicht vor einer unlösbaren Aufgabe wenn es ein Merge-Konflikt gib. Besonders die regelmäßige Aufteilung von Aufgaben in kleinere Einheiten hat mir persönlich viel gebracht. Dadurch habe ich auch ein besseres Verständnis für die Arbeit im Team entwickelt, da Konflikte bei der Zusammenführung von Code so minimiert wurden.

Docker stellt für mich derzeit keine große Herausforderung dar, da ich die grundlegenden Konzepte von Containern und ihren Nutzen für die Anwendungsentwicklung gut verstanden habe. Dieses Basiswissen ermöglicht es mir, mich weiter mit Docker zu beschäftigen und komplexere Aspekte zu erkunden. Ich sehe diesen Grundstein als wichtig an, um meine Fähigkeiten im Bereich Containerisierung weiterzuentwickeln und mich in den kommenden Semestern anspruchsvolleren Projekten zu widmen.

Besonders gut funktioniert bei mir die Methode, Dinge praktisch umzusetzen und „learning by doing“ anzuwenden. Theoretische Konzepte zu verstehen ist eine Sache, doch wirkliches Lernen findet für mich erst statt, wenn ich diese in die Praxis umsetze. Ein gutes Beispiel hierfür ist das Programmieren eines „Watchers“. Während ich zu Beginn Schwierigkeiten hatte, diesen zu Programmieren, gelang es mir nach einigen Versuchen und auswendiglernens, einen Watcher in weniger als fünf Minuten zu programmieren. Jedoch ist mir hierbei klar geworden, dass ich nicht immer alle einzelnen Teile wirklich verstehe. Es fällt mir oft leichter, das Gesamtbild zu sehen und auf Basis von bereits vorhandenem Code zu arbeiten, als die Feinheiten und die dahinterliegenden Konzepte vollständig zu durchdringen. Hier sehe ich für mich noch Verbesserungspotenzial.

Im Laufe des Semesters habe ich ungefähr 8 Stunden pro Woche in das Üben und Vertiefen meiner Kenntnisse investiert. Diese Zeit habe ich vor allem für praktische Übungen verwendet, um ein besseres Gefühl für die Technologien zu entwickeln, mit denen wir arbeiten. Gerade das eigenständige Einüben, sei es durch eigene Projekte oder durch das Bearbeiten von Teamaufgaben, hat mir geholfen, mein Wissen zu festigen. Dennoch merke ich, dass gerade bei umfangreicheren Projekten mehr Zeit notwendig ist, um wirklich tiefes Verständnis zu entwickeln.

Was die Teamarbeit betrifft, habe ich während des Semesters wertvolle Erkenntnisse gewonnen. Mir fiel es am Anfang leicht, mich ins Team einzubringen und meine Ideen aktiv zu teilen. Trotz meines vorherigen Wissens über die Bedeutung einer strukturierten Kommunikation wurde mir durch das Projekt noch klarer, wie essenziell eine gute Teamkoordination ist. Die praktische Erfahrung, die ich durch den Einsatz von Git und die Zusammenarbeit im Team gesammelt habe, hat mir gezeigt, wie wichtig regelmäßige Abstimmungen und klare Arbeitsaufteilung für den Projekterfolg sind. Diese Erfahrung hat nicht nur mein technisches Wissen erweitert, sondern auch meine Fähigkeiten in der Teamarbeit und Zusammenarbeit deutlich gestärkt.

In diesem Semester habe ich wertvolle Erkenntnisse über meinen Lernstil gewonnen. Neben dem Aufbau technologischer Fähigkeiten wurde mir klar, dass ich komplexe Konzepte im Detail verstehen muss. Während mir die praktische Umsetzung von Technologien wie Git und Docker gut gelang, sehe ich noch Potenzial beim tieferen Verständnis. Die Teamarbeit hat mir gezeigt, wie wichtig klare Kommunikation und effiziente Aufgabenverteilung für den Erfolg sind, was sowohl meine technischen als auch meine Teamfähigkeiten gestärkt hat.

Ich freue mich darauf, meine Kenntnisse im kommenden Semester weiter auszubauen und an neuen Herausforderungen zu wachsen.

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Anhang

I Anwendung

Die Anwendung läuft in unserem Team-Docker und ist unter https://informatik.hs-bremerhaven.de/docker-infra-2024-e-web/ erreichbar. Mit der E-Mail *demo@user.de* und dem Passwort *demo* ist ein Login möglich.

Das laufende Monitoring befindet sich unter https://informatik.hs-bremerhaven.de/docker-infra-2024-e-web/monitoring/.

II Vollständiger Quellcode

```
#!/bin/bash

ssh mydocker rm -rf infra-2024-e
tar -C .. -cf- infra-2024-e | (ssh mydocker tar -C . -xvf-)
ssh mydocker "cd infra-2024-e; bin/deploy-all.sh"
```

Listing II.1: ./bin/deploy-all-to-docker.sh

```
#!/bin/bash

bin/stop-all.sh
bin/deploy-db.sh
bin/deploy-rhodes-watcher.sh
bin/deploy-mail.sh
bin/deploy-cgi.sh
bin/deploy-frontend.sh
bin/deploy-monitoring.sh
```

Listing II.2: ./bin/deploy-all.sh

```
#!/bin/bash

cp -r cgi/* /usr/lib/cgi-bin/
mkdir /usr/lib/cgi-bin/private
cp ~/.my.cnf /usr/lib/cgi-bin/private/
chmod +r /usr/lib/cgi-bin/private/.my.cnf
```

```
mkdir /usr/lib/cgi-bin/cookies
chmod a+rw /usr/lib/cgi-bin/cookies
```

Listing II.3: ./bin/deploy-cgi.sh

```
#!/bin/bash

# Datenbank erstellen
./db-create/db-create.sh

# DB Cleanup starten
mkdir -p "/home/$USER/db-cleanup"
cp db-cleanup/db-cleanup.sh "/home/$USER/db-cleanup/db-cleanup.sh"
(crontab -l; echo "0 0 * * * USER=$USER /home/$USER/db-cleanup/db-cleanup.sh") |
↪  sort -u | crontab -
```

Listing II.4: ./bin/deploy-db.sh

```
#!/bin/bash

cp -r frontend/* /var/www/html/$USER-web/ 2>/dev/null
```

Listing II.5: ./bin/deploy-frontend.sh

```
#!/bin/bash

server_dir="/home/$USER/mail-server"
worker_dir="/home/$USER/mail-worker"

touch "/home/$USER/mail-queue"

# Mail-Server starten
mkdir "$server_dir"
cp -r mail-server/* "$server_dir"
ncat -e "$server_dir/server-listen.sh" -m 500 -lk 1234 &
echo "$!" > "$server_dir/server.pid"

```

```
# Mail-Worker starten
mkdir "$worker_dir"
cp -r mail-worker/* "$worker_dir"
"$worker_dir/worker.sh" &
echo "$!" > "$worker_dir/worker.pid"
```

Listing II.6: ./bin/deploy-mail.sh

```
#!/bin/bash

monitoring_dir="/home/$USER/monitoring"

mkdir -p "/var/www/html/$USER-web/monitoring"
cp monitoring/index.html "/var/www/html/$USER-web/monitoring/"

mkdir -p "$monitoring_dir"
cp -r monitoring/* "$monitoring_dir"
sed -i "s/___INPUT___/\/home\/$USER\/monitoring\/data-averaged.dat/g"
↪  "$monitoring_dir"/plot.gp
sed -i "s/___OUTPUT___/\/home\/$USER\/monitoring\/plot.png/g"
↪  "$monitoring_dir"/plot.gp
echo 0 > "$monitoring_dir"/rhodes_last.txt
echo $(cat /var/log/apache2/access.log | wc -l) >
↪  "$monitoring_dir"/requests_last.txt

"$monitoring_dir/measure.sh"
"$monitoring_dir/plot.sh"

# Messen einmal pro Minute, Graph einmal alle 15 Minuten
(crontab -l; echo "* * * * * USER=$USER $monitoring_dir/measure.sh") | sort -u |
↪  crontab -
(crontab -l; echo "0,15,30,45 * * * * USER=$USER $monitoring_dir/plot.sh") | sort
↪  -u | crontab -
```

Listing II.7: ./bin/deploy-monitoring.sh

```
#!/bin/bash

```

```
3  rhodes_dir="/home/$USER/rhodes"
4
5  mkdir "$rhodes_dir"
6  cp -r rhodes-watcher/* "$rhodes_dir"
7  (crontab -l; echo "* * * * * USER=$USER $rhodes_dir/watcher.sh") | sort -u |
   ↪  crontab -
8  $rhodes_dir/watcher.sh
```

Listing II.8: ./bin/deploy-rhodes-watcher.sh

```
1  #!/bin/bash
2
3  bin/stop-db.sh
4  bin/stop-rhodes-watcher.sh
5  bin/stop-mail.sh
6  bin/stop-cgi.sh
7  bin/stop-frontend.sh
8  bin/stop-monitoring.sh
```

Listing II.9: ./bin/stop-all.sh

```
1  #!/bin/bash
2
3  rm -rf /usr/lib/cgi-bin/*
```

Listing II.10: ./bin/stop-cgi.sh

```
1  #!/bin/bash
2
3  rm -rf "/home/$USER/db-cleanup"
4  crontab -l | grep -v "db-cleanup.sh" | crontab -
```

Listing II.11: ./bin/stop-db.sh

```
#!/bin/bash

rm -rf /var/www/html/$USER-web/*
```

Listing II.12: ./bin/stop-frontend.sh

```
#!/bin/bash

server_dir="/home/$USER/mail-server"
worker_dir="/home/$USER/mail-worker"

rm -rf "/home/$USER/mail-queue"
rmdir "/home/$USER/mail-queue.lock" 2>/dev/null

# Mail-Server stoppen
if test -f "$server_dir/server.pid"; then
  kill $(cat "$server_dir/server.pid") 2>/dev/null
fi
rm -rf "$server_dir"

# Mail-Worker stoppen
rm -f "/home/$USER/sent-mails.txt"
if test -f "$worker_dir/worker.pid"; then
  kill $(cat "$worker_dir/worker.pid") 2>/dev/null
fi
rm -rf "$worker_dir"
```

Listing II.13: ./bin/stop-mail.sh

```
#!/bin/bash

monitoring_dir="/home/$USER/monitoring"
rm -rf "$monitoring_dir"

crontab -l | grep -v "measure.sh" | crontab -
crontab -l | grep -v "plot.sh" | crontab -
```

Listing II.14: ./bin/stop-monitoring.sh

```
#!/bin/bash

rhodes_dir="/home/$USER/rhodes"

if test -f "$rhodes_dir/rhodes.pid"; then
  kill $(cat "$rhodes_dir/rhodes.pid") 2>/dev/null
fi
rm -rf "$rhodes_dir"
crontab -l | grep -v "watcher.sh" | crontab -
```

Listing II.15: ./bin/stop-rhodes-watcher.sh

```
#!/bin/bash

# .../check-cookie.sh (Cookie wird aus dem Header entnommen)
# => gibt bei bestehender Session die Email des Nutzers aus
# => gibt sonst nichts aus

echo "Content-Type: text/plain"
echo

session=$(echo $HTTP_COOKIE | grep -Eo 'infra2024-e-session=[^;]+' | cut -d '=' -f
↪  2)
email=$(cat cookies/"$session" 2>/dev/null)
if test "$email"; then
  echo "$email"
fi
```

Listing II.16: ./cgi/check-cookie.sh

```
#!/bin/bash

# .../get-positions.sh => gibt gesamten Inhalt der Tabelle Positionsmeldung zurück
# .../get-positions.sh?last_id=<ID> => gibt nur Zeilen mit id größer als <ID> zurück

echo "Content-Type: text/plain"
```

```
7  echo
8
9  last_id=$(echo $QUERY_STRING | grep -Eo 'last_id=[^&]+' | cut -d '=' -f 2)
10 if test "$last_id"; then
11   query="SELECT * from Positionsmeldung WHERE id > $last_id;"
12 else
13   query="SELECT * from Positionsmeldung;"
14 fi
15 mariadb --defaults-file="private/.my.cnf" -e "$query"
```

Listing II.17: ./cgi/get-positions.sh

```
1  #!/bin/bash
2
3  # .../get-ships.sh => gibt gesamten Inhalt der Tabelle Schiff zurück
4
5  echo "Content-Type: text/plain"
6  echo
7
8  query="SELECT * FROM Schiff;"
9  mariadb --defaults-file="private/.my.cnf" -e "$query"
```

Listing II.18: ./cgi/get-ships.sh

```
1  #!/bin/bash
2
3  # .../login.sh?email=<EMAIL>&password=<PASSWORD>
4  # => gibt bei erfolgreichem Login die Email des Nutzers aus (und setzt über den
   ↪  Response Header den Cookie)
5  # => gibt bei fehlgeschlagenem Login nichts aus
6
7  # Prüfen, ob Nutzer mit Email und Password existiert
8  email=$(echo $QUERY_STRING | grep -Eo 'email=[^&]+' | cut -d '=' -f 2)
9  password=$(echo $QUERY_STRING | grep -Eo 'password=[^&]+' | cut -d '=' -f 2)
10 query="SELECT id FROM User WHERE email = '$email' AND password = '$password';"
11 result=$(mariadb --defaults-file="private/.my.cnf" -e "$query")
12
13 if test "$result"; then
```

```
14    # Cookie setzen in Browser und Datei
15    session="$(pwgen 40 1)"
16    echo "Set-Cookie:infra2024-e-session=$session; Secure; HttpOnly; SameSite=Strict;"
17    echo "$email" > cookies/"$session"
18
19    # Mail-Anfrage schicken
20    ncat -e "send-mail-request.sh $email" 127.0.0.1 1234
21
22    echo "Content-Type: text/plain"
23    echo
24    echo "$email"
25  else
26    echo "Content-Type: text/plain"
27    echo
28  fi
```

Listing II.19: ./cgi/login.sh

```
1   #!/bin/bash
2
3   # .../logout.sh (Cookie wird aus dem Header entnommen)
4   # => gibt nichts aus
5
6   echo "Content-Type: text/plain"
7   echo
8
9   session=$(echo $HTTP_COOKIE | grep -Eo 'infra2024-e-session=[^;]+' | cut -d '=' -f
    ↪  2)
10  if test "$session"; then
11    rm cookies/"$session"
12  fi
```

Listing II.20: ./cgi/logout.sh

```
1   #!/bin/bash
2
3   mail="$1"
4
```

```
5  echo "$mail|Login-Bestätigung|Sie haben sich erfolgreich eingeloggt. Herzlichen
   ↪  Glückwunsch!"
6
7  read line
8  if test "$line" == "received"; then
9    exit 0
10 fi
11 exit 1
```

Listing II.21: ./cgi/send-mail-request.sh

```
1  #!/bin/bash
2
3  # Lösche alle Positionsmeldungen älter als 7 Tage
4
5  query="DELETE FROM Positionsmeldung WHERE TIMESTAMPDIFF(DAY,time,NOW()) >= 7"
6  mariadb -e "$query"
```

Listing II.22: ./db-cleanup/db-cleanup.sh

```
1  #!/bin/bash
2  query="
3  DROP TABLE IF EXISTS User;
4  DROP TABLE IF EXISTS Schiff;
5  DROP TABLE IF EXISTS Positionsmeldung;
6
7  CREATE TABLE User (
8    id int AUTO_INCREMENT NOT NULL,
9    email varchar(50) NOT NULL UNIQUE,
10   password varchar(50) NOT NULL,
11   PRIMARY KEY(id)
12 );
13 CREATE TABLE Schiff (
14   mmsi varchar(9) NOT NULL,
15   name varchar(50),
16   PRIMARY KEY(mmsi)
17 );
18 CREATE TABLE Positionsmeldung (
```

```
  id int AUTO_INCREMENT NOT NULL,
  time timestamp NOT NULL,
  mmsi varchar(9) NOT NULL,
  latitude float,
  longitude float,
  PRIMARY KEY(id)
);

INSERT INTO User (id, email, password) VALUES
  (0, 'demo@user.de', 'demo');
"
mariadb -e "$query"
```

Listing II.23: ./db-create/db-create.sh

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    min-height: 100vh;
    background-color: #f4f4f4;
}

h1, h2 {
    text-align: center;
    margin: 0;
    padding: 10px 0;
}

h1 {
    font-size: 2em;
}

h2 {
    font-size: 1.5em;
    margin-bottom: 20px;
}
```

```
26
27  .accordion {
28      width: 80vw;
29      max-width: 600px;
30      border: 1px solid #ccc;
31      border-radius: 5px;
32      background-color: #fff;
33      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
34      display: flex;
35      flex-direction: column;
36  }
37
38  .accordion-item {
39      border-top: 1px solid #ccc;
40  }
41
42  .accordion-header {
43      background-color: #f1f1f1;
44      padding: 15px;
45      cursor: pointer;
46      font-size: 1.2em;
47  }
48
49  .accordion-content {
50      display: none;
51      padding: 15px;
52      background-color: #fff;
53  }
54
55  .accordion-content.open {
56      display: block;
57  }
58
59  .hidden {
60      display: none;
61  }
62
63  #map { height: 350px; }
64
65  table {
66      border-collapse: collapse;
```

```
67      width: 100%;
68      font-family: Arial, sans-serif;
69  }
70
71  th, td {
72      border: 1px solid black;
73      padding: 8px;
74      text-align: left;
75  }
76
77  th {
78      background-color: #f2f2f2;
79  }
```

Listing II.24: ./frontend/css/index.css

```
1   <!DOCTYPE html>
2   <html lang="de">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>Aktuelle Schiffe in Bremerhaven | HS Bremerhaven von infra-2024-e</title>
7       <link rel="stylesheet" href="/docker-infra-2024-e-web/css/index.css">
8       <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css"
9       integrity="sha256-p4NxAoJBhIIN+hmNHrzRCf9tD/miZyoHS5obTRR9BMY="
10      crossorigin=""/>
11      <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"
12      integrity="sha256-20nQCchB9co0qIjJZRGuk2/Z9VM+kNiyxNV1lvTlZBo="
13      crossorigin=""></script>
14  </head>
15  <body>
16
17      <h1>Aktuelle Schiffe in Bremerhaven</h1>
18      <h2>von infra-2024-e | 2. Semester</h2>
19      <div class="accordion">
20      <!-- Login/Ausloggen Akkordeon -->
21      <div class="accordion-item" id="loginLogoutSection">
22          <div class="accordion-header" onclick="toggleAccordion(this)">
23              Login
24          </div>
```

```
25          <div class="accordion-content">
26              <form id="loginForm">
27                  <label for="email">E-Mail:</label><br>
28                  <input type="email" id="email" name="email"><br><br>
29                  <label for="password">Passwort:</label><br>
30                  <input type="password" id="password" name="password"><br><br>
31                  <button type="button" onclick="handleLogin()">Login</button>
32                  <p id="loginError" class="hidden">Ungültige E-Mail oder
                    ↪  Passwort!</p>
33              </form>
34          </div>
35      </div>
36
37      <div class="accordion-item" id="gruppe2">
38          <div class="accordion-header" onclick="toggleAccordion(this)">
39              Karte
40          </div>
41          <div class="accordion-content">
42              <div id="map"></div>
43          </div>
44      </div>
45
46      <div class="accordion-item" id="gruppe3">
47          <div class="accordion-header" onclick="toggleAccordion(this)">
48              Schiff-Tabelle
49          </div>
50          <div class="accordion-content">
51              <table id="schiffeTabelle" border="1">
52                  <thead>
53                      <tr>
54                          <th>MMSI</th>
55                          <th>Ship Name</th>
56                      </tr>
57                  </thead>
58                  <tbody id="schiffeDaten">
59                      <!-- Schiffsdaten werden dynamisch eingefügt -->
60                  </tbody>
61              </table>
62          </div>
63      </div>
64
```

```
65      <div class="accordion-item" id="logoutSection" class="hidden">
66          <div class="accordion-header" onclick="toggleAccordion(this)">
67              Logout
68          </div>
69          <div class="accordion-content">
70              <p id="userName">Willkommen, <span id="displayName"></span>!</p>
71              <button type="button" onclick="handleLogout()">Logout</button>
72          </div>
73      </div>
74  </div>
75
76  <script src="js/index.js"></script>
77  <script src="js/login.js"></script>
78  <script src="js/position_table.js"></script>
79  <script src="js/maps.js"></script>
80
81  </body>
82  </html>
```

Listing II.25: ./frontend/index.html

```
1   function toggleAccordion(header) {
2       const content = header.nextElementSibling;
3       const allContents = document.querySelectorAll('.accordion-content');
4
5       allContents.forEach(c => {
6           if (c !== content) {
7               c.classList.remove('open');
8           }
9       });
10
11      content.classList.toggle('open');
12  }
```

Listing II.26: ./frontend/js/index.js

```
1   // Funktion zum Abfragen von Cookies
2   function getCookie(name) {
```

```
3       const value = `; ${document.cookie}`;
4       const parts = value.split(`; ${name}=`);
5       if (parts.length === 2) return parts.pop().split(';').shift();
6   }
7
8   // Funktion zur Überprüfung der Session beim Laden der Seite
9   function checkSession() {
10      fetch('./cgi-bin/check-cookie.sh')
11          .then(response => response.text())
12          .then(data => {
13              if (data) {
14                  // Benutzer ist eingeloggt
15                  document.getElementById('gruppe2').classList.remove('hidden');
16                  document.getElementById('gruppe3').classList.remove('hidden');
17                  document.getElementById('displayName').textContent = data;
18                  document.getElementById('loginLogoutSection').style.display =
                    ↪  'none';
19                  document.getElementById('logoutSection').style.display = 'block';
20              }
21          });
22  }
23
24  // Funktion für den Login-Prozess
25  function handleLogin() {
26      const email = document.getElementById('email').value;
27      const password = document.getElementById('password').value;
28
29      // Anfrage an das Login-Skript senden
30      fetch(`./cgi-bin/login.sh?email=${email}&password=${password}`)
31          .then(response => response.text())
32          .then(data => {
33              if (data) {
34                  // Login erfolgreich
35                  document.getElementById('gruppe2').classList.remove('hidden');
36                  document.getElementById('gruppe3').classList.remove('hidden');
37                  document.getElementById('displayName').textContent = data;
38                  document.getElementById('loginForm').reset();
39                  document.getElementById('loginLogoutSection').style.display =
                    ↪  'none';
40                  document.getElementById('logoutSection').style.display = 'block';
41              } else {
```

```
                // Fehler beim Login
                document.getElementById('loginError').classList.remove('hidden');
            }
        });
}

// Funktion für den Logout-Prozess
function handleLogout() {
    fetch('./cgi-bin/logout.sh')
        .then(() => {
            // Nach Logout den Zustand zurücksetzen
            document.getElementById('gruppe2').classList.add('hidden');
            document.getElementById('gruppe3').classList.add('hidden');
            document.getElementById('logoutSection').style.display = 'none';
            document.getElementById('loginLogoutSection').style.display = 'block';
            document.getElementById('displayName').textContent = '';
        });
}

document.getElementById('gruppe2').classList.add('hidden');
document.getElementById('gruppe3').classList.add('hidden');
document.getElementById('logoutSection').classList.add('hidden');

// Aufruf beim Laden der Seite, um die Session zu überprüfen
window.onload = checkSession;
```

Listing II.27: ./frontend/js/login.js

```
// Initialisiere die Karte
var map = L.map('map').setView([53.550749, 8.585308], 13);
var tileLayer = L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 19,
    center: [0,0],
    // tileSize: 256,
    preverCanvas: true,
    worldCopyJump: true,
    attribution: '&copy; <a
    ↪  href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
}).addTo(map);

```

```
12 // Marker und Polylinien für Schiffe
13 var markers = {};  // Speichert die Marker für jedes Schiff, identifiziert durch
   ↪ MMSI
14 var polylines = {};  // Speichert die Polylinien für jedes Schiff
15
16 var lastId = 0; // Letzte Erhaltene Datensatz-ID
17
18 // Funktion zum Aktualisieren der Schiffsdaten
19 function updateShips() {
20     fetch(`./cgi-bin/get-positions.sh?last_id=${lastId}`)  // Aufruf des CGI-Skripts
21         .then(response => response.text())
22         .catch(error => console.error('Fehler beim Abrufen der Positionsdaten:',
           ↪ error))
23         .then(data => {
24             var ships = parseShipData(data);  // Parse die Schiffsdaten
25             updateMap(ships);  // Aktualisiere die Karte
26         })
27         .catch(error => console.error('Fehler bei der Verarbeitung:', error));
28 }
29
30 // Funktion, um die empfangenen Schiffsdaten zu parsen
31 function parseShipData(data) {
32     var lines = data.trim().split('\n');  // Teilt den Text in Zeilen
33     var ships = {};
34
35     lines.forEach(line => {
36         var columns = line.split('\t');  // Teilt jede Zeile in Spalten
37         var id = columns[0];
38         var timestamp = new Date(columns[1]);
39         var mmsi = columns[2];
40         var latitude = parseFloat(columns[3]);
41         var longitude = parseFloat(columns[4]);
42
43         if (isNaN(latitude) || isNaN(longitude) || notInBremerhaven(latitude,
           ↪ longitude)) {
44             return; // keine nützlichen Daten, überspringen
45         }
46
47         // Wenn das Schiff mit der MMSI noch nicht existiert, initialisiere es
48         if (!ships[mmsi]) {
49             ships[mmsi] = {
```

```
50                  coords: [],
51                  lastPositionTime: timestamp
52              };
53          }
54
55          // Füge die Koordinaten hinzu
56          ships[mmsi].coords.push([latitude, longitude]);
57          ships[mmsi].lastPositionTime = timestamp;
58
59          lastId = id;
60      });
61
62      return ships;
63  }
64
65  function notInBremerhaven(latitude, longitude) {
66      return latitude < 53 || latitude > 54 || longitude < 8 || longitude > 9;
67  }
68
69  // Funktion zum Aktualisieren der Karte mit neuen Schiffsdaten
70  function updateMap(ships) {
71      var now = new Date();
72
73      // Über alle empfangenen Schiffe iterieren
74      for (var mmsi in ships) {
75          var ship = ships[mmsi];
76          var coords = ship.coords;
77
78          var lastPositionTime = ship.lastPositionTime;
79          // Wenn das Schiff länger als 5 Minuten inaktiv ist, entferne es von der
            ↪ Karte
80          var timeDiff = (now - lastPositionTime) / (1000 * 60);  // Zeitunterschied
            ↪ in Minuten
81          if (timeDiff > 5) {
82              if (markers[mmsi]) {
83                  map.removeLayer(markers[mmsi]);  // Entferne den Marker
84              }
85              if (polylines[mmsi]) {
86                  map.removeLayer(polylines[mmsi]);  // Entferne die Polylinie
87              }
88              delete markers[mmsi];
```

```
89              delete polylines[mmsi];
90              continue;  // Überspringe das Schiff
91          }
92
93          // Wenn das Schiff schon einen Marker hat, bewege den Marker nur, sonst
            ↪  erstelle einen neuen Marker
94          if (markers[mmsi]) {
95              markers[mmsi].setLatLng(coords[coords.length - 1]);  // Setze den Marker
                ↪  auf die letzte Position
96          } else {
97              var obj = coords[coords.length - 1];
98              markers[mmsi] = L.marker(L.latLng(obj[0], obj[1]));
99              markers[mmsi].addTo(map);
100             markers[mmsi].bindTooltip(mmsi);
101         }
102         // Wenn das Schiff schon eine Polyline hat, füge neue Punkte hinzu, sonst
            ↪  erstelle eine neue Polyline
103         if (polylines[mmsi]) {
104             coords.forEach(coord => {
105                 polylines[mmsi].addLatLng(coord);  // Füge neue Koordinaten zur
                    ↪  Polyline hinzu
106             });
107         } else {
108             polylines[mmsi] = L.polyline(coords, { color: 'blue' }).addTo(map);  //
                ↪  Erstelle eine neue Polyline
109         }
110
111     }
112 }
113
114 // Initiales Laden der Schiffsdaten
115 updateShips();
116
117 // Karte jede Sekunde aktualisieren
118 setInterval(updateShips, 1000);
```

Listing II.28: ./frontend/js/maps.js

```
1  function fetchShipData() {
2      const xhr = new XMLHttpRequest();
```

```
3       xhr.open('GET', './cgi-bin/get-ships.sh', true);
4
5       xhr.onload = function () {
6           if (this.status === 200) {
7               const ships = this.responseText.trim().split('\n').slice(1); //
                ↪  Überspringen der Kopfzeile
8               const tbody = document.getElementById('schiffeDaten');
9               tbody.innerHTML = ''; // Tabelle leeren, um neue Daten einzufügen
10
11              ships.forEach(ship => {
12                  const shipData = ship.split('\t'); // Daten sind durch Tabs getrennt
13                  const row = document.createElement('tr');
14
15              // Überprüfen und Bereinigen der "Name" Spalte
16              if (isNaN(shipData[1]) && shipData[1] !== 'undefined' && shipData[1]
                ↪  !== 'null' && shipData[1] !== 'nan') {
17                  shipData[1] = shipData[1]; // Behalte den Namen, wenn es keine Zahl
                    ↪  oder undefined/null/nan ist
18               } else {
19                  shipData[1] = ''; // Leere den Namen
20              }
21
22                  // Füge die Zellen für die Tabelle hinzu
23                  shipData.forEach(data => {
24                      const cell = document.createElement('td');
25                      cell.textContent = data;
26                      row.appendChild(cell);
27                  });
28
29                  tbody.appendChild(row);
30              });
31          }
32      };
33
34      xhr.send();
35  }
36
37  // Sekündliche Aktualisierung der Schiffsdaten
38  document.addEventListener('DOMContentLoaded', function () {
39      fetchShipData();
40      setInterval(fetchShipData, 1000); // 1000 ms = 1 Sekunde
```

```
41 });
```

Listing II.29: ./frontend/js/position$_table.js$

```
1  #!/bin/bash
2
3  queuefile="/home/$USER/mail-queue"
4  queuelock="/home/$USER/mail-queue.lock"
5
6  while read line; do
7    # Auf freien Lock warten, dann erhaltene Zeile in Queue schreiben
8    while ! mkdir "$queuelock" 2>/dev/null; do sleep 0.1; done
9    echo "$line" >> "$queuefile"
10   rmdir "$queuelock"
11
12   # Bestätigung
13   echo "received"
14 done
```

Listing II.30: ./mail-server/server-listen.sh

```
1  #!/bin/bash
2
3  queuefile="/home/$USER/mail-queue"
4  queuelock="/home/$USER/mail-queue.lock"
5
6  while true; do
7    # Auf freien Lock warten, dann nächste E-Mail aus Queue entnehmen
8    while ! mkdir "$queuelock" 2>/dev/null; do sleep 0.1; done
9
10   # Erste Zeile lesen
11   first=$(cat "$queuefile" | head -n 1)
12
13   # Nur wenn Zeile nicht leer ist:
14   # Simulation des Versendens der Mail durch Schreiben in Datei
15   if test "$first"; then
16     echo "$EPOCHREALTIME $first" >> "/home/$USER/sent-mails.txt"
17   fi
```

```
18
19    # Zeile aus Queue entfernen
20    without_first=$(cat "$queuefile" | tail -n +2)
21    echo "$without_first" > "$queuefile"
22
23    rmdir "$queuelock"
24    sleep 0.3
25  done
```

Listing II.31: ./mail-worker/worker.sh

```
1  <!doctype html>
2  <html lang="de">
3    <head>
4      <title>Monitoring</title>
5      <meta charset="utf-8">
6    </head>
7    <body>
8      <img src="./plot.png" style="max-width: 100%; max-height: 100%">
9    </body>
10 </html>
```

Listing II.32: ./monitoring/index.html

```
1  #!/bin/bash
2
3  monitoring_dir="/home/$USER/monitoring"
4
5  # Daten holen
6  time=$(date "+%Y-%m-%d-%H-%M-%S")
7  users=$(ls /usr/lib/cgi-bin/cookies/ | wc -l)
8  # CPU-Last aus uptime
9  # -> Zahl herausgreppen und cutten
10 # -> Komma/Punkt entfernen, da Bash damit später beim addieren nicht umgehen kann
11 # -> ggf. führende Null entfernen, weil Bash damit auch Probleme hat
12 cpu=$(uptime | grep -Eo 'load average: [0-9]+[,\.][0-9]+' | cut -d ' ' -f 3 | sed
   ↪  's/[,\.]//' | sed 's/^0*//')
13
```

```
14 rhodes_now=$(mariadb -e 'SELECT * FROM Positionsmeldung' | wc -l)
15 rhodes_last=$(cat "$monitoring_dir"/rhodes_last.txt)
16 rhodes=$(($rhodes_now-$rhodes_last))
17 echo "$rhodes_now" > "$monitoring_dir"/rhodes_last.txt
18
19 requests_now=$(cat /var/log/apache2/access.log | wc -l)
20 requests_last=$(cat "$monitoring_dir"/requests_last.txt)
21 requests=$(($requests_now-$requests_last))
22 echo "$requests_now" > "$monitoring_dir"/requests_last.txt
23
24 # Daten in Datei schreiben
25 echo "$time $users $cpu $rhodes $requests" >> "$monitoring_dir"/data.dat
```

Listing II.33: ./monitoring/measure.sh

```
1  set terminal pngcairo size 1920,1080 font 'Verdana,32'
2  set output '___OUTPUT___'
3  set title 'Monitoring der letzten 24 Stunden'
4
5  set xlabel 'Zeit'
6  set xdata time
7  set timefmt "%Y-%m-%d-%H-%M-%S"
8  set format x "%H:%M"
9  set xrange [""."system("date +%Y-%m-%d-%H-%M-%S -d '1 day ago'"):]
10
11 set yrange [0:]
12
13 plot '___INPUT___' using 1:($2/15) with linespoints linewidth 3 title 'Eingeloggte
   ↪  User', \
14      '___INPUT___' using 1:($3/100/15) with linespoints linewidth 3 title
        ↪  'CPU-Last', \
15      '___INPUT___' using 1:($4/15) with linespoints linewidth 3 title
        ↪  'Rhodes-Nachrichten', \
16      '___INPUT___' using 1:($5/15) with linespoints linewidth 3 title
        ↪  'HTTP-Requests'
```

Listing II.34: ./monitoring/plot.gp

```
#!/bin/bash

monitoring_dir="/home/$USER/monitoring"

# Daten der letzten fünfzehn Minuten addieren (Mittelwert wird in plot.gp errechnet)
rm -f "$monitoring_dir"/data-averaged.dat
interval=15
i=0
sum_users=0
sum_cpu=0
sum_rhodes=0
sum_requests=0

# Über data.dat iterieren
cat "$monitoring_dir"/data.dat | while read line; do
  sum_users=$(($sum_users+$(echo $line | cut -d ' ' -f 2)))
  sum_cpu=$(($sum_cpu+$(echo $line | cut -d ' ' -f 3)))
  sum_rhodes=$(($sum_rhodes+$(echo $line | cut -d ' ' -f 4)))
  sum_requests=$(($sum_requests+$(echo $line | cut -d ' ' -f 5)))

  # Bei erreichtem Intervall schreiben und zurücksetzen
  if test $(($i % $interval)) == 0; then
    echo $(echo $line | cut -d ' ' -f 1)" $sum_users $sum_cpu $sum_rhodes
    ↪  $sum_requests" >> "$monitoring_dir"/data-averaged.dat
    sum_users=0
    sum_cpu=0
    sum_rhodes=0
    sum_requests=0
  fi

  i=$(($i+1))
done

# Graph generieren
gnuplot "$monitoring_dir"/plot.gp 2>/dev/null
cp "$monitoring_dir"/plot.png "/var/www/html/$USER-web/monitoring/plot.png"
```

Listing II.35: ./monitoring/plot.sh

```
#!/bin/bash

# Erhalte Daten von Rhodes und aktualisiere die Datenbank

while read line; do

  # timestamp|...|mmsi|...|...|name|...|...|latitude|longitude|...
  timestamp=$(echo $line | cut -d '|' -f 1)
  mmsi=$(echo $line | cut -d '|' -f 3)
  name=$(echo $line | cut -d '|' -f 6)
  longitude=$(echo $line | cut -d '|' -f 9)
  latitude=$(echo $line | cut -d '|' -f 10)

  query="
  INSERT INTO Schiff (mmsi, name) VALUES
    ('$mmsi', '$name')
    ON DUPLICATE KEY UPDATE name='$name';

  INSERT INTO Positionsmeldung (time, mmsi, latitude, longitude) VALUES
    ('$timestamp', '$mmsi', '$latitude', '$longitude');
  "
  mariadb -e "$query" 2>/dev/null

done
```

Listing II.36: ./rhodes-watcher/reader.sh

```
#!/bin/bash

# Prüfe, ob seit dem letzten Ausführen neue Daten von Rhodes gespeichert wurden
# Wenn nein, starte reader.sh (neu)

ip="194.94.217.72"
port="8082"
rhodes_dir="/home/$USER/rhodes"

# SELECT count()... liefert zwei Zeilen (Spaltenname und Zahl)
now=$(mariadb -e 'SELECT count(id) FROM Positionsmeldung' | tail -n 1)

```

```
# Wenn Datei nicht existiert oder Inhalt gleich $now ist...
if ! test -f "$rhodes_dir/rhodes.last" || test "$(cat "$rhodes_dir/rhodes.last")"
↪  == "$now"; then
  kill $(cat "$rhodes_dir/rhodes.pid" 2>/dev/null) 2>/dev/null
  ncat -e "$rhodes_dir/reader.sh" "$ip" "$port" &
  echo "$!" > "$rhodes_dir/rhodes.pid"
fi

echo "$now" > "$rhodes_dir/rhodes.last"
```

Listing II.37: ./rhodes-watcher/watcher.sh

```
set terminal pngcairo size 1920,1080 font 'Verdana,32'
set output '___OUTPUT___'
set title '___TITLE___'

set xlabel 'Abschlusszeit'
unset xtics

set key left top

plot '___INPUT___' using 1:2 with linespoints linewidth 3 title 'Sekunden von Start
↪  bis Logout', \
     '___INPUT___' using 1:3 with linespoints linewidth 3 title 'Größe der Queue
     ↪  nach Login', \
     '___INPUT___' using 1:4 with linespoints linewidth 3 title 'Sekunden bis Mail
     ↪  erhalten wurde'
```

Listing II.38: ./testing/lasttest/plot-results.gp

```
#!/bin/bash

users="$1"
delay="$2"

rm -f "/home/$USER/lasttest/data.dat"
for i in $(seq $users); do
    started=$(cat /home/$USER/lasttest/started.dat | grep "user$i " | cut -d ' ' -f
    ↪  2)
```

```
    finished=$(cat /home/$USER/lasttest/finished.dat | grep "user$i " | cut -d ' '
    ↪  -f 2)
    timetaken=$(echo "$finished-$started" | bc)
    queuesize=$(cat /home/$USER/lasttest/queuesize.dat | grep "user$i " | cut -d '
    ↪  ' -f 2)
    mailtime=$(cat /home/$USER/sent-mails.txt | grep "user$i@" | cut -d ' ' -f 1)
    waituntilmail=$(echo "$mailtime-$started" | bc)

    echo "$finished $timetaken $queuesize $waituntilmail" >>
    ↪  "/home/$USER/lasttest/data.dat"
done
sort "/home/$USER/lasttest/data.dat" > /tmp/tmpsort
mv /tmp/tmpsort "/home/$USER/lasttest/data.dat"

cp testing/lasttest/plot-results.gp /tmp/tmp-plot.gp
sed -i 's/___INPUT___/\/home\/'$USER'\/lasttest\/data.dat/g' /tmp/tmp-plot.gp
sed -i 's/___OUTPUT___/\/home\/'$USER'\/lasttest\/lasttest.png/g' /tmp/tmp-plot.gp
sed -i 's/___TITLE___/Lasttest mit '"$users"' Nutzern und '"$delay"'s
↪  Verzögerung/g' /tmp/tmp-plot.gp
gnuplot /tmp/tmp-plot.gp
rm -f /tmp/tmp-plot.gp
cp /home/$USER/lasttest/lasttest.png
↪  "/var/www/html/$USER-web/monitoring/lasttest.png"
```

Listing II.39: ./testing/lasttest/plot-results.sh

```
#!/bin/bash

# run-lasttest.sh <Anzahl Nutzer> <Verzögerung zwischen Nutzern>
users="$1"
delay="$2"

# Nutzer erstellen
query="INSERT INTO User (email, password) VALUES"
for i in $(seq $users); do
    query="$query ('user$i@lasttest.de', 'demo')"
    if test "$i" != "$users"; then
        query="$query,"
    fi
done
```

```
mariadb -e "$query;" 2>/dev/null

# Log vorbereiten
rm -rf "/home/$USER/lasttest/"
mkdir "/home/$USER/lasttest"
touch "/home/$USER/lasttest/data.dat"
rm -f "/home/$USER/sent-mails.txt"
touch "/home/$USER/sent-mails.txt"

# Simulationen starten
for i in $(seq $users); do
    testing/lasttest/simulate-user.sh "$i" &
    echo "$i / $users started"
    sleep $delay
done

# Auf Ende warten
while test $(cat "/home/$USER/sent-mails.txt" | wc -l) != "$users"; do
    echo "Waiting... ($(cat "/home/$USER/sent-mails.txt" | wc -l) / $users done)"
    sleep 3
done
sleep 1
testing/lasttest/plot-results.sh "$users" "$delay"
```

Listing II.40: ./testing/lasttest/run-lasttest.sh

```
#!/bin/bash

userid="$1"

# Simuliere einen Nutzer-Login und Logout

echo "user$userid $EPOCHREALTIME" >> "/home/$USER/lasttest/started.dat"

cookiefile="/home/$USER/lasttest/cookie$userid.txt"
curl -b "$cookiefile" -c "$cookiefile" "localhost/$USER-web/cgi-bin/login.sh?email=user$userid@lasttest.de&password=demo" &>/dev/null

echo "user$userid $(($(cat "/home/$USER/mail-queue" | wc -l)-1))" >> "/home/$USER/lasttest/queuesize.dat"
```

```
13
14 curl -b "$cookiefile" -c "$cookiefile" "localhost/$USER-web/cgi-bin/logout.sh"
   ↪  &>/dev/null
15
16 echo "user$userid $EPOCHREALTIME" >> "/home/$USER/lasttest/finished.dat"
```

Listing II.41: ./testing/lasttest/simulate-user.sh

```
1  #!/bin/bash
2
3  bin/stop-all.sh
4  bin/deploy-db.sh
5  bin/deploy-cgi.sh
6  bin/deploy-mail.sh
7  rm -f /tmp/received-cookie.txt
8
9  # Bei validen Login-Daten, erwarte einen Cookie im Header und im Dateisystem
10
11 curl -s -c /tmp/received-cookie.txt
   ↪  "localhost/$USER-web/cgi-bin/login.sh?email=demo@user.de&password=demo"
   ↪  >/dev/null
12
13 if test -f /tmp/received-cookie.txt -a "$(grep -R /usr/lib/cgi-bin/cookies/
   ↪  "demo@user.de")"; then
14     exit 0
15 fi
16
17 echo "Cookie erwartet, aber im Header und/oder im Dateisystem nicht gesetzt"
18 exit 1
```

Listing II.42: ./testing/test-login-cookie.sh

```
1  #!/bin/bash
2
3  bin/stop-all.sh
4  bin/deploy-db.sh
5  bin/deploy-cgi.sh
6  bin/deploy-mail.sh
```

```

# Bei invaliden Login-Daten, erwarte eine leere Antwort

response=$(curl -s
↪  "localhost/$USER-web/cgi-bin/login.sh?email=wrong@user.de&password=wrong")

if test "$response" = ""; then
    exit 0
fi

echo "\"\" erwartet, aber \"$response\" erhalten"
exit 1
```

Listing II.43: ./testing/test-login-invalid.sh

```
#!/bin/bash

bin/stop-all.sh
bin/deploy-db.sh
bin/deploy-cgi.sh
bin/deploy-mail.sh

# Bei leerer Queue, erwarte eine versendete Mail eine Sekunde nach validem Login

curl -s "localhost/$USER-web/cgi-bin/login.sh?email=demo@user.de&password=demo"
↪  >/dev/null

sleep 1

if test "$(cat "/home/$USER/sent-mails.txt" | grep "demo@user.de")"; then
    exit 0
fi

echo "Mail erwartet, aber keine vorhanden"
exit 1
```

Listing II.44: ./testing/test-login-mail.sh

```
#!/bin/bash

bin/stop-all.sh
bin/deploy-db.sh
bin/deploy-cgi.sh
bin/deploy-mail.sh

# Bei validen Login-Daten, erwarte die E-Mail des Nutzers als Antwort

response=$(curl -s
↪  "localhost/$USER-web/cgi-bin/login.sh?email=demo@user.de&password=demo")

if test "$response" = "demo@user.de"; then
    exit 0
fi

echo "\"demo@user.de\" erwartet, aber \"$response\" erhalten"
exit 1
```

Listing II.45: ./testing/test-login-valid.sh

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 13. September 2024 Unterschrift: