Technische Universität Berlin
Fakultät Elektrotechnik und Informatik
Fachgebiet Regelungssysteme

Bachelorarbeit

Entwicklung einer verschlüsselten Chat-basierten Plattform auf Basis des Matrix.org-Protokolls im Kontext von digitaler Gesundheit und Telemedizin

von

Pascal Fischer
Matrikelnummer: 371778

Betreuer: Dr. Thomas Schauer, Ardit Dvorani

Erster Prüfer : Dr.-Ing Thomas Schauer
Zweiter Prüfer : Prof.-Dr.-Ing. Clemens Gühmann

Berlin, 3. November 2022

Zusammenfassung

In dieser Arbeit befasse ich mich mit ....

Abstract

This paper describes how to ...

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den _____ Pascal Fischer_____

Inhaltsverzeichnis

1 Einleitung

Mit vorranschreiten der Digitalisierung Im Zuge der Coronapandemie im Jahre 2019 wurden nicht nur die Rufe nach

2 Problem und Motivation

Ziel dieser Arbeit ist es eine Plattform zu entwickeln welchen es Ärzten und Patienten ermöglicht in Kontakt zu treten und Nachrichten auszutauschen. Hierfür soll eine Chat-App auf basis des Matrix.org-Protokolls implementiert werden. Die App ist in der Programmiersprache Swift zu entwickeln und soll auf allen gängigen iOS geräten unterstützt werden. Darüber hinaus muss die benötigte Infrastruktur bereitgestellt werden um Nachrichten und Daten zu speichern und zu übermitteln. Da medizinische Daten dem Datenschutz in besonderer Maße unterliegen ist es besonders wichtig, dass jegliche Art von Kommunikation zwischen Ärzten und Patienten stets verschlüsselt ist. Es muss gewährleistet sein, dass zu jedem Zeitpunkt alle personenbezogenen Daten vor dem Zugriff unerlaubter Dritter geschützt sind. Da es im Bereich der Medizin hinaus ist zu beachten, dass das Userinterface So sollte beispielsweise darauf geachtet werden, dass Da es im Bereich der Medizin hinaus ist zu beachten, dass das Userinterface So sollte beispielsweise darauf geachtet werden, dass

Theoretische Grundlagen 3

In diesem Kapitel werden zunächst die theoretischen Grundlagen, welche die Implementierung der Plattform beeinflusst haben, vorgestellt. Anschließend wird das Matrix Protokoll genauer betrachtet und das Grundprinzip erklärt. Desweiteren werden die in der End-to-End Verschlüsselung genutzten keys erläutert und genauer auf die verschiedenen Verschlüsselungsalgorithmen eingegangen.

3.1 Anwendungsarchitektur

Wie im Bereich von Webapplications haben sich auch im Bereich von Mobile Apps über die Zeit bestimmte Anwendungsarchitekturen (Design Patterns) entwickelt. Diese dienen als Muster um wiederkehrende Entwurfsprobleme zu lösen, indem man auf bewährte Praktiken zurückgreift. Sie beschreiben die allgemeine Struktur der Anwendung und geben somit auch die Aufteilung des Quellcodes vor. Die richtige Wahl der Anwendungsarchitektur kann einen entscheidenden Einfluss darauf nehmen wie effizient die Entwicklung des Projektes verläuft. Darüber hinaus erleichtern sie die Arbeit im Team da sie ein einheitliches Vokabular bieten. Im Folgenden werden 3 der bekanntesten Architekturen für Mobile Apps näher erläutert.

3.1.1 Model-View-Controller

Die Model-View-Controller Architektur teilt eine Anwendung in 3 Bereiche auf welche in Abbildung 3.1 dargestellt sind.

Das Model repräsentiert die Struktur der Daten und hält Informationen über

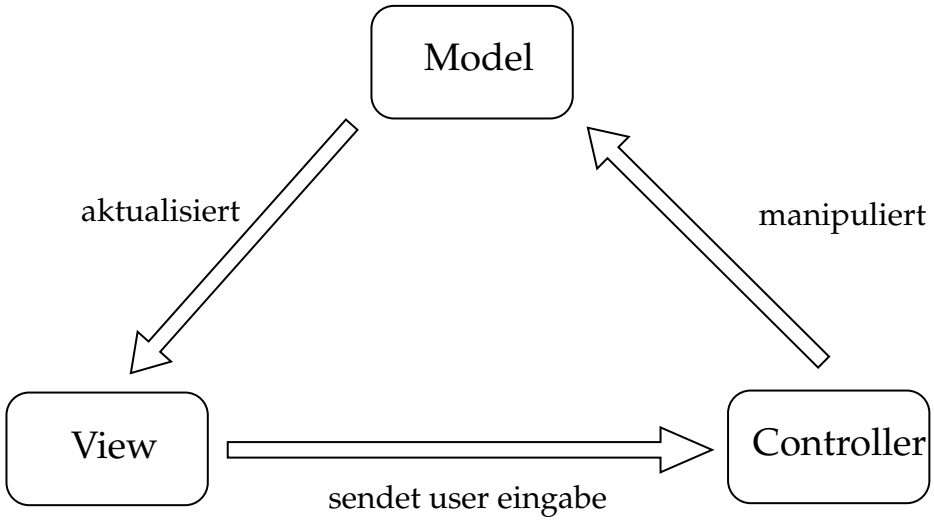

Abbildung 3.1: Interaktionen im Model-View-Controller

den aktuellen Zustand der Anwendung. Außerdem ist es verantwortlich für die Kommunikation mit externen Datenquellen wie beispielsweise Datenbanken. [3] Es liefert die Daten an die View weiter, ohne diese zu verändern. Die Daten im Model sind somit unabhängig von ihrer Präsentation. Deshalb ist es möglich mehrere Views an dasselbe Model, und somit dieselben Daten zu binden. Hierbei sind jedoch die Präsentationen der Daten auf den einzelnen Views ebenfalls unabhängig voneinander. Sollten sich die Daten im Model ändern werden die Views über diese Änderung informiert. [5]

Die View stellt die optische Oberfläche der Anwendung dar. Sie enthält alle grafischen Elemente wie Textfelder, Bilder oder anderer Elemente die zur Darstellung der Daten aus dem Model benötigt werden. Zudem nimmt sie die Eingaben des Users entgegen und leitet diese an den Controller weiter.

Der Controller ist für die Steuerung der Anwendung zuständig. Seine Aufgabe ist es je nach Eingabe des Benutzers das Model manipulieren. Basierend auf der Eingabe des Benutzers ist er ebenfalls dafür zuständig die aktuelle View zu wechseln.

Im Model-View-Controller Pattern sind Programmlogik, Datenstruktur und die Präsentation der Daten derart getrennt, dass es möglich ist diese Unabhängig voneinander zu verändern, ohne damit Einfluss auf einen der anderen Bereiche zu nehmen. Somit ist es möglich beispielsweise die Oberfläche eines Programmes zu überarbeiten, ohne sich dabei Gedanken über die darunter liegende Ausführungslogik zu machen.

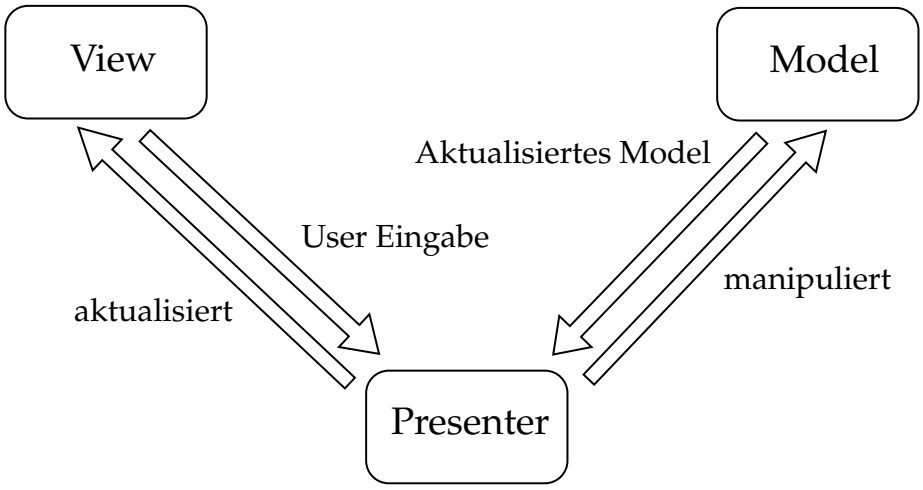

Abbildung 3.2: Interaktionen im Model-View-Presenter

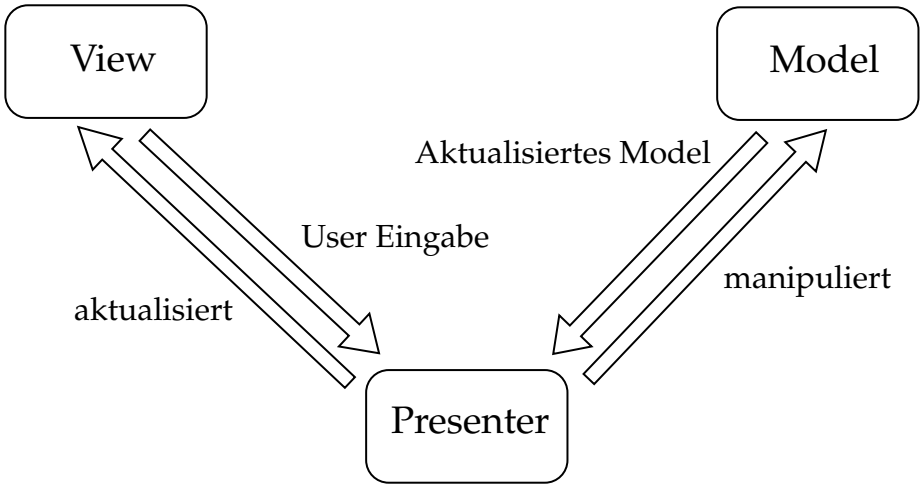

Abbildung 3.3: Interaktionen im Model-View-Presenter

3.1.2 Model-View-Presenter

Die Model-View-Presenter Architektur ist eine weiterentwicklung der Model-View-Controller Architektur.

Das Model

3.1.3 Model-View-ViewModel

Die Model-View-ViewModel Architektur ist ebenfalls eine weiterentwicklung der Model-View-Controller Architektur.

3.2 Matrix

3.2.1 Matrix Protokoll

Das Matrix Protokoll ist ein offener Standard für dezentrale Echtzeitkommunikation im Internet. Es ist ein Open-Source-Projekt welches unter Aufsicht der Matrix.org Foundation, einer Non-Profit-Organisation aus England, entwickelt wird. Das Team besteht aus 12 Personen mit weitreichenden Erfahrungen im Bereich VoIP und mit Chat-apps für Mobilgeräte. Die Entwicklung begann im Jahr 2014 und ist seit Juni 2019 für den Einsatz im Produktionsbetrieb geeignet. Mittlerweile gibt es zahlreiche Beiträge aus der Community, wodurch das Projekt weiter vorangetrieben wird. [2]

Das ursprüngliche Ziel des Projektes war es ein Protokoll zu entwickeln, welches es Usern ermöglichen soll, anderen Usern Nachrichten zu schreiben oder diese anzurufen, unabhängig davon welche Platform diese benutzen. Langfristig soll Matrix ein generisches System zur Messaging und Datensynchronisation über HTTP für das ganze Internet bilden. [2]

Hierzu definiert Matrix eine Reihe von APIs:

Client-Server-API[1] zur Kommunikation zwischen Matrix kompatiblen Clients und einem Homeserver

Server-Server-API[2] zum Nachrichtenaustausch und Synchronisation zwischen mehreren Homservern

Application-Service-API[3] zur Erweiterung der Funktionalität von Matrix und Integration anderer Systeme

Identity-Service-API[4] beschreibt das Mapping zischen Drittanbietern

Push-Gateway-API[5] um versenden von Push Notifications an Clients falls neue Events beim Homeserver eintreffen

[1] https://spec.matrix.org/v1.4/client-server-api/
[2] https://spec.matrix.org/v1.4/server-server-api/
[3] https://spec.matrix.org/v1.4/application-service-api/
[4] https://spec.matrix.org/v1.4/identity-service-api/
[5] https://spec.matrix.org/v1.4/push-gateway-api/

3.2.2 Funktionsweise

3.2.3 Events

Im Matrix Protokoll werden alle Ereignisse (Events) in einer einzigen großen Kette (Timeline) gespeichert. Diese Events können in zwei Kategorien unterteilt werden. Zum einen gibt es State Events, welche Ereginsse wie die Erzeugung eines Raumes oder das Beitreten eines Users zu einem Raum beschreiben. Zum anderen gibt es Message Events welche realen Datenaustausch zwischen Usern beschreiben. Dies können beispielsweise Textnachrichten, Fotos oder Videos sein. Jedes Event enthält unter anderem folgende Attribute:

event_id ein eindeutiger Identifier für das jeweilige Event

sender die Matrix ID des Absenders des Events

room_id ein eindeutiger Identifier über welchen ein Event einem bestimmten Raum zugeordnet werden kann

type der Typ des Events

content der tatsächliche Inhalt der Nachricht. Dieser variiert abhängig vom event Type

origin_server_ts Zeitstempel

3.2.4 Rooms

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

3.2.5 Devices

3.2.6 Verwendete Schlüssel

text

3.3 Verschlüsselung

3.3.1 Ed25519

text

3.3.2 Curve25519

text

3.3.3 AES-256

text

3.3.4 HMAC-SHA-256

text

3.4 REST

4 Verwendete Technologien

4.1 XCode

4.2 Simulator

4.3 Docker

Docker ist eine Containervirtualisierungssoftware welche es ermöglicht Anwendung vom Rest des Systems zu isolieren. Auch wenn sie nach außen wie eine virtuelle Maschine wirkt, bietet sie entscheidende Vorteile. Herkömmliche virtuelle Maschinen bilden eine vollständige Kopie eines Betriebssystems welches mittels einer Virtualisierungssoftware wie beispielsweise Oracle VirtualBox oder KVM auf einem Hypervisor betrieben werden muss. Betrachtet man den vollständigen Stack einmal von unten hat man zuallererst die Hardware des Hypervisors, auf diesem läuft ein Betriebssystem, worauf wiederum die Virtualisierungssoftware läuft, mit welcher eine isolierte Kopie eines anderen Betriebssystems erzeugt wird in welcher die eigentliche Anwendung dann läuft. Dies sorgt für erhebliche Einbußen in der Performance.

Docker Container sind in der Lage in wenigen Sekunden zu starten. Sie laufen direkt auf dem Betriebssystem des Hypervisors sind aber dennoch isoliert vom Rest des Systems. Dies geschieht mithilfe zweier Linux Kernel Technologien, namespaces und cgroups.

[4]

4.4 Synapse

4.5 Swift

4.6 Matrix SDK

Konzept 5

In diesem Kapitel werden 2 verwandte Projekte analysiert und unter Betrachtung der Zielgruppe einige Anforderungen definiert die die App zu erfüllen hat.

5.1 Analyse ähnlicher Projekte

5.2 Anforderungen

Nach der Analyse ähnlicher Projekte wurden eine Liste funktionaler Anforderungen erstellt. Diese wurden nach der MoSCoW-Methode priorisiert.

Must:

i. Der User muss über die App einen Account auf der Plattform anlegen können.

ii. Der User muss sich mit seinem Account in der App einloggen können.

iii. Der User muss sein Passwort zu ändern können.

iv. Dem User muss alle von beigetretenen Räume sehen können.

v. Dem User muss neue Räume erstellen können.

vi. Der User muss zu neuen Räumen eingeladen werden können.

vii. Der User muss alle Nachrichten die in einem Raum gesendet wurden einzusehen können.

viii. Der User muss Nachrichten in einem Raum zu senden können.

ix. Nachrichten zwischen Usern müssen End-to-End Verschlüsselt sein.

x. Der User muss einen Raum verlassen können.

xi. Der User muss seinen Account deaktivieren können.

Should:

i. Beim erstellen des Accounts soll eine zusätzliche Authentifizierungsmethode verwendet werden um Wahloses erstellen zon Accounts zu verhindern.

ii. Der User soll sich nur einmal einloggen müssen.

iii. Der User soll sich ausloggen können.

iv. Der Benutzer soll die Möglichkeit haben sein Profilbild und seinen Anzeigenamen anzupassen.

v. Der vollständige Chat-Verlauf soll nur bei Bedarf geladen wer-

den.

vi. Die App soll neben Textnachrichten auch andere Nachrichtentypen wie Fotos oder Videos unterstützen.

vii. Die App soll den User über den Erhalt einer neuen Nachricht informieren.

viii. Die Übersicht der beigetretenen Räume soll nach letzter Aktivität sortiert werden.

ix. Die App soll Räume und Chat-Verläufe local speichern und offline wiedergeben können.

Could:

i. Die Liste der beigetretenen Räume kann gefiltert werden.

ii. Dem Erstellen eines Raumes kann dem User eine Liste von Usern vorgeschlagen werden, welche dem gesuchten Namen entsprechen.

iii. Dem User kann ein Typing-Indikator gezeigt werden.

iv. Die App kann auch mit anderen Homeservern verbunden werden.

v. Der User kann den Inhalt einer Textnachricht in die Zwischenablage kopieren.

vi. Der User kann eine Nachricht weiterleiten.

vii. Der User kann Dateien welche in einem Raum verschickt wurden herunterladen.

5.3 Auswahl der Systemarchitektur

6 Implementierung

Dieses Kapitel beschreibt die 3

6.1 User Interface

Der erste Schritt in der Entwicklung der App ist das Design des User Interfaces. Hierbei sollte auf Dieses besteht aus mehreren einzelner Views welche welche sich zu verschiedenene Gruppen zusammenführen lassen.

6.2 Homeserver

Nachdem die App das erste mal gestartet wird

6.3 Account Erzeugungs Flow

Um die App nutzen zu können muss zualler erst einmal ein Account erstellt werden können. Hierzu können aufseiten des Homeservers verschiedene Authentifizierungsstufen definiert werden, welche vom Client durchlaufen werden müssen, um einen Account zu erstellen. Diese können beispielseise In Abbildung 6.1 ist ein dreistufiger Accounterzugungsvorgang mittels eines Registriersungstokes beschrieben. Dieser registrierungstoken kann entweder in der Datenbank direkt aangelegt werden oder mittels eines Administartor users auf der admin API erstellt werden. Dieser token muss dem User anschließend zur Verfügung gestellt werden beispielsweise via email. Der eigentliche Autehntifizierungsprozess beginnt dann wie folgt. Im ersten Schritt sendet der Client eine leere Anfrage zum Server in welcher er um das starten einer registrierungs session bittet. Der Server antortet daraufhin mit einer SessionID und den vom Homeserver unterstützten Authentifizierungs Flows. In diesem Beispiel ist der einzige unterstützte Flow die verifizierung über einen Registriersungstoken. Im nächsten Schritt sendet der Client dann den zuvor erstellten registrierungstoken und die vom server bereitgestellte sessionid in Kombination mit der aktuellen Stufe des Authentifizierungs prozessen. Im dritten und letzten Schritt wird nun der dummyflow aufgeführt. Dieser Flow kann nicht fehlschlagen und dient der Letzendlichen erstellung des Account. Hierbei werden der gewünsche username und passwort übergeben. Ist dies gesehen wurde der Account erfolgreich erstellt.

6.4 Raum erstellung

User directory

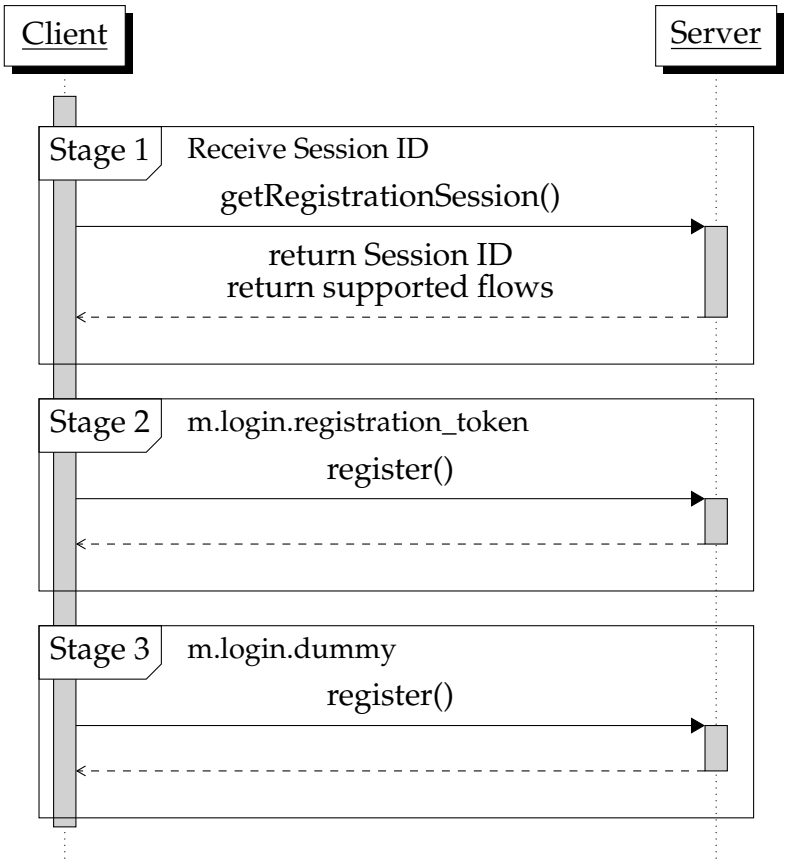

Abbildung 6.1: Account erzeugungs Flow

6.4.1 Text

Der wohl wichtigste Nachrichtentyp in

6.4.2 Foto

text

6.4.3 Video

text

6.4.4 Datei

text

6.5 Account Deactivation Flow

text

6.6 Nachrichten

some text

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

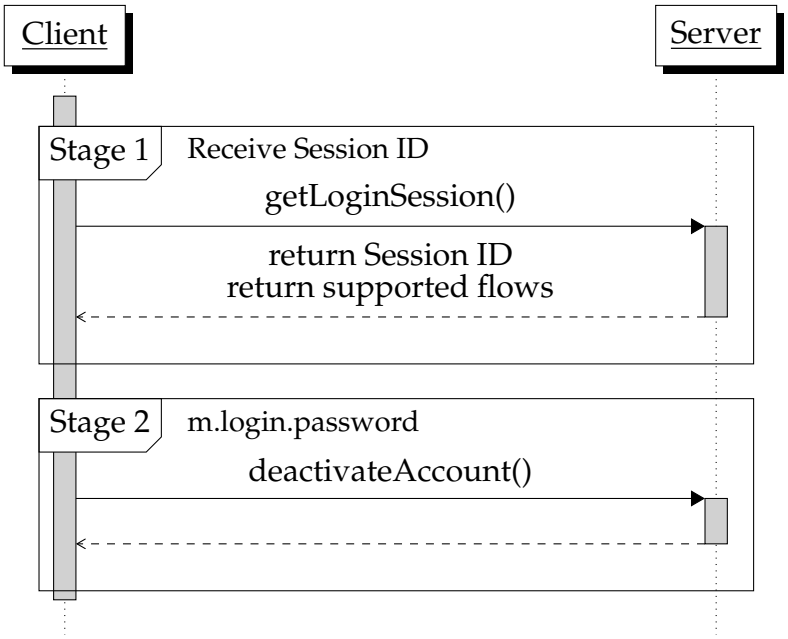

Abbildung 6.2: Account löschen Flow

6.7 Benachrichtigungen

7 Evaluation

8 Fazit

text

9 Ausblick

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] An open network for secure, decentralized communication. https://matrix.org/, . Accessed: 2022-10-01.

[2] Frequently asked questions. https://matrix.org/faq/, . Accessed: 2022-10-01.

[3] Model-view-controller-paradigma. https://glossar.hs-augsburg.de/Model-View-Controller-Paradigma. Accessed: 2022-10-01.

[4] C. Anderson. Docker [software engineering]. *IEEE Software*, 32(3):102–c3, 2015. doi: 10.1109/MS.2015.62.

[5] M. R. J. Qureshi and F. Sabir. A comparison of model view controller and model view presenter. 2014. doi: 10.48550/ARXIV.1408.5786. URL `https://arxiv.org/abs/1408.5786`.