HTBLuVA St. Pölten
Höhere Lehranstalt für Elektrotechnik

DIPLOMARBEIT

Binobo - Mechanischer Controller zur Erfassung aller Rotationspunkte einer menschlichen Hand

Ausgeführt im Schuljahr 2021/22 von:

Dominik Lovetinsky
Philipp Maschayechi

Betreuer/Betreuerin:

Dipl.-Ing. Ronald Spilka
Dipl.-Ing. Werner Damböck

St. Pölten, am 30. März 2022

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Dominik Lovetinsky

Philipp Maschayechi

Diplomandenvorstellung

Dominik Lovetinsky

Geburtsdaten:
27.07.1999 in Krems

Wohnhaft in:
Liliengasse 2
3390 Melk

Beruflicher Werdegang:
2016–2022:
HTBLuVA St.Pölten, Abteilung für Elektrotechnik
7/2021–8/2021:
Praktikum bei IMS Nanofabrications
8/2021–9/2021:
Praktikum bei Microtronics
2005–2014:
Grundausbildung in Volk-, Hauptschule und Polytechnikum

Kontakt:
dominik.lovetinsky99@gmail.com

Philipp Maschayechi

Geburtsdaten:
27.9.2003 in Wien

Wohnhaft in:
Erikenstrasse 12
3032 Eichgraben

Beruflicher Werdegang:
2009–2017:
Grundausbildung in Volksschule und Gymnasium
2017–2022:
HTBLuVA St.Pölten, Abteilung für Elektrotechnik
7/2019–8/2019:
Praktikum bei Bergbahnen Kössen
7/2020–8/2020:
Praktikum bei CareTec International GmbH

Kontakt:
Philipp.mas2003@gmail.com

Danksagungen

Die Diplomanden bedanken sich herzlichst für die Professionalität des Fotografen Tobias Sturmlechner, sowie für die zielgerichtete und angenehme Kooperation mit den Professoren der höheren technischen Bundeslehr- und Versuchsanstalt St.Pölten, Professor Ronald Spilka, Professor Werner Damböck und Fachlehrer Hermann Meiseneder. Weiters wird für die entsprechende Beaufschlagung der elektronischen Komponenten seitens des Laboranten Andreas Bergen gedankt.

Abstract

In recent years, developments such as robotic prosthetics, personal 3D-animation software and virtual reality have created demand for an affordable motion capture system. Since professional equipment is sold at professional prices, Binobo aims to offer a cheap alternative by using commonly available hardware in conjunction with open-source software on order to provide a way to capture all degrees of freedom the human hand has to offer. Our project is based on previous homemade mechanical arm control systems and VR-controllers created by people all around the world, as well as rigorous iterative testing. This project proves that it is possible to develop an open-source alternative to expensive mocap systems on the small, yet complicated scale a hand requires.

Zusammenfassung

In den letzten Jahren haben Entwicklungen wie Roboterprothesen, privat nutzbare 3D-Animationssoftware und Virtual Reality Technologie die Nachfrage nach einem erschwinglichen Motion-Capture-System geschaffen. Da professionelle Ausrüstung zu professionellen Preisen verkauft wird, zielt Binobo darauf ab, eine billige Alternative anzubieten, indem allgemein verfügbare Hardware in Verbindung mit Open-Source-Software verwendet wird, um eine Möglichkeit zu bieten, alle Freiheitsgrade zu erfassen, die die menschliche Hand zu bieten hat. Unser Projekt basiert auf früheren Hobbyprojekten von mechanischen Armsteuerungssystemen und VR-Controllern, die von Menschen auf der ganzen Welt entwickelt wurden, sowie auf strengen iterativen Tests. Dieses Projekt beweist, dass es möglich ist, eine Open-Source-Alternative zu teuren Mocap-Systemen auf dem kleinen, komplizierten Maßstab zu entwickeln, den eine Hand benötigt.

Inhaltsverzeichnis

1. Ausgangslage und Vision

Aufgrund beidseitigem Interesse an Robotik, wurde ursprünglich in Diskussion gebracht, eine bionische Roboterhand zu entwerfen, welche über einen Controller ansteuerbar sein soll. Aus diesem Projekt hervorgehen erhoffte man sich, Wissen in den Gebieten 3D-Druck, Elektronik, Serverprogrammierung und Kommunikationsprotokollen vertiefen zu können. Um möglichst viel Modularität in dieses Projekt einzubringen, soll der Controller auch isoliert für 3D Applikationen verwendbar sein. Da allerdings schnell klar war, dass ein derartiges Projekt den Rahmen einer Diplomarbeit sprengen würde, wurde entschieden, nur den Controller-Handschuh zu konstruieren und die mechanische Hand als optionales Ziel bzw. späteres Projekt nach der Diplomarbeit zu betrachten.

Dieser Handschuh soll mithilfe von 22 Potentiometern, also verstellbaren Widerständen, alle Freiheitsgrade der menschlichen Hand, insbesondere die Rotationswerte der Finger aufnehmen und an einen Computer senden, um die gemessenen Werte anhand eines Modells darzustellen.

2. Projektübersicht

Die Diplomarbeit umfasst einen Controllerhandschuh und eine Website, die von einem Server gehostet wird.
Es gibt zwei voneinander getrennte Server: Webserver und einen Websocketserver. Der Webserver wurde mithile des Java-Frameworks `Spring Boot` entwickelt. Dieser bieten die Möglichkeit sich als Client zu registrieren und einen Blog und einen Emulator zu nutzen. Der Emulator bietet die Möglichkeit zur echtzeitnahen Emulation der erfassten Rotationswerte auf einem 3D-Modell der menschlichen Hand.
Die Echtzeitemulation wird durch einen Websocketserver möglich, welcher Datenaustausch mit minimalen Latenzen ermöglicht.
Der Handschuh besteht aus einem konventionellen Nylonhandschuh, der mit 3D-gedruckten Komponenten beaufschlagt wird, sodass die Messelektronik, bestehend aus Potentiometern, Multiplexern und einem ESP32-Mikrocontroller, fixiert werden kann. Die Herausforderung hierbei bestand hauptsächlich darin, passende Elemente zu konstruieren, um möglichst genaue Messungen durchzuführen.

2.1. Aufgabeneinteilung

In Tabelle 2.1 ist die prinzipielle Arbeitsteilung zu sehen. Diese erfolgte in einer Weise, in der man sich erhoffte, dass man in den jeweiligen Bereichen persönliche Fortschritte macht. Für das jeweilige Individuum war es wichtig in in Software-Engineering, Firmwareentwicklung, Hardwareentwicklung und `three.js` empirische und professionelle Erfahrungen zu sammeln.

Dominik Lovetinsky	Software + Firmware
Philipp Maschayechi	Hardware + 3D-Emulator

Tabelle 2.1.: Arbeitsteilung

3. Hardwareaufbau

Um die Freiheitsgrade der menschlichen Hand aufzunehmen, ist Hardware in Form eines Controller-Handschuhes erforderlich. Dieser besteht aus Mess- und Kommunikationselektronik, sowie einer 3D-gedruckten Struktur, welche der Befestigung und mechanischen Artikulierung der Elektronik dient.

3.1. Elektronik

3.1.1. ESP32

Der ESP32 (Abb. 3.1) ist ein Microcontroller der Firma `Espressif`. Dieser μC wurde gewählt, da dieser die entsprechende Leistung besitzt, um in einer angemessenen Geschwindigkeit die Potentiometer lesen zu können. Zusätzlich besitzt dieser einen WIFI-Chip, wodurch es möglich ist, die Daten in einer sinnvollen Weise zu übertragen.

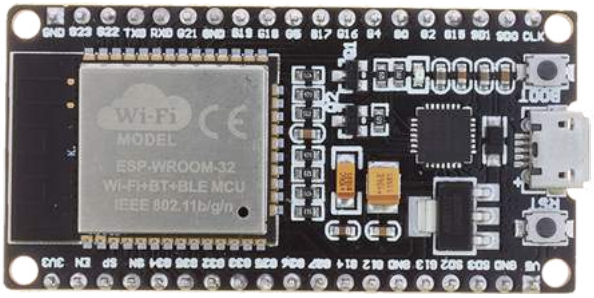


Abbildung 3.1.: Footprint: ESP32

Weiters besitzt dieser Controller ein natives USB-to-UART Interface, wodurch man über z.B. ein Smartphone kommunizieren kann.

3.1.2. Drähte

Weil die Komponenten des Controllers, und damit auch die Verbindungen dazwischen, ständig in Bewegung sind, ist es wichtig Drähte zu wählen, welche die wiederholte Verformung aushalten und keine Behinderung der Bewegungsfreiheit darstellen. Aufgrund der vorhandenen Komponenten im Lager der Schule, fiel die Wahl

auf *Flexivolt 0.25mm*-Drähte. Diese sind hochflexibel und somit perfekt für eine solche Anwendung geeignet. Obwohl der Durchmesser nur 0.25mm beträgt, fließen bei den genutzten hohen Widerständen, die in Kapitel 3.1.3 erklärt werden, keine hohen Ströme

3.1.3. Potentiometer

Um die Rotation an den Gelenken zu messen, werden *Iskra PNZ11Z*-1$M\Omega$- Drehpotentiometer genutzt. Die Wahl fiel auf variable Widerstände, da Hall-Rotationssensoren zu teuer und aufwändig für ein solches Projekt sind. Dehnmessstreifen wurden ebenfalls in Betracht gezogen, doch ist deren Messgenauigkeit stark von der Dehnbarkeit des Handschuhes abhängig, da dieser unter Umständen deutlich mehr nachgibt als der DMS.
Somit werden simple Potentiometer genutzt, aufgrund ihrer Kostengünstigkeit, Verfügbarkeit und Einfachheit. Verstellbare Widerstände der Type *PNZ11Z* bieten den Vorteil, sehr kompakt zu sein und mit einem hohen Widerstand den Stromverbrauch des Controllers gering zu halten.

Abbildung 3.2.: 3D-Footprint Potentiometer

Abbildung 3.3.: reale Potentiometer

3.1.4. Multiplexer

Da der ESP32 nur 16 Analogpins aufweist, allerdings, je nach Ausführung, an 20 bis 22 Potentiometer die Spannung gemessen werden muss, kommt es zu einem Engpass an Inputs. Dieses Problem wird durch die Nutzung von Multiplexern behoben. Multiplexer schalten basierend auf einem binären Input einen Pin auf verschiedene Anschlüsse. Somit können in diesem Fall 4 Digitalpins und ein Analogpin als 16 Analogpins genutzt werden. Es werden zwei Multiplexer des Typs **CD74HCT4067** nach folgender Aufteilung genutzt:

Multiplexer	Finger
1	Daumen Zeigefinger Mittelfinger
2	Ringfinger Kleiner Finger

Tabelle 3.1.: Multiplexereinteilung

Weiters wird das Handgelenk auf den äußeren Multiplexer, Multiplexer 2 gelegt. Obwohl ein einzelner Chip ausreichend wäre, um genügend Analogpins bereitzustellen, werden zwei genutzt, da dadurch der Aufbau einheitlicher und die Menge an notwendigen und flexiblen Leitungen verringert wird, weil der ESP32 etwas zu groß für die Montage am Handrücken ist und daher an der Handyhalterung am Arm installiert ist.

Abbildung 3.4.: 3D-Footprint Multiplexer

Abbildung 3.5.: realer Multiplexer

3.2. Eigens entwickelte Komponenten

Um die in Kapitel 3.1 detaillierten Komponenten auf dem Handschuh zu montieren, sind 3D-gedruckte Bauteile notwendig.

3.2.1. Epoxidharzdruck

Um hohe Genauigkeit, auch auf kleinem Maßstab bei den gedruckten Elementen zu gewährleisten, wurde 3D-Druck nach der SLA-Methode dem bekannteren FDM-Drucksystem bevorzugt.

Beim SLA-Druck wird photosensitives Epoxidharz als Material genutzt, welches mittels eines hochauflösenden UV-Displays in dünnen Schichten gehärtet wird. Dies bietet einige Vorteile, unter anderem höhere Genauigkeit und Qualität der gedruckten Modelle, sowie bedingt verringerte Druckzeit, da diese nur von der Höhe des Modells

abhängig ist. Diese Eigenschaften überwiegen unter den gegebenen Anforderungen den Nachteilen, wie das spröde Material und die giftigen Materialien.
Sämtliche Modelle wurden mit einem *Anycubic Photon* SLA-Drucker ausgedruckt. Da allerdings nicht der Drucker sondern die Druckmethode hier entscheidend ist, ist es möglich, die Komponenten mit jedem vergleichbaren SLA-Drucker anzufertigen.
Das genutzte Epoxidharz war ebenfalls von der Marke *Anycubic*, aufgrund der Möglichkeit von Hautkontakt mit den gedruckten Elementen wurde ein sojaölbasierendes Harz gewählt, welches für geringere Geruchsbelastung und angenehmeres Handling der Komponenten sorgt als das konventionelle, kunststoffbasierte Harz.
Eine interessante Eigenschaft des schwarz gefärbten Stoffes ist, dass sich über längere Zeiträume der Farbstoff absetzt, was zu leicht transparenten Modellen führt. Weshalb in den Fotos in Kapitel 3.2.3 manche Teile heller und transluzenter sind als andere. Dies hat keinen Einfluss auf die Qualität des Druckes.

3.2.2. Blender

Um die, im folgenden Kapitel 3.2.3 detaillierten, Modelle zu entwickeln, wurde das Open-Source 3d-Modellierungs- und Animationsprogramm *Blender*[53] einer technischeren Alternative, wie Beispielsweise *Autodesk Inventor*[54], oder *Autodesk Fusion 360*[55] vorgezogen, obwohl diese genauso geeignet für diesen Zweck wären. Der Grund für diese Entscheidung lässt sich zum Teil auf bestehende Erfahrung mit dem Programm zurückführen, doch weitere wichtige Faktoren waren unter anderem:

- Geringer Aufwand zur Kreation neuer Modelle
- 3D-Druck erfordert keine technischen Baupläne zur Produktion von Teilen
- Einfache punkt-, kanten- und flächenweise Manipulation von Modellen
- Erstellung hochwertiger Grafiken zur Dokumentation

Weiters ist *Blender* im Gegensatz zu *Inventor* kostenlos und kann somit auch nach Auslauf der Schullizenz genutzt werden. Die Downloadgröße und erforderlichen Systemressourcen von *Inventor* dienten auch als Grund für die Entscheidung.
Die einzigen Nachteile von *Blender* zeigen sich in der Organisation der Dateien, obwohl das vielmehr in den Verantwortungsbereich des Nutzers fällt.

3.2.3. Modelle

Abbildung 3.6.: Seitenansicht Standard-fingermodul

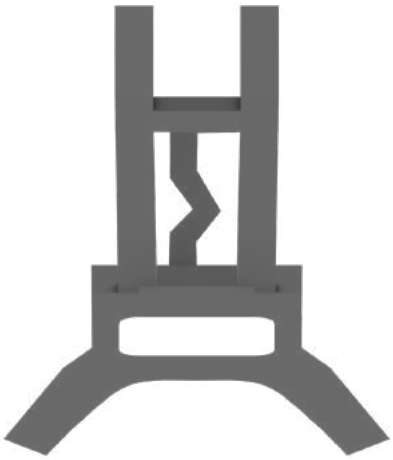


Abbildung 3.7.: Frontansicht Standardfingermodul

Name	Standardfingermodul
Datei	`controller_poti_holder`
LxBxH	19.9mm x 24.2mm x 28.3mm
Beschr.	Dieses Element erfüllt die Rolle, ein Potentiometer auf den Fingerrücken zu fixieren. Das Modul ist für die meisten Finger verwendbar, da sie eine ähnliche Größe aufweisen- Die Aussparung in der Mitte dient als Kanal für die Drähte, welche die Gelenke mit der Messelektronik verbinden

Tabelle 3.2.: Details Standardfingermodul

In Abbildung 3.6 und 3.7 werden verschiedene Ansichten des Moduls gezeigt. Da die Finger der menschlichen Hand verschiedene Breiten aufweisen, ist es erforderlich angepasste Elemente (Abbildung 3.8 und 3.9) für Daumen und Kleinen Finger zu drucken. Diese sind dem Grundmodul ähnlich, doch der Winkel und die Breite der Unterseite sind an die entsprechenden Finger angepasst.

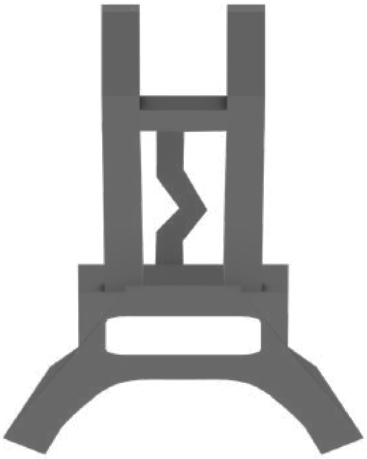


Abbildung 3.8.: Modul Kleiner Finger

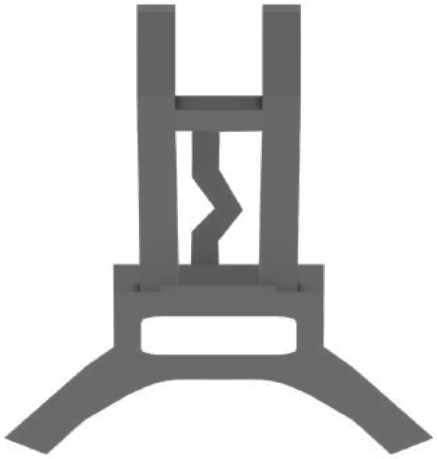
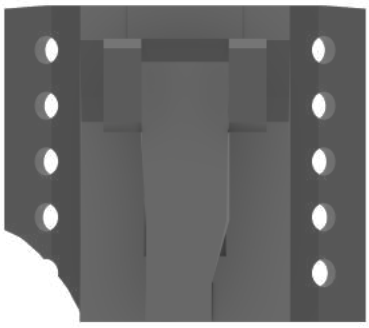


Abbildung 3.9.: Modul Daumen

Aufgrund des Aufbaus des Handschuhes ist es erforderlich, bei der untersten Haltevorrichtung des Kleinen Fingers einen Ausschnitt an der Innenseite einzuplanen, da ansonsten die Bewegungsfreiheit und der Tragekomfort reduziert wird.



In Abbildung 3.10 ist der Ausschnitt zu sehen, welcher dazu dient, Reibung und Verhaken von Handschuh und Konstruktion zu vermeiden. Die durch das fehlende Loch verlorene Stabilität ist hierbei in Kauf zu nehmen.

Abbildung 3.10.: Standardmodul Kleiner Finger mit Ausschnitt

Um zusätzliche Stabilität an den Fingerspitzen zu gewährleisten, wird bei den entsprechenden Modulen (Abbildung 3.11 und 3.12) eine Art Kappe vorgesehen, die die Fingerspitze abdeckt. Weiters ist an diesen Elementen keine Aufhängung für weitere Gelenke notwendig.

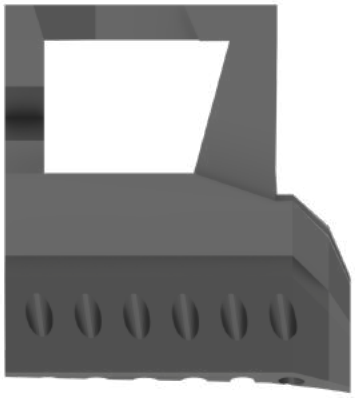


Abbildung 3.11.: Seitenansicht Fingerspitze

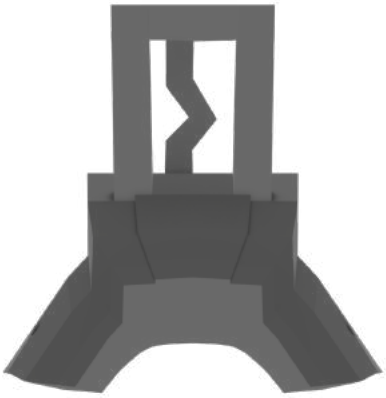


Abbildung 3.12.: Frontansicht Fingerspitze

Name	Fingerspitze
Datei	`controller_poti_fingertip`
LxBxH	21.8mm x 23.9mm x 24.4mm
Beschr.	Das Modul erfüllt die Rolle, ein Potentiometer auf der Fingerspitze zu fixieren. Die Abdeckung an der Spitze dient der Stabilität gegen längsseitiges Kippen und erhöht die Auflagefläche

Tabelle 3.3.: Details Fingerspitze

Dieses Element dient ebenfalls dazu, den beim Annähen der Spitze überschüssigen Stoff des Handschuhes zu binden, den Handschuh zu straffen, was der Stabilisierung der hinteren Fingerelemente dient, da dadurch verhindert wird, dass sie sich von der Haut aufgrund der Flexibilität des Stoffes abheben und somit Messergebnisse verfälschen. Selbstverständlich sind auch hier alternative Modelle für die äußersten Finger erforderlich (Abbildung 3.13 und 3.14). Hier wurde nicht nur die Breite der Auflagefläche modifiziert, sondern auch die Kappe an der Spitze.

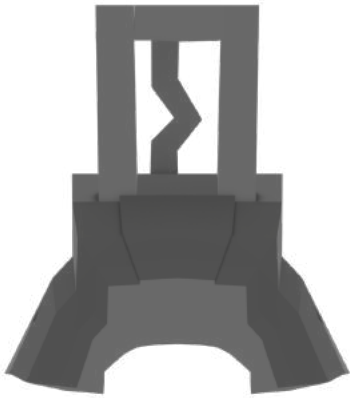


Abbildung 3.13.: Spitze des Kleinen Fingers

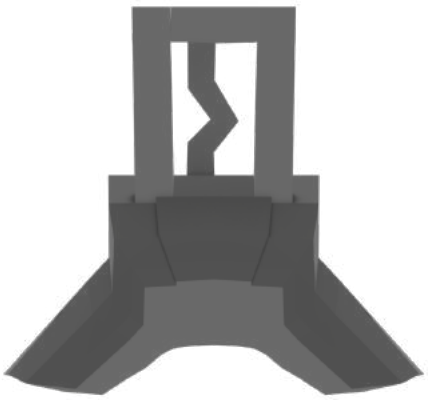


Abbildung 3.14.: Daumenspitze

Für den Daumen war aufgrund der Komplexität des unteren Gelenks eine Spezialanfertigung notwendig. Diese Halterung muss den Hebel zur Platte am Handrücken halten, doch gleichzeitig auch in einer anderen Achse ein Potentiometer fixieren, um die Greifbewegung des Daumens korrekt aufzunehmen. Die Herausforderung wurde bewältigt, indem das in den folgenden Abbildungen 3.15 und 3.16, sowie in Tabelle 3.4 beschriebene Modul durch iteratives Testen entwickelt wurde:

Abbildung 3.15.: Seitenansicht Daumenbasisgelenk

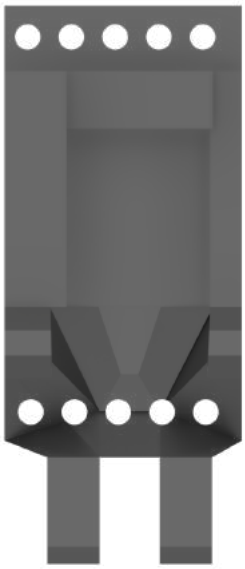


Abbildung 3.16.: Obersicht Daumenbasisgelenk

Name	Daumenbasisgelenk
Datei	`controller_poti_thumb_base`
LxBxH	14.9mm x 35.1mm x 12.5mm
Beschr.	Dieses Element dient dazu, als Übergang von der Platte zu den Daumengelenken zu agieren. Es ist über ein direktes Gelenk, sowie über einen Hebel mit der Basis-Platte verbunden.

Tabelle 3.4.: Details Daumenbasisgelenk

Um einen Bezugspunkt für die Gelenke zu haben und eine Montagemöglichkeit für die Multiplexer-Boards zur Verfügung zu stellen werden drei Platten am Handrücken befestigt. Aufgrund einer Kombination aus Praktikabilität beim 3D-Druck und Tragekomfort, kann hierfür nicht eine einzelne, große Fläche genutzt werden. Das innere Modul ist auf den folgenden Abbildungen 3.17 bis 3.19 dargestellt

Abbildung 3.17.: Seitenansicht innere Platte

Abbildung 3.18.: Frontansicht innere Platte

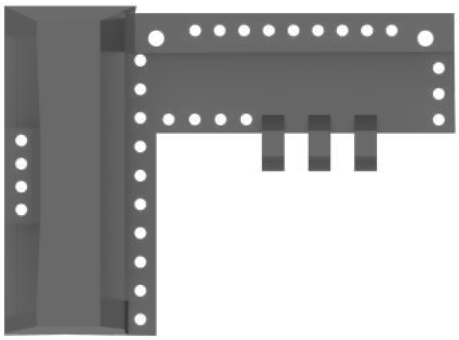


Abbildung 3.19.: Obersicht innere Platte

Name	innere Handrückenplatte
Datei	`controller_plate_inner`
LxBxH	60.1mm x 43.9mm x 12.6mm
Beschr.	Diese Platte erfüllt die Rolle, als Basis für Messungen, sowie einen Montagepunkt für die Multiplexer zu bieten. Das Gelenk auf der Innenseite dient als Befestigung für das in Tabelle 3.4 beschriebene Daumengelenk.

Tabelle 3.5.: Details innere Platte

Um die Hautverformung bei der Beugung des Daumengelenks nicht in die Messung einfließen zu lassen, ist die Platte zweigeteilt, sodass nur der relevante Teil des Winkels aufgenommen wird. Der Teil, auf dem das Daumengelenk sitzt wird in Abbildung 3.20 und 3.21 dargestellt. Hierbei ist es wichtig, dass die beiden Platten fest am Handschuh befestigt sind, da kleine Bewegungen der Platten große Messfehler verursachen können. Das vollständige Gelenk ist in Abbildung 3.22 abgebildet.

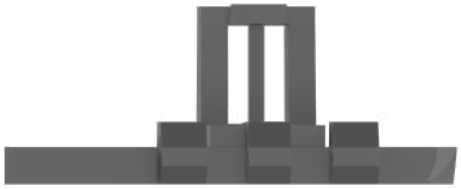


Abbildung 3.20.: Frontansicht Daumenplatte

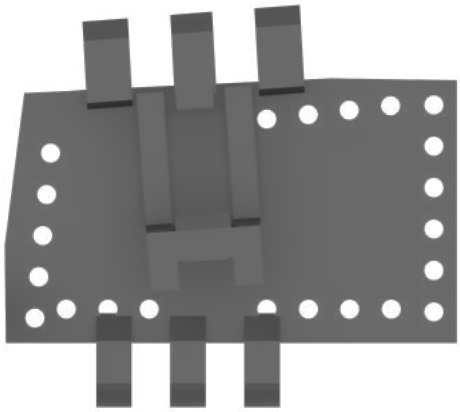


Abbildung 3.21.: Obersicht Daumenplatte

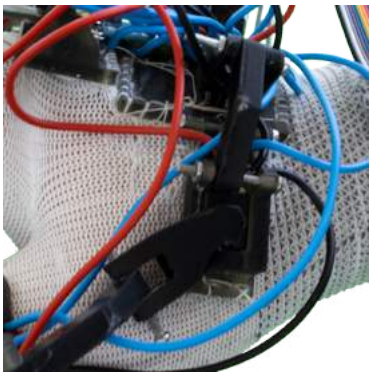


Abbildung 3.22.: Daumenkoppelung

Name	Daumenplatte
Datei	`controller_plate_thumb`
LxBxH	3.89mm x 29mm x 14.9mm
Beschr.	Dieses Element erfüllt die Rolle, Daumenverformungen abzufangen, die den gemessenen Wert verfälschen könnten. Das festere Gelenk verhindert dabei eine Drehung, die die Messung am nächsten Element beeinflussen würde.

Tabelle 3.6.: Details Daumenplatte

Auf der äußeren Seite der Hand ist eine weitere Platte befestigt (Abbildung 3.23 bis 3.25), welche dazu dient, einen Basispunkt für Ringfinger und kleinen Finger zu bieten. Dies kann nicht über die innere Platte erfolgen, da sich die menschliche Hand im normalen Gebrauch so verformt, dass eine geschlossene Fläche am Handrücken die Bewegungsfreiheit einschränken würde. Somit werden zwei unabhängig bewegliche Teile genutzt.

Abbildung 3.23.: Frontansicht äußere Platte

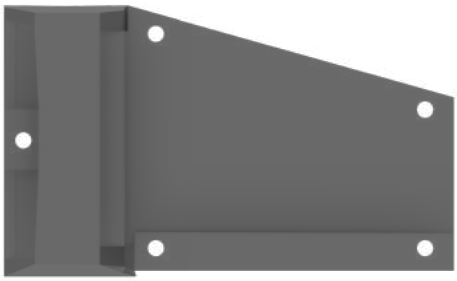


Abbildung 3.24.: Obersicht äußere Platte

Abbildung 3.25.: ausgedruckte äußere Platte

Name	äußere Handrückenplatte
Datei	`controller_plate_outer`
LxBxH	60mm x 36.1mm x 12.6mm
Beschr.	Dieses Element erfüllt die Rolle, als Basis für die äußeren zwei Finger zu dienen. Sie ist von der inneren Platte getrennt, um volle Bewegungsfreiheit zu ermöglichen.

Tabelle 3.7.: Details äußere Platte

Auf den großen Platten sind Multiplexer für die entsprechenden Finger, sowie Querverbindungen für die Versorgungsspannung montiert.

Bei der Integrierung der Potentiometer werden Hebel genutzt, die genau in den sechseckigen Ausschnitt des Schleifers passen (Abbildung 3.26 bis 3.28). Diese müssen sehr genau bemessen und gedruckt sein, da es sonst zu Abrieb des Epoxidharzes, oder zu Komplikationen bei der Konstruktion kommen kann. Um Kompatibilität für die verschiedenen Gelenke und Gelenkabstände sicherzustellen gibt es drei Ausführungen dieser Hebel in verschiedenen Längen. Zu jeder Länge wird auch ein Gegenstück bereitgestellt, das für verbesserte Stabilität sorgt.

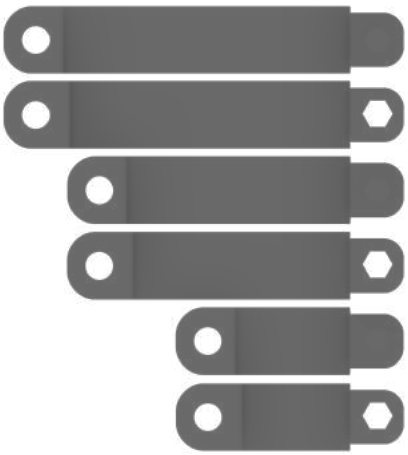


Abbildung 3.26.: Obersicht Hebel

Abbildung 3.27.: Frontansicht Hebel

Name	`controller_lever`
LxBxH L	31.5mm x 13.8mm x 5.2mm
LxBxH M	26.5mm x 13.8mm x 5.2mm
LxBxH S	17.9mm x 13.8mm x 5.2mm

Tabelle 3.8.: Details Hebel

Abbildung 3.28.: ausgedruckte Hebel

In Kombination mit diesen Hebeln werden Streben (Abbildung 3.29 und 3.30) genutzt, die dazu dienen, beim Abbiegen des zu messenden Gelenks dem Finger auszuweichen, ohne die Messung zu behindern. Die Potentiometer werden nicht an der Seite der Finger angebracht, da dies die Bewegungsfreiheit der Finger einschränken, und somit den Tragekomfort deutlich reduzieren würde. Die Streben wurden in 5 Größen kreiert, um für alle Gelenke bei den meisten Handgrößen anwendbar zu sein. Bei der kleinsten Variante ist zusätzlich der Winkel anders, um genauere Messungen zu erzielen.

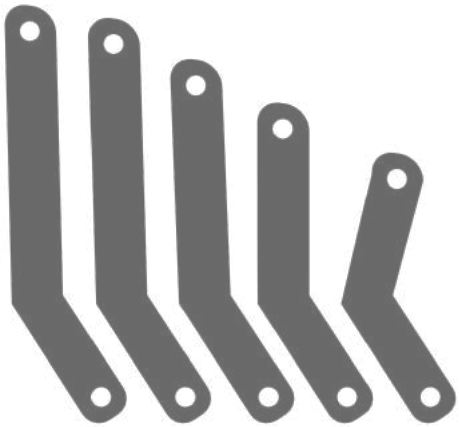


Abbildung 3.29.: Obersicht Streben

Abbildung 3.30.: ausgedruckte Streben

Name Datei	ausgedruckte Streben `controller_strut`
LxBxH XL	31.1mm x15.3mm x 3.8mm
LxBxH L	30.3mm x 14.7mm x 3.8mm
LxBxH M	27.6mm x 12.8mm x 3.8mm
LxBxH S	24.4mm x 11.4mm x 3.8mm
LxBxH XS	19.8mm x 11.3mm x 3.8mm

Tabelle 3.9.: Details Streben

An allen Streben und Hebeln sind Löcher mit einem Durchmesser von 2.2mm vorgesehen, welche für M2-Schrauben geeignet sind. Obwohl Tests bewiesen haben, dass print-in-place-Gelenke möglich wären, würde dies gegen die Modulare Struktur sprechen, die es ermöglicht, auf vielen Verschiedenen Händen nutzbar zu sein. Somit werden Schrauben benötigt, um die Teile zu verbinden.

Um die Rotation der Finger aufzunehmen werden die untersten Streben nicht direkt mit der Basisplatte verbunden, sondern über ein weiteres Potentiometer. Diese Verbindung erfolgt über das folgende Modul auf Abbildung 3.31 bis 3.33 dargestellt ist:

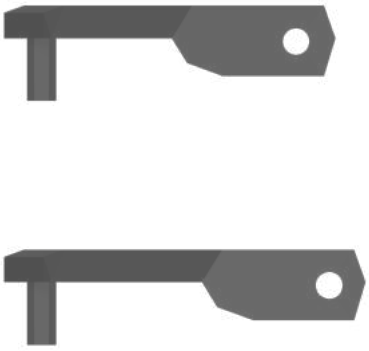


Abbildung 3.31.: Seitenansicht Fingerbasisgelenk

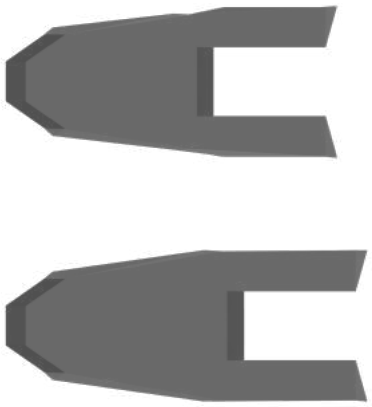


Abbildung 3.32.: Obersicht Fingerbasisgelenk

Name	Fingerbasisgelenk
Datei	`controller_strut_lr`
LxBxH M	28.4mm x 11.9mm x 7.46mm
LxBxH S	26mm x 11.9mm x 7.46mm

Tabelle 3.10.: Details Fingerbasisgelenke

Abbildung 3.33.: ausgedruckte Fingerbasisgelenke

4. Softwarearchitektur

In der nachstehenden Grafik (Abb. 4.1) ist die grundlegende Architektur der Software zu sehen:

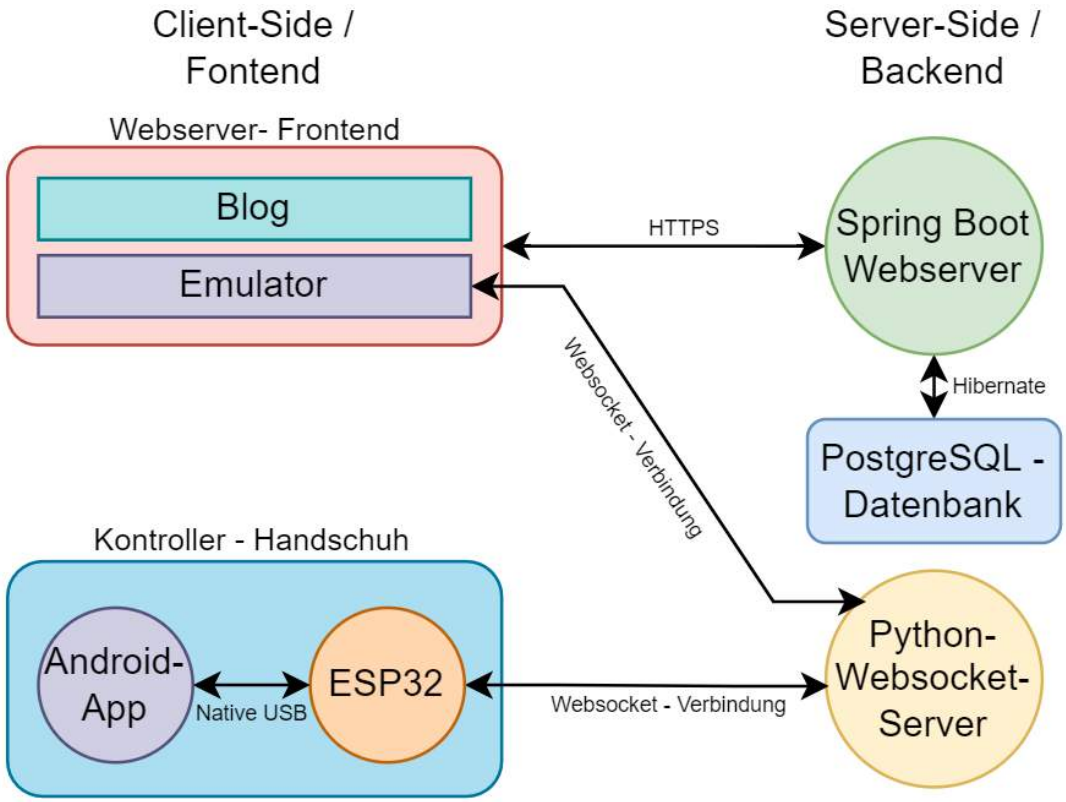


Abbildung 4.1.: Softwarearchitektur

Zu erkennen sind hier fünf wesentliche Komponenten:

- Spring Boot[1] - Webserver
- Websockets[2] - Server
- PostgreSQL[3] - Datenbank

- Controller - Handschuh, welcher den ESP32[8] und eine Android[4]-App umfasst
- Webserver - Frontend

In den Kapitel 7, 6 & 5 werde die einzelnen Komponenten genauer erläutert.

4.1. Erklärung der Softwarearchitektur

Der grundlegende Gedanke hinter der Softwarearchitektur ist jener, dass ein Emulator und ein Blog für die registrierten Nutzer verfügbar ist. Um dies zu realisieren, wurde ein Spring-Boot Webserver bereitgestellt, welcher die nötigen Funktionalitäten beherbergt, um diese Webapplikationen sinvoll und möglichst effizient bereitstellen zu können.

Mit dem Java-Package *Hibernate*[6] wurde eine Verbindung zu einer PostgreSQL Datenbank hergestellt, welche die registrierten Nutzer, die erstellten Blog-Einträge und etwaige andere benötigte Daten persistiert.

Der Python Websocket-Server wurde benötigt, um eine echtzeitnahe Verbindung mit dem ESP32 (Controllerhandschuh) herzustellen. Prinzipiell wäre dies auch mit Java-Websockets möglich, jedoch verwendet das Framwork *Spring* das Session-Level Protokoll **STOMP**[5]. Da es in dem Framework *Micropython*[7] keine Realisierung dafür gab, wurde einfachkeitshalber ein Websocket Server mithilfe von Python aufgesetzt, welcher native Socketverbindungen akzeptiert. Um die erfassten Daten nun auch an die Clienten weiterreichen zu können muss dieser Server ebenfalls wissen, wer die registrierten Clienten sind, dies wird über Python-Dictionaries realisiert, nähere Informationen, siehe Kapital 6.

Damit die Software die empfangenen Daten dem entsprechenden Nutzer zuordnen kann, muss der ESP32 nach dem Bootvorgang konfiguriert werden. Hierfür wurde eine Andriod-Applikation entwickelt, welche über eine native USB Verbindung mit dem Controller kommuniziert und diesen dann sagt, mit welchem WLAN sich der ESP32 verbinden soll und welcher Nutzer gerade verbunden ist.

5. Spring Boot Webserver

In Kapitel 4 wurde kurz die Notwendigkeit des Java Webservers erläutert, jedoch umfasst dieser weit mehr Funktionalitäten, als im vorrigen Kapitel beschrieben. Beispielsweise muss dieser Server User-Authentication handhaben, HTML-Templates rendern, Modell-Data bereitstellen und aus der Datenbank fetchen, STOMP-Endpoints bereitstellen uvm. können . Für Erklärung der jeweiligen Funktionalitäten, vergleiche nachfolgende Kapitel.

Das fertige Projekt kann auf folgendem Github-Repository gefunden werden:

`https://github.com/psykovski-extended/binobo.git`

5.1. Spring

Das Java-Framework **Spring**, oder genauer **Spring Boot** bietet diverse Funktionalitäten, welche das Entwickeln und Bereitstellen eines Webserver maßgeblich vereinfachen, wie zum Beispiel:

- Dependency Injection[28]
- Spring JDBC[29]
- Spring ORM
- Spring MVC[23]
- Spring Security[13]
- Spring AOP[30]
- Spring Test
- Embedded Webserver (Tomcat[21], TomEE[22], ...)
- Health-Checks der aktuell laufenden Applikation
- Externalised Configuration durch `application.properties` - Files

Die Entwicklerumgebung **IntelliJ**[10] bietet einen eingebetteten **Spring-Initializr**[11] Projektsetup-Dialog, bei welchen man die benötigten Dependencies direkt vorweg auswählen kann. Weiters werden hierdurch Standardkonfigurationen in das `pom.xml` oder `build.gradle` File geladen, abhängig von dem gewählten Build und Dependency - Management Tool.

Bei diesem Projekt wurde das Dependency-Management und Build Tool **Maven**[12] gewählt. Nachstehend ist eine kurze Zusammenfassung der verwendeten Java-Libraries gelistet:

- Spring Boot Security[13]
- Spring Boot Data REST[14]
- Spring Boot Data JPA[15]
- Thymeleaf[16]
- Spring Boot Starter Web[17]
- Spring Boot Validation[18]
- Lombok[19]
- Tomcat[21]
- Inject[28]
- PostgreSQL Connector[3]
- Spring Boot Mail
- Spring Boot Websockets

Eine genaue Auflistung aller verwendeten Libraries ist im Github-Repository [50] zu finden.
Aus dieser Liste kann man ebenfalls entnehmen, dass als Webserver der Apache Tomcat verwendet wird.

5.1.1. Spring MVC

`Spring MVC`[23] ist ein Modul des Spring Frameworks, welches die Integration des MVC Design-Patterns ermöglicht. `MVC` steht für `Model - View - Controller` und ist ein Entwicklungsschema für Serveranwendungen.

Model
Model steht in diesem Sachzusammenhang für die persistenten und objektrelationalen Datenbanktabellen, welche auf Javaobjekte abgebildet werden. Model-Daten werden unter anderem von der Template-Rendering Engine `Thyemleaf` dafür verwendet, um dynamische HTML-Templates zu rendern.
Die Model-Daten werden über die `Controller` an Thymeleaf übergeben durch sogenannte `Modelattribute`.

View
Das View (englisch für Präsentation) referiert in Spring auf die HTML-Templates, welche durch Thymeleaf gerendert werden. Anders als die Models gibt es keine diskrete Softwareschicht, welche die Views repräsentiert, diese Schicht zeichnet sich nur durch die gerenderten HTML-Templates aus.

Controller
Die Controller stellen in Spring Boot die `HTTP-Endpoints` dar, welche die eingehenden HTTP-Requests annehmen und verarbeiten. Weiters ist diese Schicht dafür da, die Model-Daten an die Templates weiterzugeben. Für nähere Informationen zu Controller, vergleiche Kapitel 5.1.5

5.1.2. Seperation of Concerns

Bei `Seperation of Concerns`[24] geht es um die Seperation der Fehler durch das direkte zuordnen von Java-Packages zu genau einer Aufgabe ($\hat{=}$ single-responsibility-approach). Hierdurch wird das Lösen von auftretenden Fehler maßgeblich vereinfacht.

Hierbei verpackt man Javaklassen, welche zum Beipiel die Service-Schicht einer Applikation darstellen, in einem `Package`, welches `service` genannt wird, zusammen. Das macht man dann mit den weiteren `Layern` des Projekts mit genau dieser Vorgehensweise - man benennt ein `Package` nach der Schicht, welche die darin enthaltenen Javaklassen abbilden.

```
1 package htlstp.diplomarbeit.binobo.service;
2
3 import htlstp.diplomarbeit.binobo.model.Role;
4
5 import java.util.List;
```

```
6
7  public interface RoleService {
8      List<Role> findAll();
9      Role findById(Long id);
10 }
```

Listing 5.1: Role-Service Interface

Dieses Beipiel (Listing 5.1) zeigte ein Java-Interface, welches die Funktionalität eines Services bereitstellt. Anhand des `package` - Schlüsselwortes kann man den Pfad dieses Interfaces erkennen: `htlstp.diplomarbeit.binobo.service`.
Es wurde darauf geachtet, jede Schicht der Softwarearchitektur in entsprechned benannte Packages zu verpacken um möglichst strukturiert und übersichtlich zu sein.

5.1.3. Spring Konfiguration

Damit ein Spring Boot Projekt ordnungsgemäß funktioniert und die benötigten Applikationsumgebungsvariabeln bereitzustellen, muss ein `application.properties` File erstellt werden mit den benötigten Konfigurationen, wie zum Beispiel die Datasoucre-URL. Listing 5.2 zeigt die für dieses Projekt erstelle Konfigurationsdatei:

```
1  # creates SPRING_SESSION db
2  spring.session.store-type=jdbc
3  spring.session.jdbc.initialize-schema=always
4
5  # server port
6  server.port=80
7
8  # setup PostgreSQL Database and Hibernate config
9  spring.datasource.url=jdbc:postgresql://localhost:3406/binobo_db
10 spring.datasource.username=postgres
11 spring.datasource.password=<root_password>
12 spring.jpa.hibernate.ddl-auto=update
13 spring.jpa.show-sql=false
14 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.
       PostgreSQL92Dialect
15
16 # mail config for email-verification
17 spring.mail.host=smtp.gmail.com
18 spring.mail.port=587
19 spring.mail.username=dominik.lovetinsky99@gmail.com
20 spring.mail.password=<email_password>
21 spring.mail.properties.mail.smtp.auth=true
22 spring.mail.properties.mail.smtp.starttls.enable=true
23
24 server.ssl.enabled=false
```

Listing 5.2: application.properties

Zu beachten ist, dass die Passwörter entfernt wurden, aus sicherheitstechnischen Gründen.
Die erste Konfigurationszeile mit `spring.session.store-type=jdbc` sorgt dafür, dass die Backend-Session für Datenbankkommunikation mit `JDBC`[29] initialisiert werden soll. Für nähere Informationen zu `JDBC`[29] vergleiche Kapitel 5.2.3.
Die zweite Zeile stellt ein, dass die Datenbank-Schemas immer bei start des Programmes initialisiert werden soll. Mit der Zeile `server.port=80` wird der Serverport auf den standard HTTP-Port `80` gelegt.
Die nachfolgenden Zeilen konfigurieren Hibernate und die dafür Notwendigen Umgebungsvariablen, wie zum Beipiel die `Datasource-URL` mit `spring.datasource.url=jdbc:postgresql://localhost:3406/binobo_db`, und die dafür notwendigen Login-Daten mit `spring.datasource.username=postgres` und `spring.datasource.password=<roor_password>`.

Danach ist dann die Email-Konfiguration zu sehen, welche benötigt wird, um über den Java-Mailsender Emails zu versenden. Diese Funktionalität wird benötigt, um die Emailadressen der registrierten Nutzer zu bestätigen.

Schlussendlich wurde dann noch `SSL`[33] für diesen Server deaktiviert. Da jedoch aus sicherheitstechnischen Gründen `HTTPS` verwendet werden sollten, wenn man mit einem Server kommuniziert, wurde dies dann mithilfe eines `Reverse-Proxys` und dem dazugehörigen `certbot`[34] gemacht.

5.1.4. Spring Security

`Spring Security`[13] wird benötigt um Registrierungen und eingeloggte Nutzer zu verwalten - dies wäre auch möglich ohne diesem Package, jedoch vereinfacht dies die Implementierung von Authentication und Authorization maßgeblich, sowie die Generierung und Verwaltung des `JSESSIONID` - Cookies.
Listing A.1 zeigt die erstellte Konfigurationsdatei.

Da dieses Code-Snippet sich selbst durch entsprechende Kommentare erklärt, ist dieses im Anhang zu finden.

5.1.5. Controller und Rest-Controller

`Controller` und `Rest-Controller`[31] sind HTTP - Endpoints, welche HTTP - Requests annehmen und verarbeiten. Damit Spring erkennt, welche Klasse ein Controller oder Rest-Controller ist, muss hier ebenfalls wieder die entsprechende Annotation verwendet werden.

Nachstehendes Listing zeigt die `UserController` Klasse, welche für das Verarbeiten und Rendern der Profilseite benötigt wird:

```
1  @Controller
2  @RequestMapping(value = "/user")
3  public class UserController {
4
5    private final PostService postService;
6    private final UserService userService;
7    private final BookmarkService bookmarkService;
8
9    @Autowired
10   public UserController(PostService postService, UserService
       userService,
11                         BookmarkService bookmarkService){
12     this.postService = postService;
13     this.userService = userService;
14     this.bookmarkService = bookmarkService;
15   }
16
17   @GetMapping(value = "/profile")
18   public String getProfileInfo(Model model, Principal principal){
19     User user = (User)((UsernamePasswordAuthenticationToken)
         principal).getPrincipal();
20     model.addAttribute("user", user);
21     List<Post> posts = new ArrayList<>();
22     List<Bookmark> bookmarks = bookmarkService.findAllByUser(user);
23     bookmarks.forEach(element -> {
24         posts.add(postService.findById(element.getPost().getId()));
25     });
26
27     model.addAttribute("bookmarks", posts);
28     model.addAttribute("posts", postService.findByUser(user));
29
30     return "user/profile";
31   }
32
33 }
```

Listing 5.3: User-Controller

Auf Zeile `1` sieht man direkt die Annotation `@Controller`, welche Spring mitteilt, dass diese Klasse ein Controller ist, welcher HTTP anfragen mit den Pfad `/user/**` annimmt, wie es auf Zeile `2` mit `@RequestMapping(value = /user")` durch eine weitere Annotation bestimmt wurde.
Zeile `9` ist mit `@Autowired` annotiert, dies wird benötigt für `Dependency-Injection`. Für nähere Informationen hierzu, vergleiche Kapitel 5.1.9.

Sogenannte HTTP-Endpoints werden durch das Annotieren einer Funktion mit einer der folgenden Annotationen gekennzeichnet:

- `@GetMapping`
- `@PostMapping`
- `@PutMapping`
- `@PatchMapping`
- `@DelteMapping`

Nachstehendes Listing zeigt einen dieser Endpoints, welcher Daten an das Model übergibt, welche für das Rendern des Views benötigt werden:

```
1 @GetMapping(value = "/blog")
2 public String listAllBlogs(Model model){
3   List<Post> posts = postService.findAll();
4   model.addAttribute("posts", posts);
5
6   return "blogOverview";
7 }
```

Listing 5.4: HTTP-Endpoint

Die Annotation `@GetMapping` teilt Spring Boot mit, dass diese Funktion Get-Requests der URI `/blog` annimmt und verarbeitet. Übergeben wird von Spring ein Model-Objekt, zu welchen die Daten zur weiteren Verarbeitung übergeben werden. Thymeleaf nimmt diese Daten und rendert damit das zurückgegebene HTML-Template `blogOverview`.
Die Daten werden an das Model durch den Funktionsaufruf `model.addAttribute` übergeben, und nimmt in dieser Version zwei Parameter an: Einmal der Name der Modeldaten wie es in dem jeweiligen Template referenziert wird und als zweiten Parameter die Objektdaten.
Das Returnstatement dieser Funktion teilt Thymeleaf mit, welches Template gerendert werden soll.

5.1.6. Services

Die Definition eines `Services` kommt ursprünglich von einem Design-Pattern, welches im Jahre 2003 von **Eric Evans** erstellt wurde: `Domain-Driven Design` (kurz: DDD)[25]. Hierbei geht es darum, komplexe Software in klar definierte Abstraktionsschichten zu unterteilen. So findet das DDD-Pattern auch in Spring Boot Anwendung.

Ein Service ist in diesem Sinne eine Klasse, welche gewisse Funktionen zur Verarbeitung von Daten bereitstellt.
In Spring Boot wird diese Schicht verwendet, um von den Controllern aus mit Datenbanken zu kommunizieren. Der Programmfluss geht demnach zuerst vom Controller in eine Service-Schicht und von diesem Service dann entweder wieder weiter zu einer anderen Service-Schicht oder zu einer Repository-Schicht.

In Spring Boot wird die Service-Schicht üblicherweise über Interfaces definiert, welches in einer Java-Klasse mit der Annotation `@Serive` implementiert wird. Durch diese Annotation erstellt man gleichzeitig eine `Bean`[32] auf welches durch Dependency-Injection[28] zugegriffen werden kann, für nähere Informationen zu `Dependency Injection`, vgl. Kapitel 5.1.9.

Nachstehend Listing zeigt ein Interface und die dazugehörige Implementierung von diesem:

```
1  package htlstp.diplomarbeit.binobo.service;
2
3  import htlstp.diplomarbeit.binobo.model.Category;
4  import java.util.List;
5
6  public interface CategoryService {
7      List<Category> findAll();
8
9  }
```

Listing 5.5: Category-Service Interface

```
1  package htlstp.diplomarbeit.binobo.service;
2
3  import htlstp.diplomarbeit.binobo.model.Category;
4  import htlstp.diplomarbeit.binobo.repositories.CategoryRepository;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.stereotype.Service;
7
8  import java.util.List;
9
10 @Service
11 public class CategoryServiceImpl implements CategoryService{
12
13     private final CategoryRepository categoryRepository;
14
15     @Autowired
16     public CategoryServiceImpl(CategoryRepository categoryRepository
           ) {
17         this.categoryRepository = categoryRepository;
18     }
```

```
19
20      @Override
21      public List<Category> findAll() {
22          return categoryRepository.findAll();
23      }
24  }
```

Listing 5.6: Category-Service Implementierung

Erkennbar ist die Annotation `@Service` und die Implementierung des Interfaces `CategoryService` in der Javaklasse `CategoryServiceImpl`. Ebenfalls zu sehen sind die implementierten Funktionen aus `CategoryService`, gekennzeichnet durch @Override.
Dieser Service wird zum Anzeigen der verfügbaren Kategorien, unter welchen Blogeinträge kategorisiert werden können, benötigt.

5.1.7. Asynchrones Programmieren

Asynchrones Programmieren ist eine aus der Webprogrammierung stammende Begrifflichkeit. Asynchrone Prozesse sind Prozesse, welches parallel zum Hauptprogramm laufen, aber nicht auf einem sepeartem Thread, sondern durch den Scheduler in kleinen Portionen am Mainthread abgearbeitet werden. Durch asynchrone Programmierung entseht eine `non-blocking` und `event-driven` Software[27].

Um in Spring Boot hiervon gebrauch machen zu können, muss man eine weitere Konfigurationsdatei erstellen:

```
1  package htlstp.diplomarbeit.binobo.configurator;
2
3  import org.springframework.context.annotation.Bean;
4  import org.springframework.context.annotation.Configuration;
5  import org.springframework.scheduling.concurrent.
       ThreadPoolTaskExecutor;
6
7  import java.util.concurrent.Executor;
8
9  /**
10   * This class configures all asynchronous tasks, wich eventually
         will get executed and yielded to the scheduler.
11   * The maximum amount of async task is set to 20, because there is
         no need for more in the current state of the project.
12   */
13 @Configuration
14 public class AsyncConfig {
15
16     /**
```

```
17        * Global bean, says Spring that this has to be used to
              instantiate the ThreadPoolExecutor
18        * @return Returns the configured ThreadPoolExecutor
19        */
20      @Bean
21      public Executor taskExecutor() {
22          ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor
                ();
23          executor.setCorePoolSize(20);
24          executor.setMaxPoolSize(20);
25          executor.setQueueCapacity(500);
26          executor.setThreadNamePrefix("Async␣Process␣--␣");
27          executor.initialize();
28          return executor;
29      }
30
31  }
```

Listing 5.7: Async - Task Konfiguration

Prinzipiell macht Spring Boot von asynchromen Tasks standardgemäß gebrauch beim Verarbeiten eingehender HTTP-Request, um jedoch selbst asynchrone Tasks erstellen zu können, muss ein `Bean` mit dem Rückgabewert `Excecutor` erstellt werden. Benutzerdefinierte asynchrone Tasks wurden in diesem Projekt benötigt um die Datenbank auf veraltete Einträge in der Tabelle für `robotdata` zu durchsuchen und um diese bei Bedarf zu löschen. Da anfänglich die erfassten Werte in dieser gespeichert wurden. Nach etwaigen Optimierungen viel die Notwendigkeit dieser Aufgabe jedoch weg, weswegen benutzerdefinierte asynchrone Tasks nicht mehr benötigut werden auf dem Server.

5.1.8. Field-Matcher Annotation

Eine eigens entwickelte Annotation, welche benötigt wird, um zu überprüfen ob das `password` mit `password_verify` Feld übereinstimmt.
Genauso wie ein Interface, benötigt eine Annotation ebenfalls eine `Blueprint` - Definition und eine Implementierung. Beide Features sind standardgemäß in der Programmiersprache Java eingebaut und sind nicht Spring-Spezifisch. Folgendes Listing zeigt das Interface und die Implementierung:

```
1  package htlstp.diplomarbeit.binobo.service.validation;
2
3  import javax.validation.Payload;
4  import javax.validation.Constraint;
5  import java.lang.annotation.Documented;
6  import java.lang.annotation.Retention;
7  import java.lang.annotation.Target;
```

```
8  import static java.lang.annotation.ElementType.ANNOTATION_TYPE;
9  import static java.lang.annotation.ElementType.TYPE;
10 import static java.lang.annotation.RetentionPolicy.RUNTIME;
11
12 @Target({TYPE, ANNOTATION_TYPE})
13 @Retention(RUNTIME)
14 @Constraint(validatedBy = FieldMatchValidator.class)
15 @Documented
16 public @interface FieldMatch
17 {
18     String message() default "The␣fields␣must␣match";
19     Class<?>[] groups() default {};
20     Class<? extends Payload>[] payload() default {};
21     String first();
22     String second();
23
24     @Target({TYPE, ANNOTATION_TYPE})
25     @Retention(RUNTIME)
26     @Documented
27     @interface List
28     {
29         FieldMatch[] value();
30     }
31 }
```

Listing 5.8: FieldMatch Annotation

```
1  package htlstp.diplomarbeit.binobo.service.validation;
2
3  import org.apache.commons.beanutils.BeanUtils;
4
5  import javax.validation.ConstraintValidator;
6  import javax.validation.ConstraintValidatorContext;
7
8  public class FieldMatchValidator implements ConstraintValidator<
       FieldMatch, Object> {
9
10     private String firstFieldName;
11     private String secondFieldName;
12     private String message;
13
14     @Override
15     public void initialize(final FieldMatch constraintAnnotation) {
16         firstFieldName = constraintAnnotation.first();
17         secondFieldName = constraintAnnotation.second();
18         message = constraintAnnotation.message();
19     }
20
21     @Override
22     public boolean isValid(final Object value, final
           ConstraintValidatorContext context) {
```

```
23          boolean valid = true;
24          try {
25              final Object firstObj = BeanUtils.getProperty(value,
                    firstFieldName);
26              final Object secondObj = BeanUtils.getProperty(value,
                    secondFieldName);
27
28              valid =  firstObj == null && secondObj == null ||
                    firstObj != null && firstObj.equals(secondObj);
29          }
30          catch (final Exception ignore) {}
31
32          if (!valid){
33              context.buildConstraintViolationWithTemplate(message)
34                      .addPropertyNode(firstFieldName)
35                      .addConstraintViolation()
36                      .disableDefaultConstraintViolation();
37          }
38
39          return valid;
40      }
41  }
```

Listing 5.9: FieldMatchValidator Klasse

Gedacht war diese Annotation für Java-Strings, sie kann jedoch für jeden beliebigen der nicht-primitiven Datentypen verwendet werden.

5.1.9. Dependency Injection

Dependency Injection[28] ist ein in der objektorientierten Programmierung verwendetes Entwurfsschema, welches die Abhängigkeiten eines Objekts zur Laufzeit reglementiert. Diese Abhängigkeiten werden an einem zentralen Ort hinterlegt - in Spring Boot sind dies üblicherweise `Service` Implementierungen, welche in einem Package mit Namen `serivce` hinterlegt sind.
Ein großer Vorteil hiervon ist, dass nicht bei jeder Verwendung eines Interfaces dieses in der jeweiligen Klasse initialisiert werden muss. In Spring wird dies über `Java-Beans` geregelt, welche über `@Autowired` referenziert werden können. Spring erkennt dann die jeweils benötigte Resource, welche injiziert werden muss.
Folgendes Code-Snippet zeigt ein Beispiel einer Dependency-Injection:

```
1  private final PostService postService;
2  private final UserService userService;
3  private final BookmarkService bookmarkService;
4
5  @Autowired
```

```
6  public UserController(PostService postService, UserService
       userService, BookmarkService bookmarkService){
7      this.postService = postService;
8      this.userService = userService;
9      this.bookmarkService = bookmarkService;
10 }
```

Listing 5.10: User - Controller, Autowired Konstrukteur

Die benötigten Resourcen werden beim Aufrufen dieses Konstruktors von Spring übergeben.

5.1.10. Websockets und STOMP

Websockets sind eine Erweiterung des HTTP-Protokolls, welche dafür sorgt, dass HTTP-Verbindungen nach dessen Erstellung, aufrecht erhalten werden um so einen schnellen Datenaustausch zwischen zwei oder mehreren Clients zu ermöglichen.
`STOMP`[5] steht für `Simple-Text-Oriented-Messaging-Protocol` und ist ein Protkoll, welches für Datenaustausch über Socket-Verbindungen verwendet wird. Jedoch aufgrund der nicht gebrauchten *Komplexität* dieses Protokolls, welches Spring für Websocketverbindungen voraussetzt, wurde ein weiterer Server, welcher nur Websocketverbindungen akzeptiert, programmiert. Dies ermöglicht einen einfachen Datenaustausch zwischen dem `ESP32` und dem Webbrowser.
Für nähere Informationen zu Websockets und dem Websocketserver, vgl. Kapitel 5.4.2 & 6.

5.2. Hibernate

Auch wenn Spring eine Abstraktionsschicht für JPA, ORM und JDBC bietet, benötigt man dennoch eine weitere Schicht, welche die Funktionalitäten hierfür beherbergt. Hibernate[6] ist in diesem Falle das Java-Framework, welche die Notwendigen Funktionalitäten bereitstellt um mit Datenbanken interagieren zu können.
Sowohl Hibernate, als auch Spring sind Annoation basierend. Das heißt, dass der JVM und somit auch Spring durch diverse Annotations gesagt wird, welche Klasse welche Funktionalität bereitstellt und wo welche `Beans` zu finden sind.

5.2.1. ORM

ORM steht für `Object Related Mapping`. Hierbei wird der objektorientierte Teil der Software, welcher persistent sein muss, durch sogenannte Entitäten ($\hat{=}$ POJOs) auf Datenbanktabellen abgebildet.

Der eben erwähnte Begriff **POJO** steht in der Javaprogrammierung für `Plain old Java Objects`. Ein POJO ist also eine klassische Java-Objekt-Definition mit standard Konstrukteur, Getter und Setter.

Folgende Objekte in diesem Projekt müssen persistent definiert sein:

- `User` - Die registrierten Nutzer
- `Post` - Blog-Eintrag eines Nutzers
- `Comment` - Kommentar eines Blog-Eintrags
- `Bookmarks` - Lesezeichen, um Blogeinträge zu speichern
- `Vote` - Definiert ob ein Kommentar nützlich oder unnütz ist
- `Category` - Blogeinträgen werden Katagorien zugewiesen, um sie besser einordnen zu können
- `Role` - Definiert die Rolle eines Users
- `API_Key` - Wird benötigt um asynchrone, aber sichere Rest-Calls an die Blog-API zu senden
- `DataAccessToken` - Dient der Zuordnung der erfassten Rotationsdaten auf den entsprechenden Nutzer
- `ConfirmationToken` - Dies ist ein 24h lange gültiger Token, welcher benötigt wird, um den Account eines Nutzers zu aktivieren

Solche persistenten Javaobjekte nennt man `Entities` oder `Models`.

Zur Erläuterung der Relationen der Datenstrukturen, siehe Kapitel 5.2.5.

Um der JVM mitzuteilen, dass ein POJO auf eine Datenbanktabelle abgebildet werden soll, muss diese Java-Objekt Klassendefinition mit `@Entity` von `javax.persistence` annotiert werden.
Weiters muss ebenfalls der `Primary-Key` dieser Tabelle definiert werden und die jeweiligen Spalten, welche das Abbild (die Tabelle in der Datenbank) dieses Objekts enthalten soll, folgendes Listing zeigt die Klassendefinition `Role` dieses Projekts um ein kurzes Beispiel zu listen:

```java
package htlstp.diplomarbeit.binobo.model;

import javax.persistence.*;

@Entity
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column
    private String name;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Listing 5.11: Role - Entity

Aus diesem Beispiel kann man zumal den `Primary-Key` direkt herauslesen, welcher mit `@Id` und `@GeneratedValue(strategy = GenerationType.IDENTITY)` annotiert ist. Die Annotation `@Id` gibt an, dass dieses Feld der primäre Schlüssel dieses Objekts ist, mit der zweiten Annotation dieses Objekts wird bestimmt, dass dieser Wert automatisch generiert werden soll, jedoch mit nicht zufällig, sondern inkremental generiert wird.
Das zweite Feld dieser Klasse (`private String name`) wurde mit `@Column` annotiert, welche Hibernate sagt, dass dieses Feld eine Spalte in der Datenbank abbilden soll, mit dem Datentyp `VARCHAR` (`String` entspricht in PostgreSQL einem `VARCHAR`). Der name dieser Spalte ist dann der Name des Feldes.

5.2.2. JPA

Um Java den Zugriff auf die Datenbank zu ermöglichen, benötigt man die `Java Persitence API`[15]. Diese API ermöglicht es Java eine Kommunikation mit der Datenbank herzustellen um Transaktionen zu tätigen.

JPA sorgt somit für die Persistenz der Java-Laufzeitobjekte indem diese die Objekte in der Datenbank speichert.

Der Unterschied zwischen `ORM` und `JPA` besteht darin, dass die Java Persitence API zwar die Daten laut POJO-Definition in der Datenbank persistiert, jedoch benötigt die JPA das ORM um zu erkennen, wie diese Objekte gespeichert werden sollen, deswegen spezifiziert man:

- JPA - Tätigt die notwendigen Transaktionen, um der Datenbank die notwendigen Instruktionen zu schicken, basierend auf den ORM-Patterns.
- ORM - Objektrelationales Abbilden von Javaobjekten auf Datenbanktabellen

Mit JPA werden relationale Datenbankstrukturen auf Java Objekte Abgebildet und bietet ebenfalls vordefinierte generische Java Interfaces, welche vollkommen automatisch Funktionen für die entsprechenden Objekte erstellt. Typischerweise realisiert man das Abbilden eines Objekte auf eine Datenbanktabelle zuerst mit der Annotation `@Entitiy` und den dazugehörigen Spalten der Tabelle, jedoch um nun auch Datenbankoperationen ausführen zu können, muss in dem Spring Boot Projekt eine sogenannte `Repository` - Schicht erstellt werden.
Hierfür bedient man sich eines Java-Interfaces, welches ein Kind von der JPA-Klasse `JPARepository<T, ID>` ist. Der Java-Diamond-Operator '<>' wird in der Javaprogrammierung zur Deklaration einer `Generika` verwendet. Generika sind Java-Klassen, welche mit mehreren Datentypen von Java Kompatibel sind.
Eine solche Interface-Definition um objektspezifische Datenbankoperationen erstellen zu lassen, sieht wiefolgt aus:

```
1  package htlstp.diplomarbeit.binobo.repositories;
2
3  import htlstp.diplomarbeit.binobo.model.DataAccessToken;
4  import htlstp.diplomarbeit.binobo.model.robo.RobotData;
5  import org.springframework.data.jpa.repository.JpaRepository;
6  import org.springframework.stereotype.Repository;
7
8  import java.util.List;
9  import java.util.Optional;
10
11 @Repository
12 public interface RobotDataRepository extends JpaRepository<RobotData
       , Long> {
13   RobotData findTopByOrderByIdAsc();
14   List<RobotData> findAllByDataAccessToken(DataAccessToken
         dataAccessToken);
15   Optional<RobotData> findTopByDataAccessToken(DataAccessToken
         dataAccessToken);
```

```
16    void deleteAllByDataAccessToken(DataAccessToken dataAccessToken);
17 }
```

Listing 5.12: JPA-Repository Interface Implementierungen

Dieses Interface ist ebenfalls in den oben gelisteten Github-Repository zu finden.

Etwas, was bei diesem Interface direkt auffällt, ist die Annotation `@Repository`, diese teilt Spring mit, dass dieses Interface eine `JavaBean` von Typ Repository ist.
Weiters kann man erkennen, dass die Funktionsnamen einer sehr deutlichen Struktur folgen. Spring erkennt nämlich an diesen Funktionsnamen, welche Datenbankoperation man bei Aufruf einer dieser Funktionen ausführen will und initialisiert während der Runtime den entsprechenden Funktionskörper (eng. `Function-body`).

5.2.3. JDBC

`JDBC` steht für `Java Database Connectivity` und ist eine universelle Datenbankschnittstelle für die Programmiersprache Java und ist speziell auf relationale Datenbanken ausgelegt[29].
Der Unterschied zur `JPA` ist jener, dass JDBC das Protokoll ist, mit dem Java mit der Datenbank kommunizieren kann. `JDBC` wandelt die Ergebnisse der Queries, welche an die Datenbank gesendet wurden, in eine für Java nutzbare Form um.
Deswegen muss man in Java die `Datasource-URL` wiefolgt angeben:

```
jdbc:postgresql://localhost:3406/binobo_db
```

`JDBC` ist das dominierende Protokoll, durch welches mit der Datenbank kommuniziert wird, diesem nachfolgend steht `postgresql`, was Auskunft darüber gibt, dass es sich um eine PostgreSQL Datenbank handelt.

5.2.4. PostgreSQL

PostgreSQL ist ein frei nutzbares, objektrelationales Datenbankmanagementsystem (ORDBMS)[3]. Aufgrund dessen, dass PostgreSQl Open-Source ist, wurde dieses `ORDBMS` verwendet, um die Daten des Spring Boot Servers zu persistieren.
Weiters wird zur Verwanltung der PostgreSQL-Datenbankserver das Datenbankmanagement-Tool `pgAdmin 4` verwendet, welches ebenfalls Open-Source ist.

5.2.5. Datenbankstruktur und Relationen

In der nachstehenden Abbildung ist das ER - Diagram der PostgreSQL Datenbank zu sehen:

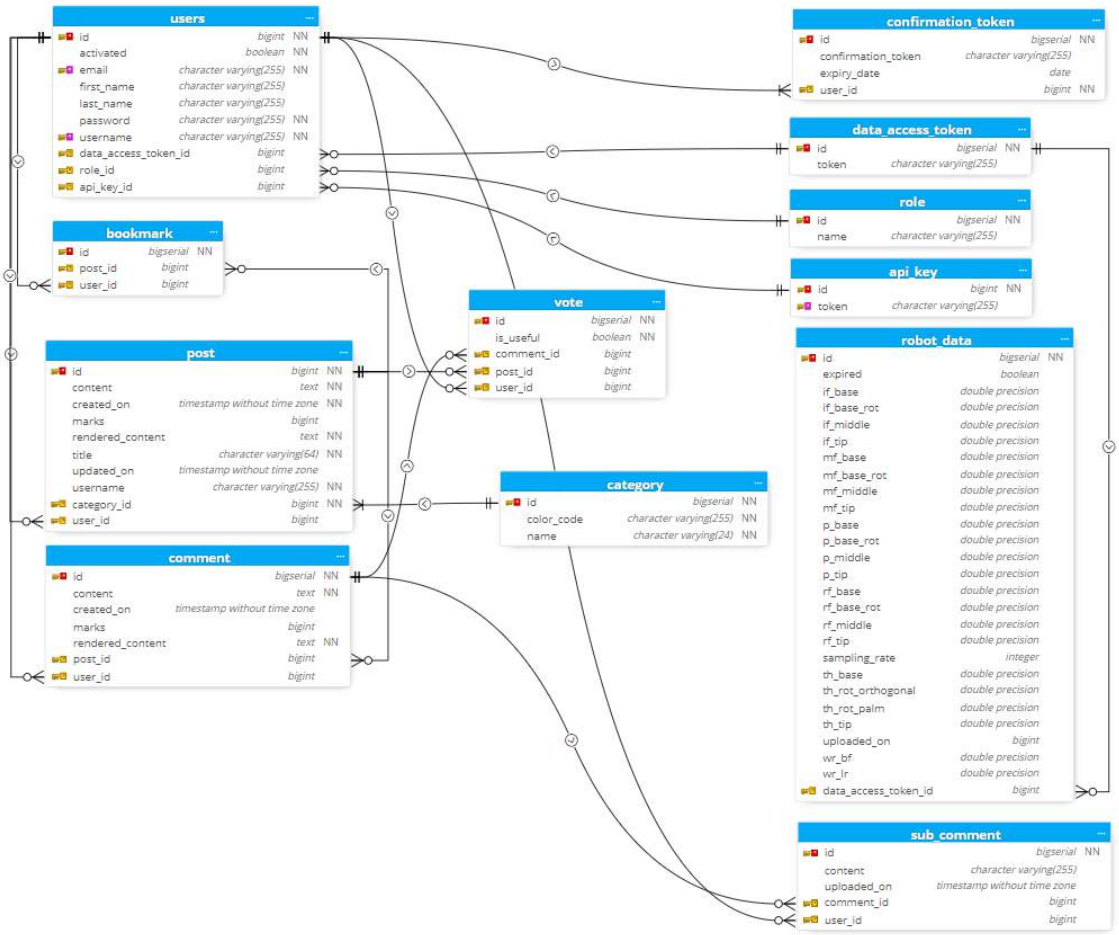


Abbildung 5.1.: ER - Diagram der Datenbank

Dieses ERD wurde mithilfe der Testversion von der Software `Moon Modeler`[52] erstellt.

5.3. Frontend

Das Frontend wurde mithilfe der dafür klassischen Markup-Sprache `HTML` oder `Hypertext Markup Language`, die Stylesheet-Sprache `CSS` oder `Cascading Style Sheets` und `JavaScript` entwickelt.
Als UI-Entwicklungstool zum erstellen der Blueprints der Website wurde `Figma` verwendet. Nachstehend ist der Link zu dem Figma-File zu sehen:

`https://www.figma.com/file/o7oOBzBj4csVCth27K6ms8/binobo?node-id=0%3A1`

Folgendes Grunddesign wurde für das Frontend entworfen:

Abbildung 5.2.: Web-UI

Auf diesem Layout basierend wurden die restlichen Seiten entworfen. Es wurde darauf geachtet, dass das Main-Layout eine fixe Größe hat und nur der Kontent, welcher innerhalb dessen ist, `scrollable` ist.

Folgende Funktionalitäten stellt das Frontend bereit:

- Homepage
- Projektübersicht
- Informationen über die Entwickler
- Login
- Blog
- Emulator
- Profilverwaltungsseite

Zur näheren Erläuterung der jeweiligen Seiten, siehe nachstehende Kapitel.

5.3.1. Template-Engine Thymeleaf

Jede der entworfenen HTML-Seiten wurde dynamisch gehalten, damit diese mithilfe von `Thymeleaf` entsprechend des übergebenen Models gerendert werden können[16]. Thymeleaf ist eine Template-Rendering Engine, welche mithilfe von Model-Daten gegebene HTML-Seiten dynamisch rendert. Spring Boot ermöglicht die Integration von Thymeleaf durch das einfache Einbinden eines Dependencies in der `pom.xml` Datei, welche von `Maven` zur Projektverwaltung verwnedet wird.

Thymeleaf kann innerhalb eines HTML-File durch das Einbinden des entsprechenden `XML-Namespaces` verwendet werden:

```
<html xmlns:th="http://www.thymeleaf.org">
```

Wie in diesem Beipiel ist der Namensraum typischerweise als Attribut des HTML-Tags `html` angeführt. Nach einbinden dieses Namensraums kann man auf die entsprechneden Funktionalitäten von Thymeleaf zugreifen, welche beim Rendern des Template von Thymeleaf erkannt werden und entsprechend Daten eingesetzt werden oder gegebene Anweisungen ausgeführt werden.
Beispielsweise kann man je nach Ergebnis einer bool'schen Vergleichsoperation eine Klasse an ein HTML-Element anfügen:

```
<div th:classappend="${current_user_vote_post == null ? '' : (!
    current_user_vote_post.isUseful ? 'checked' : '')}">
```

Der Thymeleaf-Syntax entspricht dem von Spring entworfenem `SpEL` - Syntax.

5.3.2. Webjars

`Webjars` sind in `JAR`-Files komplierte Javascript Webresourcen[35]. Hierdurch wird das Verwenden von etwaigen Javascript-Bibliotheken wesentlich vereinfacht.
Um Webjars verwenden zu können, muss folgende Dependencies in dem `pom.xml` eingebunden werden:

```
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>webjars-locator</artifactId>
  <version>0.30</version>
</dependency>
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>webjars-locator-core</artifactId>
</dependency>
```

Listing 5.13: Webjars - Maven Dependencies

Hierdurch können die benötigten Webresourcen durch mithilfe des gewählten Build und Dependency-Management Tools verwaltet werden. Folgende Javascript - Bibliotheken fanden in diesem Projekt Anwendung:

- `three.js` - vgl. Kapitel 5.4.1
- `editor.md` - vgl. Kapitel 5.5.1
- `bootstrap` - wurde jedoch nicht benutzt
- `jQuery` - benötigt für `AJAX`, vgl Kapitel 5.5.2

Analog dazu die benötigten Einträge im `pom.xml` File:

```
<dependency>
    <groupId>org.webjars.npm</groupId>
    <artifactId>super-three</artifactId>
    <version>0.111.6</version>
</dependency>
<dependency>
    <groupId>org.webjars.bowergithub.pandao</groupId>
    <artifactId>editor.md</artifactId>
    <version>1.5.0</version>
</dependency>
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>bootstrap</artifactId>
    <version>3.3.7</version>
</dependency>
```

```
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>jquery</artifactId>
    <version>3.1.1-1</version>
</dependency>
<dependency>
    <groupId>org.webjars.npm</groupId>
    <artifactId>three-orbit-controls</artifactId>
    <version>82.1.0</version>
</dependency>
```

Listing 5.14: Webjars - Javascript Libraries

Wenn diese Dependencies in dem `pom.xml` File eingebunden sind, kann auf diese in dem jeweiligen HTML-File folgendermaßen referenziert werden:

```
<script src="/webjars/editor.md/editormd.js"></script>
```

Dieses Listing zeigt anhand der Javascript-Bibliothek `editor.md` wie das einbinden solcher JAR-Webresourcen funktioniert.
Jedoch muss man, bevor der Webbrowser auf diese Resourcen zugreifen kann, diesen Pfad über eine Konfigurationsdatei bestimmen:

```
package htlstp.diplomarbeit.binobo.configurator;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.
    EnableWebMvc;
import org.springframework.web.servlet.config.annotation.
    ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.
    WebMvcConfigurer;

@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {

    private static final String[] CLASSPATH_RESOURCE_LOCATIONS = {
            "classpath:/META-INF/resources/", "classpath:/resources/
                ",
            "classpath:/static/", "classpath:/public/" };

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry
        ) {
        registry
                .addResourceHandler("/webjars/**", "/resources/**")
                .addResourceLocations("/webjars/", "/resources/");
        if (!registry.hasMappingForPattern("/**")) {
```

```
22                 registry.addResourceHandler("/**").addResourceLocations(
23                         CLASSPATH_RESOURCE_LOCATIONS);
24             }
25         }
26 }
```

Listing 5.15: WebMvcConfigurer - Implementierung

Hiermit aktiviert man den Resourcenpfad zu Webjars, und gewährleisten weiterhin den Standardpfad zu statischen Webresourcen.

5.4. Emulator

Zur Visualisierung der aufgenommenen Freiheitsgrade wird ein eigens in *Blender* kreiertes 3D-Modell auf der Website mit Hilfe der in Kapitel 5.4.1 genannten Library, `three.js`, dargestellt und den Messungen des Controllers entsprechend bewegt wird. Dieser 3D-Viewport funktioniert ohne Download innerhalb des Browsers auf den meisten Geräten und nutzt die lokale Hardware zur Darstellung von Grafiken. Innerhalb dieses Fensters kann sich der Nutzer frei bewegen und umschauen, um eine gute Ansicht für die aktuelle Handposition zu erlangen.
Um die echtzeitnahe Emulation dieser Werte gewährleisten zu können, wurde ein Websocketserver mithilfe der Skriptsprache `Python` programmiert, nähere Informationen zu Websockets und dem Websocketserver sind in Kapitel 5.4.2 & 6 zu finden.

Der entwickelte Sourcecode des Emulators ist im Anhang zu finden in Listing A.2.

5.4.1. three.js

`three.js`[9] ist eine JavaScript-Library, welche dazu genutzt werden kann, echtzeit-3D-Anwendungen in JavaScript zu programmieren. Sie bietet eine Alternative zu Browser-3D-Engines, wie zum Beispiel `Unity WebGL`.
Die Library ist frei nutzbar und auf threejs.org erhältlich. Sie ist für ein großes Spektrum an Anwendungen geeignet, von dreidimensionalen oder animierten Websitehintergründen bis hin zu 3D-Browserspielen oder sogar Virtual-Reality Anwendungen.

5.4.2. Websockets

Um eine echtzeitnahe Emulation zu ermöglichen, ist eine Websocketverbindung unabdingbar. Denn durch solche Verbindungen ist es möglich, nur Daten ohne HTTP-Request-Header zu versenden, da sicher der Server in der Mitter zweier Clienten merkt, wer verbunden ist und akzeptiert jede Art von eingehenden Daten und leitet

diese je nach Bedarf weiter.
Weiters ist durch die Verwendung von Websockets eine bidirektionale Kommunikation zwischen Server und Client möglich.

Javascript bietet vorgefertigte Objekte um mit Websockets arbeiten zu können: Die `Websocket` Klasse. Folgendes Listing (5.16) zeigt die `connect-Funktion` des Emulator-Sourcecodes:

```
1  function connect(node) {
2    node.onclick = () => disconnect(node);
3    node.classList.add("connected");
4
5    socket_data_receiver = new WebSocket('wss://emulator.binobo.io/' +
         token);
6    socket_ping = new WebSocket('wss://emulator.binobo.io/');
7    socket_ping_receiver = new WebSocket('wss://emulator.binobo.io/
        ping_' + token);
8
9    socket_ping_receiver.addEventListener('message', ({ data }) => {
10       let x = new Date();
11       let ping = x.getTime() - eval(data);
12       let node = document.getElementById("ping")
13       node.innerHTML = ping + "ms"
14   });
15
16   socket_ping.addEventListener('open', evt => {
17       socket_ping_interval = setInterval(() => {
18           socket_ping.send('["ping_' + token + '",' + (new Date()).
                getTime() + "]");
19       }, 10000);
20       setTimeout(() => {
21           socket_ping.send('["ping_' + token + '",' + (new Date()).
                getTime() + "]");
22       }, 1000);
23   });
24
25   socket_ping_receiver.addEventListener('open', evt => {
26       socket_ping_receiver.send('ping␣receiver␣for:␣' + token)
27   });
28
29   socket_data_receiver.addEventListener('message', ({data}) => {
30       try {
31           for(i of eval(data)){
32               data_buffer[data_buffer.length] = i;
33           }
34       } catch (e) {
35           console.log('wrong␣data-format␣received:␣' + data)
36       }
```

```
37    });
38    socket_data_receiver.addEventListener('open', evt => {
39        socket.send('receiver for: ' + token)
40    })
41    document.getElementById("connect").innerHTML = "stop";
42  }
```

Listing 5.16: Webbrowser - Websocketclient

In dieser Funktion werden drei Websocketverbindungen aufgebaut:

1. `socket_data_receiver` - Empfängt und verarbeitet die eingehenden Daten
2. `socket_ping` - Ping-Sender
3. `socket_ping_receiver` - Ping-Empfänger

Die Websocketverbindung für das Pingen musste aufgrund der Websocketserverstruktur auf einen Ping-Sender und einen Ping-Empfänger Socket aufgeteilt werden, vgl. Kapitel 6. Gemessen wird das Ping alle 10 Sekunden und funktioniert folgendermaßen:

- `socket_ping` sendet dem Server die aktuelle Zeit in Milisekunden
- Der Server empfängt diesen Timestamp und leiten dieses weiter zu dem dazugehörigen Client, gekennzeichnet durch einen Token
- `socket_ping_receiver` empfängt diesen Timestamp und subtrahiert diesen von der aktuellen Zeit, die Differenz gibt an, wie lange es dauert, Daten zu dem Server zu schicken und wieder zu empfangen

Durch diese Methodik kann man ungefähr auf die tatsächliche Latenz rückschließen, welche von der Datenerfassung des ESP32 bis hin zum Empfangen und Anzeigen der Daten am Server auftreten wird.

Das Empfangen der Daten wird durch sogenannte `Eventlistener` ermöglicht, welche asynchron und parallel zueinander arbeiten.

Der `socket_data_receiver`-Websocket empfängt die Daten immer in folgendem Format:

```
data = [[double, double, double, double, double, double, double,
    double, double, double, double, double, double, double, double,
     double, double, double, double, double, double], [...], ...]
```

Ein zweidimensionales Array, bestehend aus 22 - double-precission-Werten. Jeder Double-Wert bildet genau ein Gelenk ab.

5.4.3. IIR - Filter

Da die gemessenen Werte ein dezentes Rauschen aufwiesen, wurde ein sogenannter `Infinite Inpulse Response Filter`[47] - Algorithmus auf der Website implementiert. Dieser Filteralgorithmus ist rekursiv und benötigt zum filtern einen frei wählbaren Dämpfungsgrad w und den vorherigen gefilterten Wert (daher rekursiv):

$$y_n = w \cdot x_n + (1 - w) \cdot y_{n-1}$$

Der gefilterte Wert, gekennzeichnet durch y_n wird berechnet, indem man den aktuellen Messwert x mit dem Dämpfungsgrad w multipliziert. Dann wird der letzte gefilterte Wert y_{n-1} mit $1 - w$ multipliziert und zu $x \cdot w$ addiert. Die Summe ergibt den neuen Wert y_n.
Dieser Filter heißt `exponentieller Filter`[48] und wurde gewählt, da dieser wenig rechenaufwändig ist.

Nachstehendes Listing (5.17) zeigt die Funktion, welche diesen Filter anwendet:

```
1  function apply_filter_to_data(data_to_filter){
2    let w = 0.6;
3    let w_m1 = 1 - w;
4    let filtered_data = [];
5
6    for(let i = 0; i<data_to_filter.length; i++){
7        filtered_data[i] = w * data_to_filter[i] + w_m1 *
             last_filtered_data[i]
8    }
9    last_filtered_data = filtered_data;
10   return filtered_data;
11 }
```

Listing 5.17: IIR-Filter Implementierung

5.5. Blog

In Kapitel 4.1 wurde kurz erwähnt, dass den registrierten Nutzern ein Blog zur Verfügung gestellt wird. Zur Erklärung der relationalen Datenstruktur, siehe Kapitel 5.2.5.
Dieser Blog bietet folgende Funktionalitäten:

- Blogeinträge erstellen, bearbeiten und löschen
- Blogeinträge speichern

- Blogeinträge mit Votes beaufschlagen
- Kommentare schreiben und editieren
- Kommentare mit Votes beaufschlagen

Einige dieser Funktionalitäten wurden mittels `REST-Calls` realisiert, für nähere Informationen hierzu, vergleiche Kapitle 5.5.2.

5.5.1. editor.md

Um Blogeinträge erstellen zu können, wurde auf den opens-ource `Markdown` - Editor von `editor.md`[36] zurückgegriffen. Dieser bietet die Möglichkeit, mithilfe der Auszeichnungssprache `Markdown` `HTML`-Elemente zu erstellen.

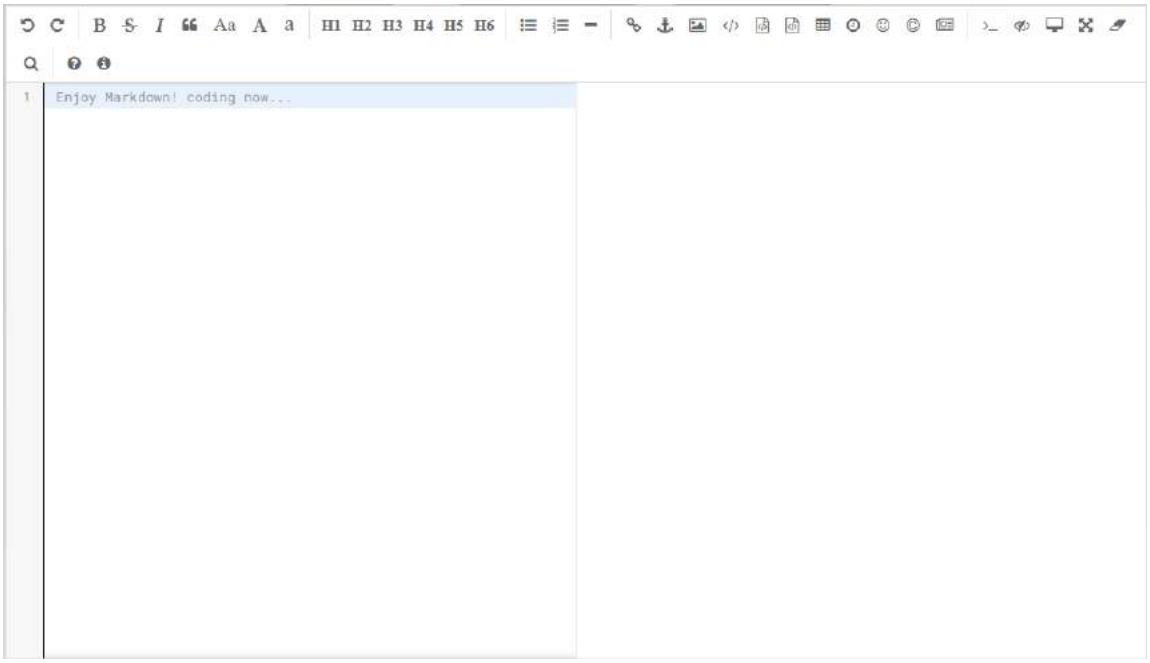


Abbildung 5.3.: Markdowneditor `editor.md`

Um diesen Editor auf der Website anzeigen zu lassen, muss folgender Javascript-Code ausgeführt werden:

```
1 const new_post = editormd("new_post", {
2     width                 : "100%",
3     height                : "100%",
4     path                  : "/webjars/editor.md/lib/",
```

```
5       theme                   : localStorage.getItem("data-theme") === "
            dark" ? "dark" : "default",
6       editorTheme             : localStorage.getItem("data-theme") === "
            dark" ? "monokai" : "solarized",
7       previewTheme            : localStorage.getItem("data-theme") === "
            dark" ? "dark" : "default",
8       saveHTMLToTextarea      : true,
9       emoji                   : true,
10      tex                     : true,
11      taskList                : true,
12      flowChart               : true,
13      sequenceDiagram         : true,
14      tocm                    : true,
15      tocDropdown             : true,
16      readOnly                : false,
17      markdown                : document.querySelector('#raw_content').
            value !== '' ? document.querySelector('#raw_content').value :
             "## Welcome to our small and geeky network! Enjoy it!",
18      onchange                : function () {
19          document.querySelector('#raw_content').value = this.
                getMarkdown();
20          document.querySelector('#rendered_content').value = this.
                getHTML();
21      }
22  });
```

Listing 5.18: editor.md - Deklaration

Dieses Listing (5.18) zeigt einen Ausschnit aus dem HTML-Dokument `src/main/resources/templates/blogForm.html`. Hier wird ein neuer Editor erstellt, welcher benötigt wird, um einen neuen Blogeintrag zu erstellen.

5.5.2. API Key und REST

In Kapitel 5.5 wuren kurz erwähnt, dass für einige der Funktionalitäten, welche dieser Blog bietet, asynchrone `REST-Calls` gemacht werden müssen. `REST` steht für `Representational State Transfer` und ist im Grunde das HTTP-Protokoll, nur wird `REST` für Datenaustausch verwendet, und HTTP ist die Protokolldefinition für den Datenaustausch generell über das Internet.

Spring Boot, als auch Javascript bieten bereits etliche APIs um mit RESTful Services zu arbeiten. Bei Spring ist das Erstellen eines Rest-Controllers sehr ähnlich dem Erstellen eines normalen HTTP-Controllers: Die entsprechende Javaklasse muss mit `@RestController` annotiert werden. Folgendes Listing (5.19) zeigt den erstellten Rest-Controller, welcher entwaige Funktionalitäten für den Blog bereitstellt:

```
1  package htlstp.diplomarbeit.binobo.controller.restController;
2
3  import htlstp.diplomarbeit.binobo.controller.util.FlashMessage;
4  import htlstp.diplomarbeit.binobo.model.*;
5  import htlstp.diplomarbeit.binobo.service.*;
6  import org.springframework.beans.factory.annotation.Autowired;
7  import org.springframework.http.ResponseEntity;
8  import org.springframework.web.bind.annotation.*;
9
10 import java.util.Objects;
11 import java.util.concurrent.atomic.AtomicBoolean;
12
13 @RestController
14 @ResponseBody
15 @RequestMapping(value = "/blog_rest_api")
16 public class BlogRestAPI {
17
18     private final PostService postService;
19     private final UserService userService;
20     private final CommentService commentService;
21     private final SubCommentService subCommentService;
22     private final CategoryService categoryService;
23     private final BookmarkService bookmarkService;
24     private final VoteService voteService;
25
26     @Autowired
27     public BlogRestAPI(PostService postService,
28                        UserService userService,
29                        CommentService commentService,
30                        SubCommentService subCommentService,
31                        CategoryService categoryService,
32                        BookmarkService bookmarkService,
33                        VoteService voteService) {
34         this.postService = postService;
35         this.userService = userService;
36         this.commentService = commentService;
37         this.subCommentService = subCommentService;
38         this.categoryService = categoryService;
39         this.bookmarkService = bookmarkService;
40         this.voteService = voteService;
41     }
42
43     @PatchMapping(value = "/toggle_bookmark")
44     public ResponseEntity<FlashMessage> toggleBookmark(@RequestParam
           ("post_id") Long post_id, @RequestParam("user_id") Long
           user_id, @RequestParam("api_key") String token){
45         User user = userService.findById(user_id);
46
47         if(Objects.equals(user.getApi_key().getToken(), token)) {
```

```
48              Iterable<Bookmark> bookmarks = bookmarkService.
                    fetchAllBookmarksFromUser(user);
49              AtomicBoolean bookmark_exists = new AtomicBoolean(false)
                    ;
50              bookmarks.forEach(bookmark -> {
51                  if (Objects.equals(bookmark.getPost().getId(),
                        post_id)){
52                      bookmarkService.deleteBookmark(bookmark);
53                      bookmark_exists.set(true);
54                  }
55              });
56              if (!bookmark_exists.get()) {
57                  Bookmark bookmark = new Bookmark();
58                  bookmark.setPost(postService.findById(post_id));
59                  bookmark.setUser(user);
60                  bookmarkService.saveBookmark(bookmark);
61              }
62
63              return ResponseEntity.ok().body(
64                      new FlashMessage("Bookmark toggled!",
65                              FlashMessage.Status.SUCCESS));
66          }else {
67              return ResponseEntity.badRequest().body(
68                      new FlashMessage("Invalid API Key!",
69                              FlashMessage.Status.FAILURE));
70          }
71      }
72
73      // upvote post, down-vote post, remove vote from post
74
75      @PatchMapping(value = "/toggle_vote_for_post")
76      public ResponseEntity<FlashMessage> toggleVotePost(@RequestParam
            ("post_id") Long post_id, @RequestParam("user_id") Long
            user_id, @RequestParam("action") String action, @RequestParam
            ("api_key") String token) {
77          User user = userService.findById(user_id);
78          if(Objects.equals(user.getApi_key().getToken(), token)) {
79              Post post = postService.findById(post_id);
80              Vote vote = voteService.findByUserAndPost(user, post);
81
82              if (vote == null) {
83                  Vote newVote = new Vote();
84                  newVote.setIsUseful(Objects.equals(action, "up"));
85
86                  newVote.setUser(user);
87                  newVote.setPost(post);
88
89                  voteService.save(newVote);
90              } else {
91                  switch (action) {
```

```
92                      case "up": {
93                          if (vote.getIsUseful()) {
94                              vote.setPost(null);
95                              vote.setUser(null);
96                              // this is needed to disassociate the
                                    vote from the user and post
97                          } else {
98                              vote.setIsUseful(true);
99                          }
100                     } break;
101                     case "down": {
102                         if (!vote.getIsUseful()) {
103                             vote.setPost(null);
104                             vote.setUser(null);
105                         } else {
106                             vote.setIsUseful(false);
107                         }
108                     } break;
109                 }
110                 voteService.save(vote);
111                 if(vote.getUser() == null && vote.getPost() == null)
                        voteService.delete(vote);
112             }
113             return ResponseEntity.ok().body(
114                     new FlashMessage("Set Vote",
115                             FlashMessage.Status.SUCCESS));
116         }
117         return ResponseEntity.badRequest().body(
118                 new FlashMessage("Invalid API Key!",
119                         FlashMessage.Status.FAILURE));
120     }
121 }
```

Listing 5.19: Blog - REST API

Zu sehen ist, dass diese Klasse mit `@RequestMapping(value = /blog_rest_api")` und `@ResponseBody` annotiert ist. Die Annotation `@RequestMapping` setzt einen URI-Prefix für alle in dieser Klasse definierten HTTP-Endpoints, der Prefix wurde durch `value = /blog_rest_api"` definiert. Die zweite Annoation `@ResponsBody` gibt an, dass die Funktionen dieser Klasse einen Responsebody zurücksenden als Antwort.

Die erste Funktion `toggleBookmark` erstellt oder löscht ein Lesezeichen, welches von einem Nutzer bei einem Blogeintrag gesetzt wurde.
Die zweite Funktion in dieser Klasse `toggleVotePost` erstellt oder löscht ein sogenanntes `Vote` eines Blogeintrags. Anhand einses Votes kann man angeben, ob ein Blogeintrag nützlich ist, oder nicht. Eine ähnliche Funktion wurde auch für Kommentare erstellt, jedoch wurde diese in diesem Listing nicht eingefügt, da ansonsten

das Listing zu lange wäre.

Im Funktionskopf bei der Parameterdeklaration kann man eine weitere Annotation erkennen `@RequestParam`, diese gibt an, dass ein in der Request-URL stehender Parameter an diesen Parameter beim Aufruf übergeben werden soll. Durch die Übergabe eines Parameters an die Annotation kann man den Namen des URL-Parameters definieren.

API Key

Der `API Key` ist eine 128-Bit `UUID` (Universally Unique Identifier). Jeder der registrierten Nutzer bekommt jedes mal wenn sich dieser einloggt, einen neuen zufällig generierten Schlüssel zugewiesen, um größtmögliche Sicherheit der API gewährleisten zu können.

Javascript AJAX

`AJAX`[37] steht für `Asynchronous JavaScript and XML` und wird einerseits für asynchrone REST-Calls verwendet. Um vom Browser aus mit der Blog-API kommunizieren zu können, wurde folgende Javascript-Funktion entwickelt:

```
1  const voteComment = (node, comment_id, user_id, action) => {
2    let api_key = document.getElementById("api_key").valueOf().value;
3    let url = `${window.location.protocol}//${window.location.hostname
       }/blog_rest_api/toggle_vote_for_comment?comment_id=${comment_id
       }&user_id=${user_id}&api_key=${api_key}&action=${action}`;
4    $.ajax({
5        type:"Patch",
6        url:url,
7        success: function(response) {
8            let isUpVoted = document.getElementById('comment_upvote_'
                + comment_id).classList.contains('checked');
9            let isDownVoted = document.getElementById('comment_devote_
                ' + comment_id).classList.contains('checked');
10           let comment_vote_counter = document.getElementById('
                comment_vote_count_' + comment_id);
11
12           let upvoteNode = document.getElementById('comment_upvote_'
                + comment_id);
13           let devoteNode = document.getElementById('comment_devote_'
                + comment_id);
14
15           if(isUpVoted && action === 'up') {
16               comment_vote_counter.innerHTML = (Number.parseInt(
                     comment_vote_counter.innerHTML) - 1) + "";
17               upvoteNode.classList.remove('checked');
18
19           } else if(isUpVoted && action === 'down') {
```

```
20                comment_vote_counter.innerHTML = (Number.parseInt(
                      comment_vote_counter.innerHTML) - 2) + "";
21                node.classList.remove('checked');
22                upvoteNode.classList.remove('checked');
23                devoteNode.classList.add('checked');
24
25            } else if(isDownVoted && action === 'up') {
26                comment_vote_counter.innerHTML = (Number.parseInt(
                      comment_vote_counter.innerHTML) + 2) + "";
27                upvoteNode.classList.add('checked');
28                devoteNode.classList.remove('checked');
29
30            }else if(isDownVoted && action === 'down') {
31                comment_vote_counter.innerHTML = (Number.parseInt(
                      comment_vote_counter.innerHTML) + 1) + "";
32                devoteNode.classList.remove('checked');
33
34            } else if(!isDownVoted && !isUpVoted && action === 'down')
                  {
35                comment_vote_counter.innerHTML = (Number.parseInt(
                      comment_vote_counter.innerHTML) - 1) + "";
36                devoteNode.classList.add('checked');
37
38            } else if(!isDownVoted && !isUpVoted && action === 'up') {
39                comment_vote_counter.innerHTML = (Number.parseInt(
                      comment_vote_counter.innerHTML) + 1) + "";
40                upvoteNode.classList.add('checked');
41            }
42        }
43    });
44 };
```

Listing 5.20: AJAX Implementierung

Listing 5.20 zeigt die Implementierung der AJAX-API.
Die gezeigte `Pointer-Funktion voteComment` macht einen `Patch-Request` an den Server, welcher je nach aktuellen Standt des Votes an diesem Kommentar wird dieses entweder gelöscht, positiv oder negativ gewertet. Sobald der Response von dem Server kommt, wird in der Funktion `success` der AJAX-API je nach aktuellen Stand des Votes dieses abgeändert.

6. Python Websocket-Server

Websockets sind eine Erweiterung von HTTP, welche es ermöglicht, aufgebaute Verbindungen beliebig lage aufrecht zu erhalten. Um des Server mitzuteilen, dass man seine Verbindung `upgraden` will, wird folgender Request - Header an den Server geschickt[38]:

```
GET / HTTP/1.1
Host: emuesp32.binobo.io
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: /
Sec-WebSocket-Version: 13
```

Listing 6.1: HTTP-Header für Websocket-Upgrade

Wenn dieser Request akzeptiert wurde, kommt folgender Response:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: /
```

Listing 6.2: Respone um auf Websockets-Protokoll zu wechseln

Der Python Websocketserver diente als Schnittstelle zwischen dem Browser und dem ESP32. Dieser ermöglicht es, eine Websocketverbindungen zwischen diesen beiden Clienten aufzubauen.
Dieser Server wurde benötigt, um einen möglichst echtzeitnahen Datentransfer zu ermöglichen.

Dieser Server wurde mithilfe der Python-Library `websockets`[2] entwickelt.
So wie die Serversoftware, ist auch dieser Teil des Projekts als `open-source` - Software auf Github zu finden:

`https://github.com/psykovski-extended/binobo_websocket_server.git`

Python wurde verwendet, um den den Datenaustausch zwischen dem ESP32 um dem Browser so unkompliziert wie möglich zu gestalten. Aufgrund dessen, das die Programmiersprache von Client und Server die selbe ist, weiß man genau in welchen Format die Daten ankomment und kann diese unkompliziert weiterverarbeiten.

6.1. Sourcecode

Nachstehendes Listing (6.3) zeigt der Sourcecode des Webservers:

```
1  import asyncio
2  import websockets
3  import socket
4
5  receiver_clients = {}
6
7
8  async def handle_receiver_clients(ws_client, message):
9      token = ws_client.path.split('/')[-1]
10     if message == "close session":
11         for client_index in range(len(receiver_clients[token])):
12             if ws_client == receiver_clients[token][client_index]:
13                 del receiver_clients[token][client_index]
14                 break
15     else:
16         if token in receiver_clients.keys():
17             receiver_clients[token].append(ws_client)
18         else:
19             receiver_clients[token] = [ws_client]
20         await ws_client.send(str({'STATE': 'OK'}))
21     print(receiver_clients)
22
23
24 async def retrieve_data(websocket, path):
25     if websocket.path != '/':
26         async for message in websocket:
27             await handle_receiver_clients(websocket, message)
28     else:
29         async for message in websocket:
30             data = eval(message)
31             try:
32                 token = data[0]
33                 robo_data = data[1]
34                 clients_to_delete = []
35                 for client_index in range(len(receiver_clients[token
                       ])):
36                     try:
```

```
37                          await receiver_clients[token][client_index].
                                send(str(robo_data))
38                      except Exception as exc:
39                          print('error occurred: ' + str(exc))
40                          clients_to_delete.append([client_index])
41                  clients_to_delete.reverse()
42                  for i in clients_to_delete:
43                      del receiver_clients[token][i]
44              except Exception as exc:
45                  await websocket.send(str({'Error': str(exc)}))
46
47
48  async def main():
49      sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
50      sock.connect(("8.8.8.8", 80))
51      ip = sock.getsockname()[0]
52      print("Websocket Server running on:\n" + ip + ":8080")
53      async with websockets.serve(retrieve_data, ip, 8080,
            ping_interval=None, ping_timeout=None):
54          await asyncio.Future()  # run forever
55
56
57  asyncio.run(main())
```

Listing 6.3: Websocketserver Sourcecode

In Listing 6.3 erkennt man von Zeile 1 bis 3 folgende Imports:

1. `asyncio`[39] - Aktiviert die Unterstützung von Asynchronen Tasks
2. `websockets`[2] - Benötigt um den Websocketserver aufzusetzten
3. `socket`[40] - Benötigt um die Lokale-IP Adresse herauszufinden

Die Python-Library `websockets` unterstützt asynchrones Programmieren, weswegen `asyncio` importiert wurde. Hierdurch wird die Softwarearchitektur `non-blocking` und `event-driven`. Dies ist hilfreich, sollten mehrere Clienten verbunden sein.

Auf Zeile 5 erkennt man die Deklaration des Python-Dictionaries `receiver_clients`. Diese Variable wird benötigt, um die entsprechenden Empfänger je nach Token zu sortieren, um diese dann einfacher dem ESP32 Client zuordnen zu können.

Die Funktion `handle_receiver_clients` fügt diesem Buffer neue Clienten hinzu oder löscht diese, wenn der entsprechende Nutzer die Verbindung trennen will. Ebenfalls ist auf Zeile `17` & `19` erkennbar, dass nicht bloß der Client dem Buffer hinzugefügt wird, sondern dass eine `Python-Liste` aus Clienten angefügt wird. Dies wird gemacht, um zu ermöglichen, dass mehrere Clienten mit einem ESP32 verbunden

sein können.
Sollte ein Client eine Nachricht mit `"close session"` als Inhalt dem Websocketserver senden, wird die Referenz dieses Clients gelöscht (Zeile 13) und die Verbindung geschlossen.

Damit die Software erkennt, welcher Client ein Empfänger ist, wurde folgendes definiert:

Jeder Client, dessen `Request-URI` ungleich der `Root-URI /` ist, wird als Empfänger eingestuft, der entsprechende Token soll als letzter Unterpfad der URL angegeben werden.

Die `if-Abfrage` auf Zeile 25 realisiert diese Definition.

Sollten jedoch Daten von einem Sender-Client kommen (dem ESP32, ab Zeile 28), so werden diese Daten entsprechend verarbeitet. Zuerst wird der empfangene `String` mithilfe der Python-Funktion `eval` (Zeile 30) evaluiert, der Rückgabewert von diesem funktionsaufruf is eine Python-Liste mit folgendem Datenformat:

```
data = [128-Bit UUID, [[double, double, double, double, double,
    double, double, double, double, double, double, double, double,
     double, double, double, double, double, double, double],
    [...], ...]]
```

Listing 6.4: Datenübertragungsprotokoll - Datenformat

Zeile 32 & 33 filtern zuerst den Token aus dieser Liste heraus und dann die Daten zum weiterleiten.

Auf Zeile 34 wird eine neue Liste definiert: `clients_to_delete`. Diese Liste wird benötigt, um die Indizes derer Client zwischenzuspeichern, bei welchen beim senden der Daten ein Fehler aufgetreten ist. Da hier dann davon ausgegangen wird, dass die Verbindung nicht mehr aufrecht ist oder ein anderes Problem aufgetreten ist, wird diese Referenz gelöscht, Zeile 42 & 43.

Da durch das Löschen aus dieser Liste sich die jeweiligen Indizes der anderen gespeicherten Referenzen verändern würden (da sich die Größe der Liste ändert), wird auf Zeile 41 die Liste `clients_to_delete` einfach ungekehrt. Da sich die Indizes nur der darüberliegenden Elemente verändert.

Auf Zeile 53 wird der Websocketserver gestartet mit der Lokalen-IP Adresse des aktuellen Rechners. Die IP-Adresse wird auf Zeile 49 - 51 ermittelt.

7. Micropython Firmware

`Micropython`[7] ist ein CPython-Derivat, welches speziell für das Programmieren auf μ`Cs` entwickelt wurde. Micropython ist ebenfalls als open-source Software, lizensiert unter der MIT-Lizenz, frei im Internet erhältlich.

Als Referenz und Nachschlagewerk gibt es im Internet, neben dem Github-Repository, eine offizielle Micropython-Homepage, auf welcher ausführlich erklärt wird, wie die jeweilgen Libraries aufgebaut sind und wie man diesem am besten anwendet:

`https://micropython.org`

Nachstehender Link verweist auf das entsprechende Github-Repository der für dieses Projekt entwickelten Firmware:

`https://github.com/psykovski-extended/binobo_firmware`

7.1. Sourcecode

Nachstehende Listings zeigen den Sourcecode der Firmware, aufgeteilt in kleinere Portionen, um Übersichtlichkeit zu wahren:

```
1  import time
2  import network
3  import uwebsockets.client
4  import urequests
5  import _thread
6  import ujson
7  import os
8
9  try:
10     import asyncio as uasyncio
11 except ImportError:
12     import uasyncio
13 from machine import ADC, Pin, Timer
```

Listing 7.1: Firmware: Imports

Zu sehen sind hier (Listing 7.1) die verwendeten `Micropython-Libraries`, sowie die Drittanbieter-Library `uwebsockets`[41]. Nachstehender Link verweist auf das originale Github-Repository des Entwicklers dieser Library:

`https://github.com/danni/uwebsockets.git`

Diese Library ist als open-source Software unter der MIT-Lizenz verfügbar.

```
1  class Multiplexer:
2    def __init__(self, pin1, pin2, pin3, pin4, a_readable: ADC,
         a_channels, enable: Pin):
3        """
4        Represents the CD74HCT4067 16 Channel Analogue Multiplexer,
             with 4 digital pins to select the analogue channel
5        to be read.
6
7        :param pin1: First digital Pin, should not be null.
8        :param pin2: Second digital Pin, can be null.
9        :param pin3: Third digital Pin, can be null.
10       :param pin4: Fourth digital Pin, can be null.
11       :param a_readable: Signal from multiplexed channel
12       :param a_channels: amount of analogue channels that are
             connected to the board
13       :param enable: Enable-Pin
14       """
15       self.pins = []
16
17       if pin1 is not None:
18           self.pins.append(pin1)
19       if pin2 is not None:
20           self.pins.append(pin2)
21       if pin3 is not None:
22           self.pins.append(pin3)
23       if pin4 is not None:
24           self.pins.append(pin4)
25
26       self.enable = enable  # inverted
27       self.a_readable = a_readable
28       self.a_readable.atten(ADC.ATTN_11DB)
29       self.a_readable.width(ADC.WIDTH_12BIT)
30
31       self.len_pins = len(self.pins)
32
33       self.a_channels = a_channels
34
35       self.enable_()
36
37   def enable_(self):
```

```
38          """
39          Enables the board
40          """
41          self.enable.off()
42
43      def disable(self):
44          """
45          Disables the board
46          """
47          self.enable.on()
48
49      def map_nibble_to_pins(self, nibble):
50          """
51          Maps a given 4-bit value to the multiplexer-pins
52
53          :param nibble: Nibble to be mapped --> f.e.: 0b0101 - S3 = 0,
                S2 = 1, S1 = 0, S0 = 1
54          """
55          if nibble > 15:
56              raise ValueError(
57                  "There are not more than 16 channels per Multiplexer
                        (0 to 15)! Pared value: " + str(nibble))
58
59          for i in range(self.len_pins):
60              self.pins[i].value(nibble & 0b0001)
61              nibble >>= 1
62
63      def read_all(self):
64          """
65          Reads all channels connected to the multiplexer
66          :return: Returns an array of the retrieved values, in raw
                format
67          """
68          res = []
69          for i in range(self.a_channels):
70              self.map_nibble_to_pins(i)
71              res.append(self.a_readable.read())
72          return res
73
74      def read_one(self, ch):
75          """
76          Reads one channel connected to the multiplexer and returns teh
                retrieved value
77          :param ch: Channel to be read --> channel counting begins at
                0!
78          :return: returns the retrieved value
79          """
80          self.map_nibble_to_pins(ch)
81          res = self.a_readable.read()
82          return res
```

```
83
84
85  class ADCIter:
86    def __init__(self, *multiplexer: Multiplexer):
87        """
88        Creates a kind-of Iterator, where the objects itself is
              iterating over the parsed multiplexers, so there is no
89        need to iterate over it with calling iter(obj)
90        :param multiplexer: Multiplexer-Objects to iterate over
91        """
92        self.multiplexer = multiplexer
93        self.a_ch = []
94        for i in self.multiplexer:
95            self.a_ch.append(i.a_channels)
96
97    def retrieve_data_raw(self):
98        """
99        Reads all analogue channels and converts the result to a 1D-
              List, where the items are ordered by the order the
100       of the parsed multiplexer and chronologically by the number of
              the pins on the multiplexer board
101       :return: 1D-List of the retrieved analogue values
102       """
103       res = []
104       for multi in self.multiplexer:
105           for i in multi.read_all():
106               res.append(i)
107       return res
```

Listing 7.2: Firmware: Multiplexer- und Iterator-Objekt

Da sich Listing 7.2 durch Kommentare selbst erklärt, wir an dieser Stelle auf diese verwiesen.

```
1  # data buffer
2  data = []
3  # vector, storing zero position
4  zero_pos = []
5  # indicates that the iteration for data-retrieving is done or still
       ongoing
6  iteration_done = False
7  # token to identify the connected user
8  token = ""
9  # wifi-ssid
10 ssid = ""
11 # wifi-password
12 password = ""
13 # websocket client object
14 ws = None
15 # mechanical degree of freedom
```

```
16 mdof = 340
17 # factor for data conversion
18 f_rc = mdof / 4095
19 # multiplexer 1
20 multi1 = Multiplexer(Pin(25, Pin.OUT), Pin(33, Pin.OUT), Pin(32, Pin
       .OUT), Pin(12, Pin.OUT), ADC(Pin(34)), 12,
21                       Pin(26, Pin.OUT))
22 # multiplexer 2
23 multi2 = Multiplexer(Pin(23, Pin.OUT), Pin(22, Pin.OUT), Pin(21, Pin
       .OUT), None, ADC(Pin(35)), 10, Pin(5, Pin.OUT))
24 # ADCIter obj
25 adc_iter = ADCIter(multi1, multi2)
```

Listing 7.3: Firmware: Globale Variablen

In Listing 7.3 sind die global deklarierten Variablen und Objekte zu sehen.

```
1  def connect_to_wifi(ssid="", pw=""):
2    """
3    Tries to establish a connection to a wifi
4    :param ssid: SSID of the wifi to connect to
5    :param pw: Password of the wifi to connect to
6    :return: returns the wrapper object of the connection
7    """
8    sta_if = network.WLAN(network.STA_IF)
9    sta_if.active(True)
10
11   sta_if.connect(ssid, pw)
12   sta_if.config(reconnects=2)
13
14   while not sta_if.isconnected():
15       pass
16
17   if not sta_if.isconnected():
18       raise Exception("Could␣not␣connect␣to␣WIFI!")
19
20   return sta_if
```

Listing 7.4: Firmware: WIFI-Connect Routine

Listing 7.4 zeigt die Funktion `connect_to_wifi`, welche versucht eine Verbindung mit einem Netzwerk zu verbinden, mithilfe des Parameters `ssid` gibt man den Namen des gewünschten Netzwerks an, und mit `password` gibt man das Passwort dieses Netzwerks an.
Danach probiert diese Funktion sich zweimal mit dem Netzwerk zu verbinden, wenn beide Versuche scheitern, wird ein Fehler geworfen. Sollte das Verbindungaufbauen erfolgreich sein, wird eine Referenz auf diese Verbindung zurückgegeben.

```
1 def convert_retrieved_data(data_to_conv: list):
```

```
2     """
3     converts the retrieved data from an analogue value to degrees
4     :param data_to_conv: Data to be converted
5     :return:
6     """
7     for i in range(22):
8         data_to_conv[i] = data_to_conv[i] * f_rc - zero_pos[i]
9     return data_to_conv
```

Listing 7.5: Firmware: Datenkonvertierung

Die Funktion `convert_retrieved_data` aus Listing 7.5 konvertiert die aufgenommenen Werte zu Grad und gibt die Liste mit den neuen Werten zurück.

```
1  def retrieve_data():
2      """
3      retrieves data from all connected multiplexer boards and appends
           it to the data-buffer
4      """
5      global iteration_done, data
6      iteration_done = False
7      data.append(convert_retrieved_data(adc_iter.retrieve_data_raw()))
8      iteration_done = True
9
10     if len(data) > 10:
11         data = []
```

Listing 7.6: Firmware: Datenerfassung

In der Funktion `retrieve_data` aus Listing 7.6 werden asynchron alle Spannungen, welche gerade an den Potentiometern anliegen, gemessen und an einen Datenpuffer angefügt.

```
1  async def publish_data():
2      """
3      Sends the buffered data to the websocket server over a websocket-
           connection
4      """
5      global data
6      while not iteration_done or len(data) < 3:
7          await uasyncio.sleep(0.001)
8      temp = data
9      data = []
10     try:
11         ws.send(str([token, temp]))
12     except:
13         connect_websocket()
```

Listing 7.7: Firmware: Datenveröffentlichung über eine Websocketverbindung

Listing 7.7 zeigt die Funktion `publish_data`, welche die Daten über eine Websocket-verbindung an `emuesp32.binobo.io` sendet.

```
async def async_data_publishing():
  """
  Endlessly publishes data over the websocket connection
  """
  while True:
      await publish_data()
```

Listing 7.8: Firmware: Asynchroner Funktionsaufruf von `publish_data`

```
async def uart_input_reader():
  """
  Reads commands retrieved over UART0 --> not fully implemented yet!
  """
  while True:
      cmd = input()
      # TODO: interpret command
      print(cmd)
```

Listing 7.9: Firmware: UART cmd-Reader

Die Funktion `uart_input_reader` aus Listing 7.9 empängt dauerhaft und asynchron zum zweiten Thread des Prozessors des ESP32 UART-Daten, welche als Befehle interpretiert werden sollen. Dieses Feature ist jedoch noch nicht gänzlich implementiert.

```
def uart_data_thread_main():
  """
  Thread-routine, publishes data on separate thread to avoid
     blocking of data retrieving
  This routine should als read uart-data to interpret commands
     received while runtime, but this feature is not implemented yet
  """
  event_loop = uasyncio.get_event_loop()
  event_loop.create_task(async_data_publishing())
  # event_loop.create_task(uart_input_reader())
  event_loop.run_forever()
```

Listing 7.10: Firmware: Thread-Routine zum Senden von Daten und UART-Daten empfangen

Die Funktion `uart_data_thread_main` aus Listing 7.10 ist die Thread-Routine des zweiten Threads. Diese Routine erfasst asynchron die Positionen der Potentiometer und sollte auch auf einem seperaten asynchronen Prozess UART-Daten empfangen und interpretieren, jedoch wie oben bereits kurz erwähnt, ist dieses Feature noch nicht vollständing implementiert.
Mithilfe des Funktionsaufrufes `event_loop.run_forever()` laufen die asynchronen Prozesse solange, bis man `event_loop.stop()` aufruft, dies verhindert das vorzeitige

Terminieren der asynchronen Routinen.

```
def connect_websocket():
  """
  Tries to connect to 'ws://emuesp32.binobo.io' to establish a
     websocket connection
  """
  global ws
  print("[ESP32]: Connecting to Websocket Server...")
  try:
      ws = uwebsockets.client.connect('ws://emuesp32.binobo.io')  #
      print("[ESP32]: Connections successfully established!")
  except:
      print("[ESP32]: Couldn't connect to Websocket!")
```

Listing 7.11: Firmware: Funktion zum Erstellen einer Websocketverbindung

```
def calibrate():
  """
  Reads all channels to determine where the zero-position of the
     potentiometers are, needed to measure the positions
  of the fingers
  """
  global zero_pos
  print("[ESP32]: Calibration starts...")
  input("[ESP32]: Zero Position --> Waiting for verification...\n")
  zero_pos = adc_iter.retrieve_data_raw()

  for i in range(len(zero_pos)):
      zero_pos[i] = zero_pos[i] * f_rc

  print("[ESP32]: Calibration done.")
```

Listing 7.12: Firmware: Funktion zur Nullpunktbestimmung

In der Funktion `calibrate` aus Listing 7.12 wird der verbundene Client dazu aufgefordert, die Nullposition der Potentiometer einzustellen. Eingestellt wird dies, indem man die entsprechende Hand auf eine flache Oberfläche legt. Sobald diese Werte erfasst wurden, werden diese umgerechnet zu Grad.

```
def main():
  global token, ssid, password, ws

  input("Hit <enter> to start configuration...\n")

  print("[ESP32]: Configuration starts...")

  is_storage = "config.txt" in os.listdir()
  use_storage = False
```

```
10
11  if is_storage:
12      with open("config.txt", "r") as config:
13          lines = config.readlines()
14          ssid = lines[0][:-1]
15          password = lines[1][:-1]
16          token = lines[2][:-1]
17          print("[1]" + ssid, "[2]" + password, "[3]" + token, sep="
                \n")
18      x = input("Use␣local␣stored␣config␣data?␣[y/n]:\n")
19      use_storage = x is "y"
20
21  connected = False
22  is_connection_error = False
23  while not connected:
24      if not use_storage or is_connection_error:
25          ssid = input("SSID:\n")
26          password = input("Password:\n")
27      try:
28          connect_to_wifi(ssid, password)
29          print("[ESP32]:␣Connection␣successfully␣established!")
30          connected = True
31      except:
32          is_connection_error = True
33          input("[ESP32]:␣Error␣occurred␣while␣connecting,␣please␣
                try␣again.")
34
35  if not use_storage:
36      token = input("Token:\n")
37
38  calibrate()
39  connect_websocket()
40
41  if not use_storage:
42      store_data = input("Store␣configuration␣data?␣[y/n]:\n") == "y
            "
43      if store_data:
44          with open("config.txt", "w") as config:
45              config.write(ssid + "\n" + password + "\n" + token + "
                    \n")
46              print("[ESP32]:␣Config-Data␣stored!")
47
48  print("[ESP32]:␣Configuration␣done!␣Have␣fun!")
49
50  timer = Timer(0)
51  timer.init(period=33, mode=Timer.PERIODIC, callback=lambda t:
        retrieve_data())
52  _thread.start_new_thread(uart_data_thread_main, ())
53
54
```

```
55 if __name__ == "__main__":
56   main()
```

Listing 7.13: Firmware: Main-Funktion

Listing 7.13 zeigt die Main-Funktion der Firmware sowie dessen Aufruf. In dieser Funktion werden folgende Aufgaben abgearbeitet:

1. Filesystem scannen um gespeicherte Konfigurationsdaten zu finden
2. Den Client fragen, ob dieser die gespeicherten Daten verwenden möchte, sollten Daten am Gerät gespeichert sein
3. WIFI-Verbindung aufbauen
4. Client auffordern, seinen Token (128-Bit UUID) einzugeben
5. Nullposition der Potentiometer bestimmen
6. Verbindung zu Websocketserver aufbauen
7. Sollte der Nutzer neue Daten eingegeben haben, wird dieser gefragt, ob er die neuen Daten speichern will (alte Daten werden hierbei überschrieben)
8. Timer starten, welche mit einer Frequenz von 30Hz die Funktion `retrieve_data`, aus Listing 7.6, aufruft
9. Thread starten, welche die Routine `uart_data_thread_main` startet

Nach dem erfolgreichem Abschließen aller dieser Aufgaben ist der Controller vollkommen einsatzbereit. Der Client ist dann in der Lage in einem beliebigen Webbrowser seine Handbewegungen über den Emulator (`https://www.binobo.io/emulator3D`) mitzuverfolgen.

8. Android App

Die Android Applikation bietet dem jeweiligen Client eine benutzerfreundliche Schnittstelle, um mit dem ESP32 über ein Smart-Phone zu kommunizieren.
Die Android-App wurde mithilfe von Java und der von Google bereitgestellten Android-API und dem dazugehörigem SDK entwickelt. Als IDE wurde `Android Studio`[49] verwendet.

Mithilfe eines OTG Kabels (Host-Mode), kann eine Verbindung zu dem entsprechendem μC hergesellt werden. Zu beachten ist hierbei, dass das Handy der Host sein muss.

Der vollständige Sourcecode ist auf folgendem Github-Repository zu finden:

`https://github.com/psykovski-extended/binobo_connector.git`

An dieser Stelle ist jedoch kurz zu erwähnen, dass die App zum aktuellem Stand nicht gänzlich fertig ist.
Dennoch ist es bereits möglich, sich mit dem ESP32 zu verbinden und soweit zu konfigurieren, dass dieser sich mit dem Websocketserver verbinden kann.

8.1. Verwendete Libraries

Für dieses Projekt fand die USB-Serial Library von `mik3y` Anwendung. Folgendes Github-Repository enthält den gesamten Sourcecode, sowie eine Anleitung zur Einbindung dieser Library mithilfe von `Gradle`:

`https://github.com/mik3y/usb-serial-for-android.git`

Diese Library ist unter der `MIT-Lizenz` als open-source Software verfügbar.

8.2. Funktionsweise

Der Programmfluss dieses Projektes wurde maßgeblich durch die Funktionsweise der Firmware bestimmt, da diese als Basis hierfür diente.

Zu Beginn wurden einige globale Variablen definiert, damit verschiedene Objekte im gesamten Classpath verfügbar sind:

```
1  package android.io.binobo.connector;
2
3  import android.hardware.usb.UsbDeviceConnection;
4
5  import com.hoho.android.usbserial.driver.UsbSerialDriver;
6  import com.hoho.android.usbserial.driver.UsbSerialPort;
7  import com.hoho.android.usbserial.util.SerialInputOutputManager;
8
9  import java.util.Vector;
10
11 public class Globals {
12
13     public static UsbSerialDriver usbSerialDriver;
14     public static UsbDeviceConnection connection;
15     public static UsbSerialPort port;
16     public static SerialInputOutputManager serialIOManager;
17     public static final Vector<String> uartData = new Vector<>();
18     public static Configuration.State configState = Configuration.
           State.UNKNOWN_STATE;
19     public static StringBuilder dataBuffer = new StringBuilder();
20     public static String SSID = "";
21     public static String PASSWORD = "";
22     public static String TOKEN = "";
23
24 }
```

Listing 8.1: Globale Variablen

Innerhalb der App konnte nur mithilfe des Outputs, welchen der ESP32 über den UART0-Port liefert, navigiert werden. Deswegen wurde, um das aktuelle State abzufragen, auf Java-Enumerations zurückgegriffen:

```
1  package android.io.binobo.connector;
2
3  public class Configuration {
4
5    public enum State {
6        HIT_ENTER_TO_START,
7        UNKNOWN_STATE,
8        CONFIG_START,
```

```
9         WIFI_DATA_VALID,
10        WIFI_DATA_INVALID,
11        TOKEN_VALID,
12        TOKEN_INVALID,
13        TOKEN_VALIDATING,
14        LOCAL_DATA_FOUND,
15        WIFI_CONFIG_SSID,
16        WIFI_CONFIG_PASSWORD,
17        TOKEN,
18        CALIBRATION,
19        CALIBRATION_ZERO_POS,
20        CALIBRATION_DONE,
21        WEB_SOCKET_CONNECTING,
22        WEB_SOCKET_CONNECTED,
23        WEB_SOCKET_COULD_NOT_CONNECT,
24        STORE_DATA,
25        DATA_STORED,
26        DONE
27    }
28
29    public static State getState (String state) {
30        switch (state) {
31            case "Hit␣<enter>␣to␣start␣configuration...": return State
                  .HIT_ENTER_TO_START;
32            case "[ESP32]:␣Configuration␣starts...": return State.
                  CONFIG_START;
33            case "Use␣local␣stored␣config␣data?␣[y/n]:": return State.
                  LOCAL_DATA_FOUND;
34            case "SSID:": return State.WIFI_CONFIG_SSID;
35            case "Password:": return State.WIFI_CONFIG_PASSWORD;
36            case "Token:": return State.TOKEN;
37            case "[ESP32]:␣Connection␣successfully␣established!":
                  return State.WIFI_DATA_VALID;
38            case "[ESP32]:␣Error␣occurred␣while␣connecting,␣please␣try
                  ␣again.": return State.WIFI_DATA_INVALID;
39            case "[ESP32]:␣Validating␣token...": return State.
                  TOKEN_VALIDATING;
40            case "[ESP32]:␣Token␣valid.": return State.TOKEN_VALID;
41            case "[ESP32]:␣Token␣not␣valid,␣try␣again.": return State.
                  TOKEN_INVALID;
42            case "[ESP32]:␣Calibration␣starts...": return State.
                  CALIBRATION;
43            case "[ESP32]:␣Zero␣Position␣-->␣Waiting␣for␣verification
                  ...": return State.CALIBRATION_ZERO_POS;
44            case "[ESP32]:␣Calibration␣done.": return State.
                  CALIBRATION_DONE;
45            case "[ESP32]:␣Connecting␣to␣Websocket␣Server...": return
                  State.WEB_SOCKET_CONNECTING;
46            case "[ESP32]:␣Connections␣successfully␣established!":
                  return State.WEB_SOCKET_CONNECTED;
```

```
47              case "[ESP32]: Couldn't connect to Websocket!": return
                    State.WEB_SOCKET_COULD_NOT_CONNECT;
48              case "Store configuration data? [y/n]:": return State.
                    STORE_DATA;
49              case "[ESP32]: Config-Data stored!": return State.
                    DATA_STORED;
50              case "[ESP32]: Configuration done! Have fun!": return
                    State.DONE;
51              default: return State.UNKNOWN_STATE;
52          }
53      }
54  }
```

Listing 8.2: State-Enumeration Objekt

In Listing 8.2 ist die erstellte Enumeration, sowie die Statezuweiungsfunktion `getState` zu sehen.

Um serielle Daten asynchron zum Main-Thread empfangen zu können, um diesen nicht in seiner Funktiosweise zu blockieren, wurde ein sogenannter `Service` erstellt:

```
1  package android.io.binobo.connector;
2
3
4  import android.app.Service;
5  import android.content.Context;
6  import android.content.Intent;
7  import android.hardware.usb.UsbDeviceConnection;
8  import android.hardware.usb.UsbManager;
9  import android.os.Binder;
10 import android.os.Handler;
11 import android.os.IBinder;
12 import android.os.Looper;
13 import android.widget.Toast;
14
15 import androidx.annotation.Nullable;
16
17 import com.hoho.android.usbserial.driver.UsbSerialDriver;
18 import com.hoho.android.usbserial.driver.UsbSerialPort;
19 import com.hoho.android.usbserial.driver.UsbSerialProber;
20 import com.hoho.android.usbserial.util.SerialInputOutputManager;
21
22 import java.io.IOException;
23 import java.nio.charset.StandardCharsets;
24 import java.util.List;
25
26 public class SerialService extends Service implements
       SerialInputOutputManager.Listener {
27
28     class SerialBinder extends Binder {
```

```
29          SerialService getService() { return SerialService.this; }
30      }
31
32      private final Handler mainLooper;
33      private final IBinder binder;
34
35      public SerialService() {
36          mainLooper = new Handler(Looper.getMainLooper());
37          binder = new SerialBinder();
38      }
39
40      @Nullable
41      @Override
42      public IBinder onBind(Intent intent) {
43          return binder;
44      }
45
46      @Override
47      public int onStartCommand(Intent intent, int flags, int startId)
            {
48          // Find all available drivers from attached devices.
49          UsbManager manager = (UsbManager) getSystemService(Context.
                USB_SERVICE);
50          List<UsbSerialDriver> availableDrivers = UsbSerialProber.
                getDefaultProber().findAllDrivers(manager);
51          if (availableDrivers.isEmpty()) {
52              return super.onStartCommand(intent, flags, startId);
53          }
54
55          // Open a connection to the first available driver.
56          UsbSerialDriver driver = availableDrivers.get(0);
57          UsbDeviceConnection connection = manager.openDevice(driver.
                getDevice());
58          if (connection == null) {
59              return super.onStartCommand(intent, flags, startId);
60          }
61
62          UsbSerialPort port = driver.getPorts().get(0); // Most
                devices have just one port (port 0)
63          try {
64              port.open(connection);
65              port.setParameters(115200, 8, UsbSerialPort.STOPBITS_1,
                    UsbSerialPort.PARITY_NONE);
66
67              SerialInputOutputManager serialInputOutputManager = new
                    SerialInputOutputManager(port, this);
68              serialInputOutputManager.start();
69
70              Toast.makeText(this, "Connection␣successfully␣
                    established!", Toast.LENGTH_LONG).show();
```

```
71
72              Globals.connection = connection;
73              Globals.port = port;
74              Globals.usbSerialDriver = driver;
75              Globals.serialIOManager = serialInputOutputManager;
76          } catch (IOException e) {
77              Toast.makeText(this, "Error␣occurred!", Toast.
                    LENGTH_LONG).show();
78          }
79
80          return super.onStartCommand(intent, flags, startId);
81      }
82
83      @Override
84      public void onNewData(byte[] data) {
85          bufferData(data);
86      }
87
88      synchronized private void bufferData(byte[] data) {
89          String dataIn = new String(data);
90
91          for (char c : dataIn.toCharArray()) {
92              Globals.dataBuffer.append(c);
93
94              if (c == '\n'){
95                  String dataAsString = Globals.dataBuffer.toString().
                        trim();
96
97                  Globals.configState = Configuration.getState(
                        dataAsString);
98                  Globals.uartData.add(dataAsString);
99
100                 if (Globals.configState == Configuration.State.
                        HIT_ENTER_TO_START) {
101                     try {
102                         Globals.port.write("\r".getBytes(
                                StandardCharsets.UTF_8), 100);
103                     } catch (IOException e) {
104                         e.printStackTrace();
105                     }
106                 }
107
108                 if (dataAsString.startsWith("[1]")) Globals.SSID =
                        dataAsString.substring(3);
109                 else if (dataAsString.startsWith("[2]")) Globals.
                        PASSWORD = dataAsString.substring(3);
110                 else if (dataAsString.startsWith("[3]")) Globals.
                        TOKEN = dataAsString.substring(3);
111
112                 Globals.dataBuffer = new StringBuilder();
```

```
113                 }
114             }
115         }
116
117         @Override
118         public void onRunError(Exception e) {
119             stopSelf();
120         }
121     }
```

Listing 8.3: SerialService Klasse

Um einen Service zu erstellen, muss eine eigene Javaklasse erstellt werden, welche von `android.app.Service` erbt.
Anzumerken ist an dieser Stelle, dass auch hier das `DDD` Pattern angewendet wird, sowie eine Abwandlung des MVC - Designpatterns, genannt `MVP`[42]. `MVP` steht für `Model - View - Presenter` und teilt ein Projekt, ähnlich wie bei Spring Boot, in drei abstrakte Schichten auf. Da jedoch bei einer Android Applikation keine HTTP-Requests eingehen, gibt es keine Controller-Schicht.

9. Webhosting

Um das entwickelte Projekt für die Welt zugänglich zu machen, mussten die entwickelten Server in World-Wide-Web verfügbar gemacht werden. Dies ist möglich, indem man eine Domain erwirbt und diese auf die IP-Adresse des entwickelten Servers verweist.
Nachstehende Kapitel 9.1 bis ?? erläutern das Vorgehen, um Software im Internet verfügbar zu machen.

9.1. DDNS und Domainname

`DDNS` steht für `Dynamic-Domain-Name-Service`. Ähnlich wie ein DNS ist DDNS eine Möglichkeit um einen Computer in Internet zu lokalisieren um mit diesem zu kommunizieren. Anders als bei DNS, wo eine statische IP-Adresse benötigt wird, kann bei DDNS die entsprechende IP-Eintrag am DNS-Server dynamisch geändert werden. Dies ermöglicht es, Server an Orten zu hosten, welche keine statische IP-Adresse besitzen.
Für dieses Projekt wurde der DDNS-Anbieter `Dynu` verwendet. Dynu bietet die entsprechenden Funktionalitäten kostenlos an.
Bei der Wahl des DDNS-Anbieters wurde auch darauf geachtet, dass man eine erworbene Domain zu diesem transferieren kann.

Für dieses Projekt wurde folgende Domain erworben:

`binobo.io`

Erworben wurde diese Domain über den Domain-Anbieter `https://www.name.com`.

Um diese Domain über den DDNS-Anbieter verwalten zu lassen, müssen für diese Domain auf `name.com` die Domain-Server von `Dynu` angegeben werden. Die entsprechenden Domain-Server von `Dynu` findet man unter folgender URL:

`https://www.dynu.com/en-US/ControlPanel/DDNS`

Hier muss man dann die entsprechende Domain auswählen, und unter der Registerkarte `Configuration` sind diese Server gelistet.

9.2. Docker

Docker[43] wird verwendet, um Software zu containerisieren. Die Containerisierung von Software ermöglicht es, ein eigenes System (Linux) mit allen benötigten Abhängigkeiten und Konfigurationen zu entwerfen und kompakt von A nach B zu befördern. Weiters fällt die Konfiguration der Software - nach dem kreieren des Docker-Images - komplett weg. Danach kann das entsprechende Docker-Image mittels eines kurzen Befehls gestartet werden und bedarf keiner weiteren Konfiguration.
Weiters ermöglicht Docker sogenannte `Microservices` zu erstellen. `Microservices` sind Komponenten einer Softwarearchitektur, welche getrennt voneinander innerhalb verschiedener Docker-Containers gesartet werden. Jedoch können diese Container trotz ihrer virtuellen Trennung miteinander über das TCP/IP und UDP Protokoll kommunizieren.

Für dieses Projekt wurden drei Docker-Container erstellt:

- PostgreSQL - Datenbank
- Spring-Boot Webserver
- Python Websocketserver

In Kapitel 9.2.1 & 9.2.2 ist die Konfiguration dieser Container zu sehen und wie diese mithilfe von `docker-compose` zu einem `Netzwerk` zusammengefasst wurden.

9.2.1. Dockerfiles

Um einen Docker-Container zu erstellen, muss dieser konfiguriert werden. Auf `Docker Hub` gibt es bereits vorgefertigte Container für z.B.: Java, Python und PostgreSQL. Hierdurch fällt die Installation der benötigten Software innerhalb des jeweiligen Containers weg.
Nachstehendes Listing (9.1) zeigt das erstelle Dockerfile für den Spring-Boot Webserver:

```
1   FROM openjdk:13-alpine
2   MAINTAINER binobo.io
3   ADD target/binobo-2.0-Alpha.jar binobo.jar
```

```
4   ENTRYPOINT ["java","-jar","binobo.jar"]
```

Listing 9.1: dockerfile: Spring-Boot Webserver

In Listing 9.1 wird ein Docker-Container, welcher von `openjdk:13-alpine` sämtliche Konfigurationen übernimmt, erstellt. Zeile 2-4 fügen weitere Konfigurationen zu diesem Container hinzu.

Für den Python-Websocketserver wurde ebenfalls ein Docker-Container erstellt, welche als Basis das Docker-Image `python:3.9` verwendet:

```
1   FROM python:3.9
2   ADD socket_server.py server.py
3   RUN pip install websockets
4   EXPOSE 8080
5   CMD ["python", "-u", "server.py"]
```

Listing 9.2: dockerfile: Python Websocketserver

In Listing 9.2 auf Zeile 2 wird das Python-Sript, welches der Websocketserver erstellt, in den Docker-Container kopiert. Zeile 3 installiert die benötigte Python-Libray. Mit Zeile 4 wird der Port 8080 von diesem Image freigegeben und mit Zeile 5 wird das entsprechende Python-Skript gestartet.

Weiters wurde noch ein PostgreSQL Container erstellt, jedoch benötigt man hierfür kein eigenes Dockerfile. Dieser wird innerhalb eines `docker-compose.yml` - Files definiert - siehe Kapitel 9.2.2.

9.2.2. docker-compose

Um die erstellten Docker-Container zu einem Netzwerk zusammenzufassen, bietet Docker das `CLI` mit Namen `docker-compose`[44] an. Hiermit ist es möglich, über eine YAML-Konfigurationsdatei mit dem Namen `docker-compose.yml` mehrere Microservices über einen Befehl zu starten.

Nachstehendes Listing zeigt das entworfene `docker-compose.yml` - File dieses Projekts:

```
1   version: '3.8'
2   services:
3     app:
4       container_name: binobo_server
5       image: 'lovetinsky99/binobo_server:latest'
6       build: ./
7       ports:
```

```
8           - "80:80"
9         depends_on:
10          - binobo_database
11        environment:
12          - SPRING_DATASOURCE_URL=jdbc:postgresql://binobo_database
                :5432/binobo_db
13          - SPRING_DATASOURCE_USERNAME=postgres
14          - SPRING_DATASOURCE_PASSWORD=password
15          - SPRING_JPA_HIBERNATE_DDL_AUTO=update
16      websocket_server:
17        container_name: binobo_websocket_server
18        image: 'lovetinsky99/binobo_websocket_server:latest'
19        ports:
20          - "8080:8080"
21        depends_on:
22          - binobo_database
23      binobo_database:
24        image: postgres
25        container_name: binobo_database
26        ports:
27          - "3506:5432"
28        environment:
29          - POSTGRES_PASSWORD=password
30          - POSTGRES_USER=postgres
31          - POSTGRES_DB=binobo_db
```

Listing 9.3: docker-compose.yml

Um das `docker-compose.yml` - File zu starten, muss folgender Befehl ausgeführt werden in der Befehlszeile:

```
docker-compose up --build
```

Mit diesem Befehl wird das entsprechende Docker-Netzwerk gestartet und gleichzeitig - sollten Anderungen an den Images vorgenommen worden sein - neu kreiert (angegeben durch den Parameter `-build`).

Zu beachten ist jedoch, dass diese Images nur für Prozessoren der Type `amd64` kompiliert wurden. Aus diesem Grund können diese Container nicht auf z.B. einem RaspberryPI gestartet werden, da dieser einen `AMR`-Prozessor besitzt.

9.2.3. Docker Hub - Repositories

Docker bietet ebenfalls - ähnlich wit Github - ein Remote-Repository-Hub an, um die entsprechende Software auf externen Servern zu speichern und teilen.
Für dieses Projekt wurde zwei öffentliche Docker-Hub-Repositories erstellt:

- `https://hub.docker.com/repository/docker/lovetinsky99/binobo_server`
- `https://hub.docker.com/repository/docker/lovetinsky99/binobo_websocket_server`

Diese Docker-Images sind ebenfalls als open-source Software frei im Internet verfügbar.

9.3. Reverse-Proxy

Der Reverse-Proxy ist ein Proxy, welcher externen Computern den Zugriff auf netzwerkinterne Ressourcen gewährt. Anders als ein gewöhnlicher Forward-Proxy, welcher internen Computern den Zugriff auf ein externes Netzwerk gewährt - somit ist dieser atypisch und umgekehrt[45]. Die Richtung des Aufrufes der Ressourcen ist daher umgekehrt.
Ein Reverse-Proxy wurde benötigt, um Anfragen, welche von außerhalb des Netzwerkes kommen, an den entsprechenden Client weiterzuleiten. Weiters ermöglicht solch ein Proxy die Erstellung von `Subdomains` - vergleiche Kapitel 9.3.1
Nachstehende Abbildung (9.1) zeigt die Aufrufskette mit einem Reverse-Proxy, welche für dieses Projekt realisiert wurde:

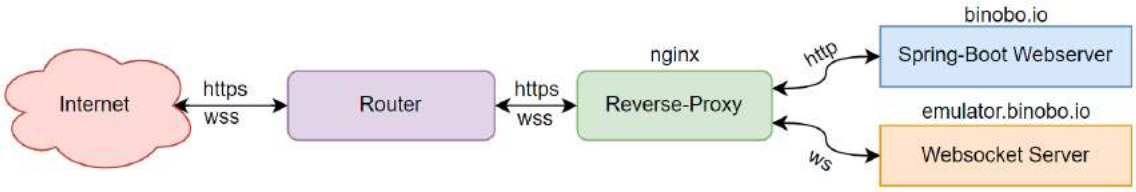


Abbildung 9.1.: Reverse-Proxy

Sobald eine Anfrage aus dem Internet kommt, nimmt diese der Reverse-Proxy entgegen und filtert die entpsrechende Ziel-Domain dieser Anfrage. Sollte diese Domain existieren, wird der Request weitergeleitet, andernfalls schlägt die Anfrage fehl.

9.3.1. nginx

`nginx`[46] ist ein Reverse-Proxy, mit welchen man Client-Server Verbindungen verwalten kann. Mit dieser Software ist es ebenfalls möglich Subdomains aus Client-Requests zu filtern und entsprechend weiterzuleiten.
Der Reverse-Proxy wurde auf einem `MSI` - Rechner aufgesetzt, auf welchem `Ubuntu`

20.04 läuft. Durch folgenden Befehl kann nginx installiert werden:

```
sudo apt install nginx nginx-extras
```

Um den Reverse-Proxy zu konfigurieren, muss man in des entsprechende Verzeichnis wechslen. Folgender Befehl wechselt in dieses Verzeichnis:

```
cd /etc/nginx/sites-available
```

Innerhalb dieses Verzeichnisses muss ein File mit den entsprechenden Konfigurationen beaufschlagt werden:

```
sudo nano reverse-proxy.conf
```

Innerhalb des Files reverse-proxy.conf müssen dann die entsprechenden Zeilen hinzugefügt werden. Aus sicherheitstechnischen Gründen wird jedoch die erstellte Konfigurationsdatei nicht gelistet. Dies wird nicht gemacht, da in diesem File das SSL-Zertifikat angeführt wird und die entpsrechenden IP-Adressen der etwaigen Hosts.
Nachstehendes Listing zeigt ein Beispiel einer solchen Konfigurationsdatei:

```
1   server {
2       server_name binobo.io;
3       location / {
4           proxy_pass http://192.168.0.4:8080
5       }
6   }
7   server {
8       server_name emulator.binobo.io; # py-websocketserver domain
9       location / {
10          proxy_pass http://192.168.0.5:8081
11          proxy_http_version 1.1;
12          proxy_set_header Upgrade $http_upgrade;
13          proxy_set_header Connection "Upgrade";
14          proxy_set_header Host $host;
15      }
16  }
17  server {
18    server_name emuesp32.binobo.io; # py-websocketserver endpoint
          for esp32
19    location / {
20        proxy_pass http://192.168.0.5:8082
21        proxy_http_version 1.1;
22        proxy_set_header Upgrade $http_upgrade;
23        proxy_set_header Connection "Upgrade";
24        proxy_set_header Host $host;
25    }
```

```
26 }
```

Listing 9.4: reverse-proxy.conf

Anzumerken ist an dieser Stelle, dass tatsächlich zwei Websocket-Endpoints erstellt wurden. Dies musste getan werden, da der ESP32 nicht über verschlüsselte Websocketverbindungen kommunizieren kann.

In Abb. 9.1 ist die Aufrufskette über einen Reverse-Proxy zu sehen. Dieser nimmt die Anfragen von Clienten stellvertretend an und sendet die entsprechenden Ergebnisse für den jeweiligen Server zurück zum Client. An dieser Stelle ist anzumerken, dass jeder Request vor dem Reverse-Proxy, welche von außerhalb kommt, mittels SSL verschlüsselt ist - außer die Domain `emuesp32.binobo.io`.

9.3.2. certbot

Der `certbot`[34] stellt `self-signed` SSL Zertifikate aus. Die entsprechende Software kann über folgenden Befehl installiert werden:

```
sudo apt install certbot python3-certbot-nginx
```

Nach der Installation kann mit folgendem Befehl die jeweilige Domain verschlüsselt werden:

```
sudo certbot --nginx
```

Nach Ausführung dieses Befehls starte in der Konsole ein Dialog, welcher den Nutzer durch den Prozess der SLL-Zertifizierung führt.

Nach Abschluss dieses Dialogs muss der Reverse-Proxy neu gestartet werden:

```
sudo nginx -s reload
```

10. Setup-Anleitung

10.1. Voraussetzungen

Um dieses Projekt wie in dieser Dukomentation beschrieben realisieren zu können, müssen gewisse Voraussetzungen erfüllt werden. Weiters muss man entsprechende Hardware besitzten, um unter anderem die Controller-Teile zu drucken:

- SLA-3D Drucker (z.B.: Anycubic)
- UV-Licht härtendes Epoxidharz
- ESP32
- Reinraum-Handschuh
- hochflexible Drähte
- Mirco-USB Kabel
- OTG-Kabel (Handy als Host, benötigt um über die App den Controller zu konfigurieren)

Für die Software müssen - je nach dem wie diese lokal gehostet werden soll - entsprechende Programme installiert werden:

- IDE mit Java-Support (IntelliJ, Eclipse, ...)
- Docker
- docker-compose
- Thonny-IDE oder Mu Editor (für Micropython)

10.2. Software

10.2.1. ESP32 aufsetzen

Um die entwickelte Firmware auf den ESP32 laden zu können müssen folgende Arbeitsschritte ausgeführt werden:

1. Firmware herunterladen
2. Micropython-Firmware flashen mit `esptool.py`
3. `uwebsockets`-Library auf den ESP32 laden
4. Firmware aus Kapitel 7 auf den ESP32 laden

Um die Firmware herunterzuladen, muss zuerst das Terminal gestartet werden. Sobald dies gestartet ist, wird empfohlen mit `cd` in des gewünschte Verzeichnis zu navigieren. Dort angekommen muss folgender Befehl ausgeführt werden, um die Firmware herunterzuladen:

```
git clone https://github.com/psykovski-extended/binobo_firmware.
    git
```

Listing 10.1: Firmware herunterladen mithilfe von git

Anzumerken ist hier, dass für den in Listing 10.1 ausgeführten Befehl `git` auf dem entsprechenden Rechner installiert sein muss.

Das erwähnte Python Programm `esptool.py` ist ein Python-Skript, welches von der Firma `Espressif` als open-source Software zur Verfügung gestellt wird. Dieses Skript wird verwendet, um mit dem ROM-Bootloader der μ`Cs` von der Firma `Espressif` kommunizieren zu können. Folgendes Github-Repository enthält den Sourcecode dieses Skripts:

`https://github.com/espressif/esptool.git`

Mithilfe des Python-Package-Management-Tools `pip` kann man diese Library installieren:

```
pip install esptool
```

Um entwaige Files auf den ESP32 lden zu können, bietet sich ein weiteres Python-Skript der Firma `adafruit` an, welches ebenfalls über `pip` installiert werden kann:

```
pip install adafruit-ampy
```

Die aktuelle Micropython-Firmware findet man unter folgendem Link:

```
https://micropython.org/download/esp32/
```

Bevor nun die Firmware geflasht werden kann, muss man den ROM-Speicher des ESP32 vollständig löschen, dies gelingt mit folgendem Befehl:

```
esptool.py --chip esp32 --port COM3 erase_flash
```

Danach kann man die entsprechende Firmware auf den ESP32 laden:

```
esptool.py --chip esp32 --port COM3 --baud 460800 write_flash -z 0
    x1000 esp32-20220117-v1.18.bin
```

Empfohlen ist, immer die aktuellste Version der Firmware zu verwenden. Weiters ist anzumerken, dass der richtige COM-Port angegeben werden muss. Um diesen ausfindig zu machen, kann man im Gerätemanager von Windows unter `Anschlüsse (COM & LPT)` nachsehen:

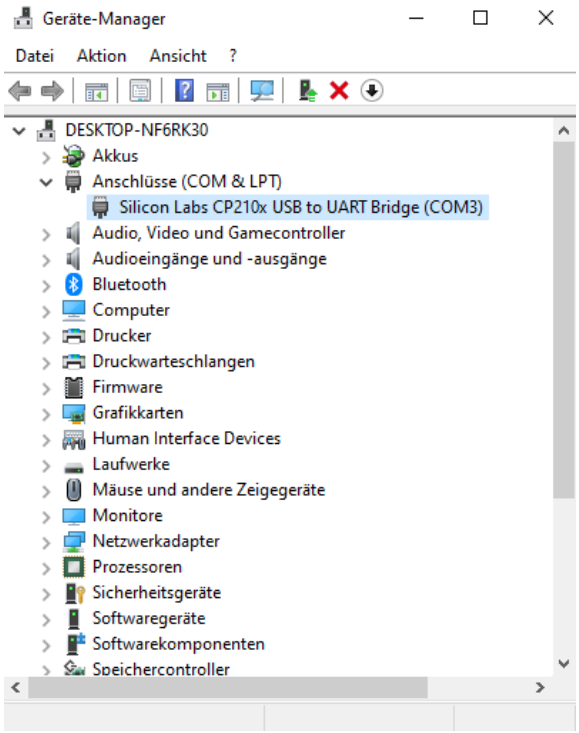


Abbildung 10.1.: Geräte-Manager, COM-Port bestimmen

Mit folgendem Befehl wird die Library `uwebsockets`, auf den ESP32 geladen:

```
ampy --port COM3 mkdir uwebsockets && ampy --port COM3 put protocol.
   py uwebsockets/protocol.py && ampy --port COM3 put client.py
   uwebsockets/client.py
```

Listing 10.2: Flashen der `uwebsockets`-Library

In Listing 10.2 wird mithilfe von `ampy` ein Verzeichnis mit Namen `uwebsockets` erstellt. Danach wird das Skript `protocol.py` und `client.py` auf den Chip geladen. Anzugeben ist hier der entsprechende Port, die Aktion, welche man ausführen will (in diesem Fall: `put`), das File, welches hochgeladen werden soll, und als letzter Parameter muss man angeben, wohin das File gespeichert werden soll.

Sobald die oben genannten Schritte erfolgreich abgeschlossen wurden, kann nun die Firmware aus Kapitel 7 hochgeladen werden:

```
ampy --port COM3 put firmware_v1_alpha.py boot.py
```

Listing 10.3: Flashen der Binobo-Firmware

In Listing 10.3 wird die Binobo-Firmware auf den ESP32 geladen, hierbei wird die `boot.py` Datei überschrieben. Dies wird gemacht, damit das Skript nach dem Bootvorgang gestratet wird.
Sollte man nicht wollen, dass das Skript immer beim Bootvorgang gestartet wird, so kann man das Skript auch nur ausführen lassen auf dem ESP32:

```
ampy --port COM3 run firmware_v1_alpha.py
```

Listing 10.4: Starten der Binobo-Firmware ohne Flashen

Hier (Listing 10.4) wird das Skript auf den ESP32 geladen und gestartet. Es wird aber wieder gelöscht, sobald das Programm terminiert oder der ESP32 neu gestartet wird.

10.2.2. Webserver als JAR lokal hosten

Um den Server vom Sourcecode heraus zu starten, muss zumal dieser von Github geklont werden. Der entsprechende Link ist in Kapitel 5 zu finden.
Als IDE wird die `IntelliJ` empfohlen. Jedoch bietet `VS-Code` ebenfalls den notwendigen Support.

Weiters muss `PostgreSQL` installiert werden. Ebenfalls ist es empfehlenswert die Software `pgAdmin 4` zu installieren.

Nach Installation von `PostgreSQL` und `pgAdmin 4` muss man eine Datenbank mit Namen `binobo_db` erstellen. `pgAdmin` bietet hierfür entsprechende Funktionalität.

Um den Server nun lokal starten zu können, muss innerhalb des Projektordners ein `application.properties` - File erstellt werden. Dieses File muss in dem Verzeichnis *\src\main\resources* gespeichert werden. Folgende Zeilen müssen eingefügt werden:

```
1  # creates SPRING_SESSION db
2  spring.session.store-type=jdbc
3  spring.session.jdbc.initialize-schema=always
4
5  # setup PostgreSQL Database
6  spring.datasource.url=jdbc:postgresql://localhost:5432/binobo_db
7  spring.datasource.username=postgres
8  spring.datasource.password=<your-root-pw>
9  spring.jpa.hibernate.ddl-auto=update
10 # this can be set to false
11 spring.jpa.show-sql=true
12
13 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.
       PostgreSQL92Dialect
14
15 # mail config for email-verification
16 # here you can also change this to any other smtp server
17 spring.mail.host=smtp.gmail.com
18 spring.mail.port=587
19 spring.mail.username=your.email@something.co
20 spring.mail.password=<your-pw>
21 spring.mail.properties.mail.smtp.auth=true
22 spring.mail.properties.mail.smtp.starttls.enable=true
```

Listing 10.5: application.properties

Zu beachten ist jedoch, dass einige Konfigurationen noch geändert werden müssen:

- Zeile 6: Port ändern bei bedarf
- Zeile 7: `spring.datasource.username` auf den entsprechenden Username ändern
- Zeile 8: `spring.datasource.password` auf das entsprechende Passwort ändern
- Zeile 19: `spring.mail.username` auf ein existierendes Gmail-Konto ändern
- Zeile 20: `spring.mail.password` auf das entsprechende Passwort des Gmail-Kontos ändern

Nachdem dieses File erstellt worden ist, kann der Server gestartet werden. Jedoch ist noch zu beachten, dass die Datenbank nocht nicht die notwendigen Tabelleneinträge enthält, welche für das Registrieren benötigt werden. Nachstehendes Listing (10.6) zeigt die entsprechende Query, welche innerhalb der Datenbank `binobo_db` ausgeführt werden soll:

```
1  insert into role (name, id) values ('ROLE_USER', 1);
2  insert into role (name, id) values ('ROLE_ADMIN', 2);
3  insert into role (name, id) values ('ROLE_OPERATOR', 3);
4
5  insert into category (name, color_code) values ('Program', '#403891'
       );
6  insert into category (name, color_code) values ('Mathematics', '
       #39568c');
7  insert into category (name, color_code) values ('Troubleshooting', '
       #ff4c4c');
8  insert into category (name, color_code) values ('Science', '#1f968b'
       );
9  insert into category (name, color_code) values ('Coding', '#73d055')
       ;
10 insert into category (name, color_code) values ('Other', '#fde725');
```

Listing 10.6: data.sql - Initiale Datenbankeinträge

Hiernach sollte der Server problemlos verwendet werden können!

10.2.3. Websocketserver lokal hosten

Um den Websocketserver zu starten wird eine IDE mit Python-Support benötigt. Hier bietet sich entweder `PyCharm oder VS-Code` an. Hier muss ebenfalls wieder ein Repository geklont werden, welches in Kapitel 6 erwähnt worden ist.

Bevor dieser jedoch gestartet werden kann, muss die Python-Library `websockets` installiert werden:

```
pip install websockets
```

Nach der erfolgreichen Installation dieser Library kann dieser Server ebenfalls gestartet werden.

10.2.4. Server-Pool mit docker-compose lokal starten

Um das Server-Pool als Docker-Netzwerk zu starten, wird vorausgesetzt, dass Docker sowie `docker-compose` installiert ist.

Sollten diese Programme installiert sein, muss innerhalb eines beliebeigen Ordners folgendes File mit Namen `docker-compose.yml` erstellt werden:

```
version: '3.9'
services:
  app:
    container_name: binobo_server
    image: 'lovetinsky99/binobo_server:latest'
    ports:
      - "80:80"
    depends_on:
      - binobo_database
    environment:
      - SPRING_DATASOURCE_URL=jdbc:postgresql://binobo_database
        :5432/binobo_db
      - SPRING_DATASOURCE_USERNAME=some_username
      - SPRING_DATASOURCE_PASSWORD=some_password
      - SPRING_JPA_HIBERNATE_DDL_AUTO=update
  websocket_server:
    container_name: binobo_websocket_server
    image: 'lovetinsky99/binobo_websocket_server:latest'
    ports:
      - "8080:8080"
    depends_on:
      - binobo_database
  binobo_database:
    image: postgres
    container_name: binobo_database
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_PASSWORD=some_password
      - POSTGRES_USER=some_username
      - POSTGRES_DB=binobo_db
```

Listing 10.7: docker-compose.yml - Server-Pool als Docker-Netzwerk

Nach der erstellung dieser Datei kann mithilfe folgendem Befehls das Netzwerk gestartet werden:

```
docker-compose up
```

Nach dem Ausführen dieses Befehls ist der Server unter `http://localhost` erreichtbar.

10.3. Hardware

Die Konstruktion der Hardware ist in den folgenden Kapitlen auf drei Überkategorien an Arbeitsschritten aufgeteilt, welche in chronologischer Reihenfolge auszuführen sind. Innerhalb der Kategorien ist der Ablauf nicht notwendigerweise streng einzuhalten, allerdings ist empfohlen, logisch zu denken, bevor gehandelt wird.

10.3.1. Druck der Komponenten

Bevor mit dem Aufbau begonnen werden kann, müssen die entsprechenden Komponenten 3D-gedruckt werden. Dazu wird ein SLA-Drucker empfohlen, aus Gründen, die in Kapitel 3.2.1 genannt wurden. Da keine Experimente mit FDM-Druckern durchgeführt wurden, ist nicht garantiert, dass damit nutzbare Komponenten produziert werden können. Bei Details zu den Modellen wird auf Kapitel 3.2.3 verwiesen. Die STL-Dateien stehen auf dem folgend verlinkten GitHub zur Verfügung:
`https://github.com/psykovski-extended/binobo_models.git`

Da es viele verschiedene Drucker und damit verbundene Software gibt, sind diese Files selbst zu slicen. Tabelle 10.1 gibt unbedingt notwendige Komponenten und deren Stückzahl an:

Name	Stückzahl
controller_poti_holder	6
controller_poti_holder_th	1
controller_poti_holder_p	1
controller_poti_holder_p_cut	1
controller_poti_fingertip	3
controller_poti_fingertip_th	1
controller_poti_fingertip_p	1
controller_poti_thumb_base	1
controller_plate_inner	1
controller_plate_thumb	1
controller_plate_outer	1
controller_strut_lr M	4
controller_strut_lr S	1

Tabelle 10.1.: Liste erforderlicher Komponenten

Die Größe der benötigten Hebel und Streben ist abhängig von der ungefähren Größe der Hand. Weil jeder Mensch anders ist, kann nicht mit 100% Genauigkeit angegeben werden, welche dieser Teile für die entsprechende Hand passen. Daher sind hier

nur die Maße des Prototypen angegeben. Bei der Montage ist es möglicherweise erforderlich weitere Streben und Hebel nachzudrucken, wenn andere Maße notwendig sind.

Name	Stückzahl
controller_strut L	1
controller_strut M	2
controller_strut S	7
controller_strut XS	5
controller_lever L	5
controller_lever M	9
controller_lever S	1

Tabelle 10.2.: Liste varabel erforderlicher Komponenten

Die Hebel (`controller_lever`) sind als Paare angegeben, da in den entsprechenden Dateien auch beide Komponenten inkludiert sind.

Anzumerken ist, dass die Teile für eine geringe Druckzeit optimiert sind und daher größtenteils ohne Stützkonstruktion zu drucken sind. Dabei ist die Orientierung aus den STL-Files zu übernehmen. Die Halterungen am Fingerrücken stellen eine Ausnahme dar, da diese eine Lasche zur Befestigung des nächsten Moduls aufweisen, welche nicht ohne Stützen erfolgreich gedrückt werden kann.
Somit müssen bei diesen Elementen folgende oder vergleichbare Stützstrukturen hinzugefügt werden:

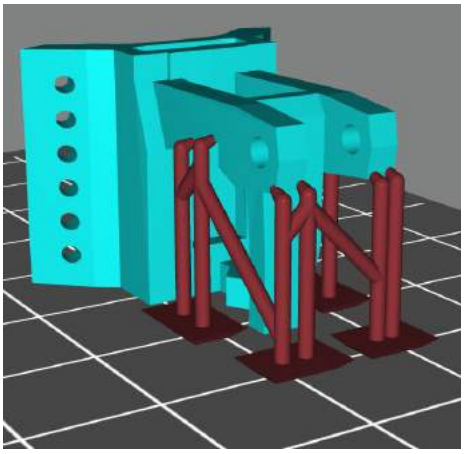


Abbildung 10.2.: Konfiguration der Stützen bei controller_poti_holder

In Abbildung 10.2 wird das empfohlene Stützmuster anhand der Komponente controller_poti_holder dargestellt. Bei allen ähnlichen Modellen mit diesen Laschen müssen Stützen nach diesem Vorbild nachgerüstet werden.
Je nach Druckereigenschaften ist es notwendig den Überhang bei den Fingerspitzen ebenfalls zu stützen.

Sobald die Komponenten ausgedruckt sind, kann mit deren Montage begonnen werden.

10.3.2. Montage der Komponenten

Es wird empfohlen, die Konstruktion zu zweit durchzuführen, da Nähen in Nähe der eigenen primären Hand alleine kompliziert ist.
Die Montage der Potentiometer verläuft leichter, wenn sie vor der Befestigung am Handschuh erfolgt, daher ist empfohlen, damit zu beginnen. Am einfachsten verläuft dies, wenn man dieser Bewegung folgt:

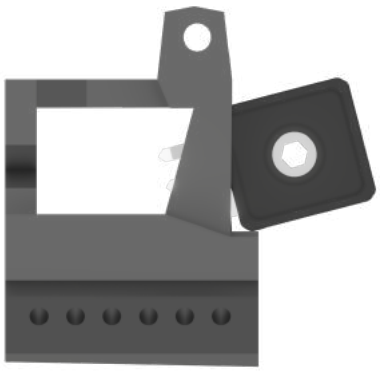


Abbildung 10.3.: Montage: Schritt 1

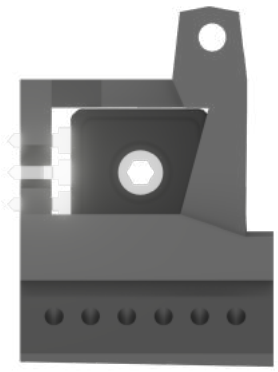


Abbildung 10.4.: Montage: Schritt 2

Bei der Montage ist etwas Kraft von Nöten, da die Potentiometer nur durch die unterdimensionierte Lücke in der Halterung fixiert werden. Es ist auch darauf zu achten, die Potentiometer richtig herum zu implementieren, sodass sie in die Aussparungen an der Strebe passen. Auf der Daumenbasisplatte ist das Potentiometer ebenfalls nach diesem System anzustecken. Beim unteren Daumengelenk, `controller_poti_thumb_base`, muss der Widerstand seitwärts auf eine ähnliche weise in die Passform gepresst werden.

Die Potentiometer, die an der Basis der normalen Finger liegen müssen erst mit den entsprechenden `controller_strut_lr`-Varianten beaufschlagt werden, bevor sie in ihre Position geschoben werden, da erst durch diese Komponente die erforderliche Höhe erreicht wird, um den Teil zu stabilisieren.
Am unteren Daumengelenk muss ebenfalls ein `controller_strut_lr M`-Element aufgesteckt werden.

Nun wird es Zeit für den ersten komplexeren Arbeitsschritt: Das Annähen der Elemente. Es wird empfohlen, einen flexiblen, engen Handschuh zu nutzen, da es an-

sonsten zum Abheben der Komponenten von der Hand kommen kann.
Die Platzierung der Komponenten ist zum Teil nutzerabhängig, allerdings ist die Reihenfolge fix vorgegeben. Diese ist in folgender Grafik, Abbildung 10.5, gegeben:

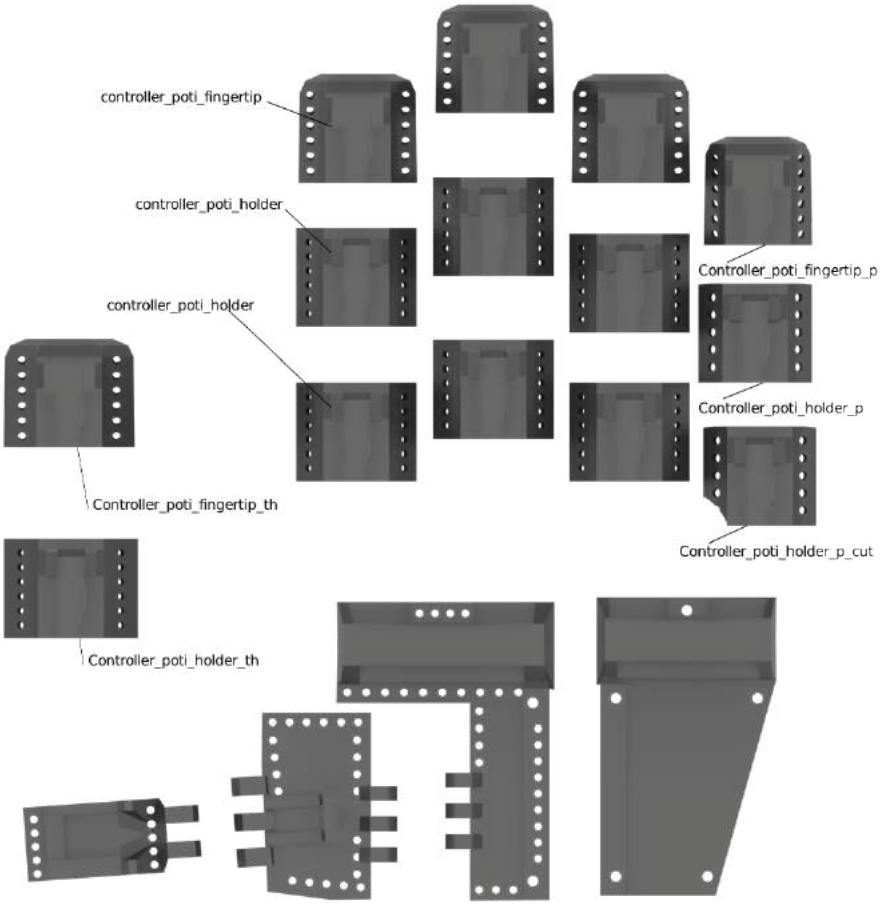


Abbildung 10.5.: Anordnungsschema auf dem Handschuh

Alle Halterungen sollten möglichst mittig auf den unterliegenden Fingerknochen liegen und lotrecht nach oben stehen. Empfohlen ist, die Halterungen an den Fingern zuerst zu befestigen, bevor die Platten und `controller_poti_thumb_base` montiert werden, da deren genaue Position und Orientierung von ihnen abhängig sind.
Nachdem die Teile angenäht und angeschraubt wurden, müssen die beiden großen Platten mit den Multiplexer-Boards beaufschlagt werden. dazu werden sie, nach einer Mutter, die als Abstandhalter dient, mit den Löchern auf die beiden handinneren Schrauben gesteckt und mit einer weiteren Mutter befestigt.

Als letzter Schritt dieser Etappe müssen passende Hebel- und Strebenlängen gewählt werden. Man erkennt die korrekte Längenkombination daran, dass beim Abbiegen

des Fingers kaum spürbarer Widerstand beim Erreichen des vollen Winkels spürbar ist und die am Potentiometer verfügbaren 90° voll ausgekostet werden. Hierbei kann es erforderlich sein weitere Komponenten zu drucken. Auch wichtig anzumerken ist, dass die dickeren Bottom-Layers, die beim SLA-Druck auftreten an Löchern manchmal aufgestochen oder aufgeritzt werden müssen. Weiters ist wichtig, dass die Hebel, besonders die, die in das Potentiometer greifen, vollständig ausgehärtet sind, bevor sie verbaut werden, da es andernfalls zu Abrieb und Verschleiß kommen kann.

10.3.3. Verlöten der Elektronik

Zuletzt müssen die Verbindungen zwischen den Widertständen und den Multiplexern gelötet werden. Dazu werden hochflexible Drähte empfohlen, beispielsweise *Flexivolt 0.25mm*-Drähte, wie sie im Prototypen genutzt werden. Tabelle 10.3 gibt an, wie die Läuferkontakte der Potentiometer in jedem Segment zu verschalten sind:

Segment	Kontakt
	Multiplexer 1
Daumen auf Platte	S0
Daumen Links/Rechts	S1
Daumen Mitte	S2
Daumenspitze	S3
Zeigefinger Links/Rechts	S4
Zeigefinger unten	S5
Zeigefinger Mitte	S6
Zeigefingerspitze	S7
Mittelfinger Links/Rechts	S8
Mittelfinger unten	S9
Mittelfinger MItte	S10
Mittelfingerspitze	S11
	Multiplexer 2
Ringfinger Links/Rechts	S0
Ringfinger unten	S1
Ringfinger Mitte	S2
Ringfingerspitze	S3
Kleiner Finger Links/Rechts	S4
Kleiner Finger unten	S5
Kleiner Finger Mitte	S6
Kleiner Finger Spitze	S7

Tabelle 10.3.: Verschaltungstabelle der Läuferkontakte

Die anderen Kontakte der Potentiometer müssen mit 5V bzw GND beaufschlagt werden, sodass ein Spannungsteiler entsteht, um die Rotation zu messen. Obwohl die Polung technisch irrelevant ist, müssen die Widerstände richtig gepolt sein, um Vorzeichenfehler und falsche Messungen zu vermeiden. Das Programm gibt vor, dass immer der obere Kontakt des Potentiometers auf 5V angeschlossen ist, der übrige auf GND. Bei den liegenden Potentiometern wird das selbe, nur gedreht angenommen, also links GND und rechts 5V. Da an dieser Stelle zweimal je 25 Drähte miteinander verbunden werden müssen, ist eine Form von Spannungsschiene stark zu empfehlen. Beim Prototypen wurden zwei Stücke einer querverbundenen Lochrasterplatte genutzt, auf der alle Verbindungen der entsprechenden Spannung zusammengeführt wurden. Die Anschlüsse für den ESP32 dürfen nicht vergessen werden. Beim Prototypen wurden Jumper-Wires genutzt, um den μC schnell austauschen zu können, doch bei einem permanenten Projekt können diese Verbindungen auch gelötet werden.

Auch die Multiplexer benötigten Spannung, daher müssen sie ebenfalls an der Spannungsschiene angeschlossen werden. der EN-Pin ist auf GND zu ziehen, da dieser einen negierten Enable-Pin darstellt, somit ist der Chip permanent aktiv, wenn auf EN 0V anliegen.

Die restlichen Pins des Multiplexers müssen wie in Tabelle 10.4 folgt mit dem ESP32 verbunden werden:

	ESP32 Pin	
Pin Name	Multiplexer 1	Multiplexer 2
S0	25	23
S1	33	22
S2	32	21
S3	12	19
Z	34	35
EN	26	5

Tabelle 10.4.: Löteinteilung der Multiplexer

Bevor der Controller in Betrieb genommen wird, sollte mit einem Ohmmeter der Widerstand zwischen den 5V und GND-Leitungen gemessen werden. Er sollte sich mindestens im zweistelligen Kiloohm-Bereich befinden, ansonsten fällt der ESP32 in einen Reset-Loop durch den Kurzschluss an der Versorgungsspannung.

Somit muss der μC nur noch mit der entsprechenden Software beaufschlagt werden und mit einem Handy und der Binobo-App verbunden werden, um den Controller in Betrieb zu nehmen. Zuletzt folgen noch zwei Bilder des vollständigen Controllers,

Abbildung 10.6 und 10.7.

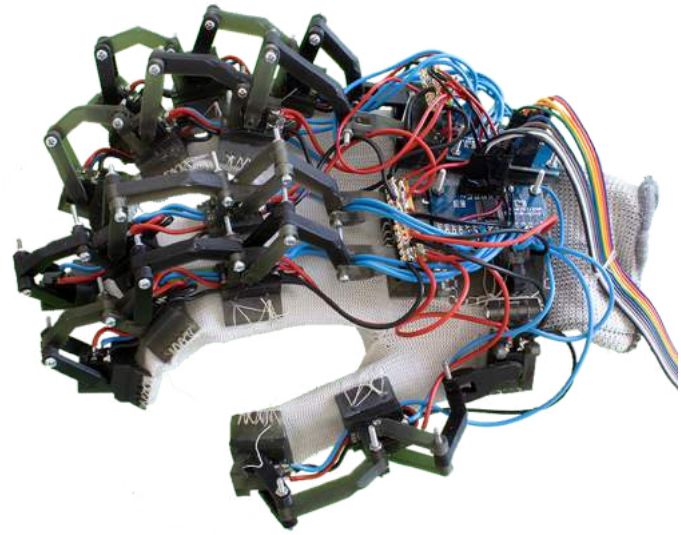


Abbildung 10.6.: Leerer, vollständiger Handschuh

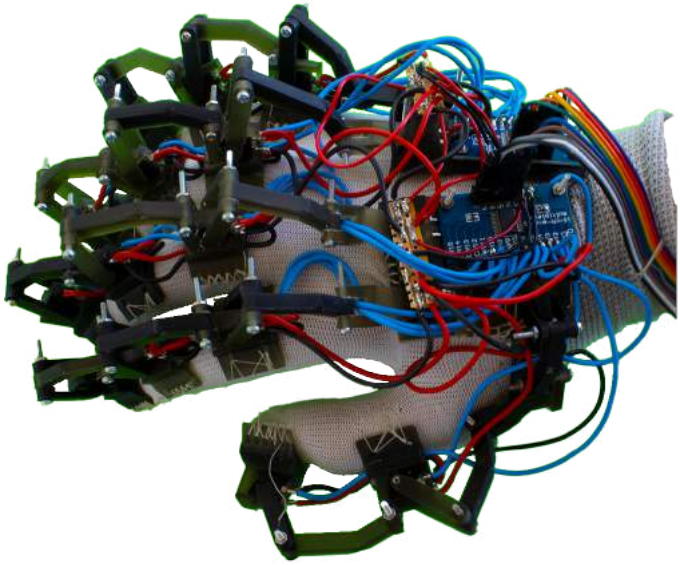


Abbildung 10.7.: Angezogener Handschuh

11. Ideensammlung zur Weiterentwicklung

11.1. Software

Das Projekt enthält zum aktuellem Stand zwar sämtliche Features, welche zu Beginn als Mindestmaß gesetzt worden waren, jedoch sind im Laufe der Entwicklung einige Feature-Wünsche aufgetaucht. Nachstehende Liste zeigt sämtliche Feature-Ideen, welche in Zukunft zu diesem Projekt hinzugefügt werden:

- `editor.md`-Plugin mit `editor.js` ersetzten
- Öffentliche Ansicht eines Nutzerkontos
- Persönliche Daten ändern können
- Passwort vergessen - Knopf zu Login hinzufügen
- OAuth2
- Firebase Login
- Neues Layout überlegen und mit React implementieren
- Eigener Cloud-Server um Bilder hochzuladen
- Searchbar für Blogeinträge
- Tags für Blogeinträge
- User blacklisten
- Live-Chat bei Emulator hinzufügen
- BeagleBone Blue statt ESP32 verwenden
- Setup-Anleitung für Controller

- Wählen zwischen BLE und Websocket Datenstreamen - Wenn BLE Verbindung, dann Nutzer fragen, ob diese Daten gebroadcastet werden sollen über eine Websocketverbindung
- Unity WebGL
- VR - Modus / Sterioskpoie-Modus
- visuelles Motiontracking mit OpenCV

11.2. Hardware

- BeagleBone Blue statt ESP32 verwenden
- Steuerbare bionische Roboterhand konstruieren
- Modell mit DMS
- Eigene Platine planen, um Kabelmenge zu reduzieren
- Verbesserte Handyhalterung
- Überarbeitung der Modelle für erhöhte Messgenauigkeit

A. Code Listings

A.1. Spring Boot

A.1.1. Spring Security

```
1  package htlstp.diplomarbeit.binobo.configurator;
2
3  import htlstp.diplomarbeit.binobo.controller.util.FlashMessage;
4  import htlstp.diplomarbeit.binobo.model.User;
5  import htlstp.diplomarbeit.binobo.service.UserService;
6  import org.springframework.beans.factory.annotation.Autowired;
7  import org.springframework.context.annotation.Bean;
8  import org.springframework.context.annotation.Configuration;
9  import org.springframework.security.config.annotation
10             .authentication.builders.AuthenticationManagerBuilder;
11 import org.springframework.security.config.annotation
12             .web.builders.HttpSecurity;
13 import org.springframework.security.config.annotation
14             .web.builders.WebSecurity;
15 import org.springframework.security.config.annotation
16             .web.configuration.EnableWebSecurity;
17 import org.springframework.security.config.annotation
18             .web.configuration.WebSecurityConfigurerAdapter;
19 import org.springframework.security.crypto.bcrypt.
      BCryptPasswordEncoder;
20 import org.springframework.security.crypto.password.PasswordEncoder;
21 import org.springframework.security.web.authentication
22             .AuthenticationFailureHandler;
23 import org.springframework.security.web.authentication
24             .AuthenticationSuccessHandler;
25
26 /**
27   * This class configures Spring's Security Plugin, telling it, how
        the
28   * Filter Chain hast to work, and which patters shall
29   * be ignored and which not.
30   * This class is annotated with @Configuration and
        @EnableWebSecurity,
31   * telling the JVM that it has to be evaluated
32   * on Runtime, as well as that this will configure the Spring
        environment
33   * and that Spring has to enable the Security Plugin.
```

```
34   */
35 @Configuration
36 @EnableWebSecurity
37 public class SecurityConfig extends WebSecurityConfigurerAdapter {
38
39     // Autowiring throws exception when changed to
40     // constructor-based-approach
41     @Autowired
42     private UserService userService;
43
44     @Autowired
45     public void configureGlobal(AuthenticationManagerBuilder auth)
46               throws Exception {
47         auth.userDetailsService(userService);
48     }
49
50     @Override
51     public void configure(WebSecurity webSecurity) throws Exception
          {
52         webSecurity.ignoring().antMatchers("/pictures/**", "/styles
              /**",
53                 "/scripts/**","/login/register",
54                 "/blog_rest_api/**", "/roboData/rest_api/**");
55                 // ignore theses patterns - spring security will not
                       get triggered
56                 // when trying to access one of this URIs
57     }
58
59     @Override
60     public void configure(HttpSecurity httpSecurity) throws
          Exception {
61         httpSecurity.authorizeRequests()
62                 .antMatchers("/home", "/project",
63                             "/developer", "/sponsoring")
64                     // permit all requests to this URIs
65                     .permitAll()
66                 .antMatchers("/blog/**", "/user/**", "/emulator3D")
67                     .hasAnyRole("USER", "ADMIN", "OPERATOR")
68                 .antMatchers("/admin/**").hasAnyRole("ADMIN", "
                       OPERATOR")
69                 // access needs to be authorized
70         .and()
71             .formLogin().loginPage("/login")
72             .permitAll()
73             .successHandler(loginSuccessHandler())
74             .failureHandler(loginFailureHandler())
75         .and()
76             .logout()
77             .permitAll()
78             .logoutSuccessUrl("/")
```

```
79          .and()
80              .csrf();
81      }
82
83      /**
84        * Authentication Success Handler - gets called,
85        * when login is successful
86        * @return Anonymous Interface implementation gets returned
87        */
88      public AuthenticationSuccessHandler loginSuccessHandler(){
89          return (request, response, authentication) -> {
90              // generate new token when user logs in
91              User user = (User)authentication.getPrincipal();
92              userService.generateNewTokenForUser(user);
93              response.sendRedirect("/blog");
94          };
95      }
96
97      /**
98        * Authentication Failure Handler - gets called, when
99        * login is unsuccessful
100       * @return Anonymous Interface implementation gets returned
101       */
102     public AuthenticationFailureHandler loginFailureHandler(){
103         return ((request, response, exception) -> {
104             request.getSession().setAttribute("flash_err",
105                     new FlashMessage("Incorrect␣data␣parsed!",
106                             FlashMessage.Status.FAILURE));
107             response.sendRedirect("/login");
108         });
109     }
110
111     /**
112       * Password encoder Bean
113       * @return returns an BCryptPasswordEncoder Object with
              strength 10
114       */
115     @Bean
116     public PasswordEncoder passwordEncoder() {
117         return new BCryptPasswordEncoder(10);
118     }
119
120 }
```

Listing A.1: Spring Security - Konfiguration

A.1.2. Emulator-Sourcecode

```
1   let scene;
2 let canvas;
3 let renderer;
4
5 function emu3D() {
6
7     //canvas setup
8     canvas = document.querySelector('#c');
9     renderer = new THREE.WebGLRenderer({antialias: true, canvas});
10    renderer.setSize(canvas.innerWidth, canvas.innerHeight);
11
12    //scene setup
13    scene = new THREE.Scene();
14    scene.background = new THREE.Color('#333');
15
16    //camera and controls setup
17    const fov = 45;
18    let aspect = canvas.innerWidth / canvas.innerHeight;
19    const near = 1;
20    const far = 10000;
21    let camera = new THREE.PerspectiveCamera(fov, aspect, near, far)
          ;
22    scene.add(camera);
23    const controls = new THREE.OrbitControls(camera, renderer.
          domElement)
24    controls.maxPolarAngle = degToRad(160);
25    controls.minPolarAngle = degToRad(20);
26    camera.position.set(500, 100, 200);
27    controls.update();
28
29    //lighting setup
30    const dir = new THREE.DirectionalLight(0xffffff, 1);
31    const dirm = new THREE.DirectionalLight(0xff8800, 0.5);
32    dirm.translateY(-10);
33
34    const ambient = new THREE.AmbientLight(0x404040, 2.5);
35    scene.add(dir);
36    scene.add(dirm);
37    scene.add(ambient);
38    dirm.target = dir;
39    //loader setup
40    let fbxloader = new THREE.FBXLoader();
41    let palm = scene;
42    let wrist = new THREE.Object3D()
43    const ppos = blenderToThree(new THREE.Vector3(0, 0, 0));
44
45    let frame = 0;
46    let path = '';
```

```
47
48      fbxloader.load('scripts/simulatorModels/hand_new/new_palm.fbx',
            async function (object) {
49
50          scene.add(wrist)
51          wrist.add(object);
52          object.position.x = -ppos.x;
53          object.position.y = -ppos.y;
54          object.position.z = -ppos.z;
55          palm = object;
56          palm.name = 'palm';
57
58          await loadFingers();
59      });
60
61      let fingers = [
62          [palm, palm, palm],
63          [palm, palm, palm],
64          [palm, palm, palm],
65          [palm, palm, palm],
66          [palm, palm, palm]
67      ];
68      /**
69       * This array is necessary in order to set proper rotation
             points for the imported models.
70       * The coordinates are copied from the corresponding .blend file
             .
71       * TODO: change this array from being hardcoded to being read
             from a .json file to enable the use of multiple models
             without code changes.
72       * @type {Vector3[][]}
73       */
74      const fpos = [ //DO NOT TOUCH!, new version
75          [
76              blenderToThree(new THREE.Vector3(3.59622, 0.353044,
                    -0.018004)),
77              blenderToThree(new THREE.Vector3(2.90712, 0.353044,
                    -0.000326)),
78              blenderToThree(new THREE.Vector3(1.98374, 0.355688,
                    -0.005718))
79          ], [
80              blenderToThree(new THREE.Vector3(3.78517, -0.121099,
                    -0.018709)),
81              blenderToThree(new THREE.Vector3(3.02742, -0.121099,
                    0.000276)),
82              blenderToThree(new THREE.Vector3(2.03528, -0.118192,
                    -0.005514))
83          ], [
84              blenderToThree(new THREE.Vector3(3.55936, -0.61488,
                    -0.018362)),
```

```
85              blenderToThree(new THREE.Vector3(2.84435, -0.61488,
                    -0.00002)),
86              blenderToThree(new THREE.Vector3(1.90818, -0.612137,
                    -0.005614))
87          ], [
88              blenderToThree(new THREE.Vector3(3.09757, -1.02955,
                    -0.016404)),
89              blenderToThree(new THREE.Vector3(2.52417, -1.02955,
                    -0.001695)),
90              blenderToThree(new THREE.Vector3(1.77339, -1.02735,
                    -0.006181))
91          ], [
92              blenderToThree(new THREE.Vector3(1.99089, 1.39046,
                    -0.004993)),
93              blenderToThree(new THREE.Vector3(1.38421, 1.16946,
                    -0.004993)),
94              blenderToThree(new THREE.Vector3(0.46195, 0.78016,
                    -0.004993))
95          ]]
96      ;
97
98      /**
99       * places a specific finger segment from the /simulatorModels
             folder
100      * @param object loaded object
101      * @param d digit
102      * @param s segment number
103      */
104     async function loadSegment(object, d, s) {
105         let point = new THREE.Object3D();
106         if (s < 3)
107             fingers[d - 1][s].add(point);
108         else
109             palm.add(point);
110         point.add(object)
111         fingers[d - 1][s - 1] = point;
112
113         point.position.x = fpos[d - 1][s - 1].x;
114         point.position.y = fpos[d - 1][s - 1].y;
115         point.position.z = fpos[d - 1][s - 1].z;
116         if (s < 3) {
117             point.position.x -= fpos[d - 1][s].x;
118             point.position.y -= fpos[d - 1][s].y;
119             point.position.z -= fpos[d - 1][s].z;
120         }
121     }
122
123     function loadTheFinger(resolve, reject) {
124         fbxloader.load(path, object => resolve(object));
125     }
```

```
126
127     /**
128      * loads a finger from the scripts/simulatorModels folder
129      */
130     async function loadFingers() {
131         for (let s = 3; s > 0; s--) {
132             for (let d = 1; d < 6; d++) {
133                 path = 'scripts/simulatorModels/hand_new/
                        new_simHand_id' + d + '' + s + '.fbx';
134                 let object = await new Promise(loadTheFinger);
135                 object.name = namePart(d, s);
136                 await loadSegment(object, d, s);
137             }
138         }
139     }
140
141     /**
142      * generates a name for a loaded segment
143      * @param d digit
144      * @param s segment number
145      * @returns {string} new object name
146      */
147     function namePart(d, s) {
148         return 'digit' + d + '␣segment' + s;
149     }
150
151     /**
152      * The almighty render function
153      */
154     function render() {
155         frame++;
156
157         let data;
158         try {
159             if (frame % 2 === 0) {
160                 data = apply_filter_to_data(data_buffer.shift());
161                 frame = 0;
162                 fingers[0][2].rotation.y = degToRad(-data[4]); //
                        if_base_rot
163                 fingers[0][2].rotation.z = degToRad(-data[5]); //
                        if_base
164                 fingers[0][1].rotation.z = degToRad(-data[6]); //
                        if_middle
165                 fingers[0][0].rotation.z = degToRad(-data[7]); //
                        if_tip
166                 fingers[1][2].rotation.y = degToRad(-data[8]); //
                        mf_base_rot
167                 fingers[1][2].rotation.z = degToRad(-data[9]); //
                        mf_base
```

```
168                 fingers[1][1].rotation.z = degToRad(-data[10]); //
                        mf_middle
169                 fingers[1][0].rotation.z = degToRad(-data[11]); //
                        mf_tip
170                 fingers[2][2].rotation.y = degToRad(-data[12]); //
                        rf_base_rot
171                 fingers[2][2].rotation.z = degToRad(-data[13]); //
                        rf_base
172                 fingers[2][1].rotation.z = degToRad(-data[14]); //
                        rf_middle
173                 fingers[2][0].rotation.z = degToRad(-data[15]); //
                        rf_tip
174                 fingers[3][2].rotation.y = degToRad(-data[16]); //
                        p_base_rot
175                 fingers[3][2].rotation.z = degToRad(-data[17]); //
                        p_base
176                 fingers[3][1].rotation.z = degToRad(-data[18]); //
                        p_middle
177                 fingers[3][0].rotation.z = degToRad(-data[19]); //
                        p_tip
178                 fingers[4][2].rotation.y = degToRad(-data[0]); //
                        th_rot_orthogonal
179                 fingers[4][2].rotation.x = degToRad(-data[1]); //
                        th_rot_palm
180                 fingers[4][1].rotation.y = degToRad(-data[2]); //
                        th_base
181                 fingers[4][0].rotation.y = degToRad(-data[3]); //
                        th_tip
182                 wrist.rotation.x = degToRad(0);//-data[20]); //
                        wr_lr
183                 wrist.rotation.z = degToRad(0);//-data[21]); //
                        wr_bf
184             }
185         } catch (e) {
186         }
187
188         camera.aspect = canvas.clientWidth / canvas.clientHeight;
189         renderer.setSize(canvas.clientWidth, canvas.clientHeight,
              false);
190         camera.updateProjectionMatrix();
191
192         requestAnimationFrame(render);
193
194         controls.update();
195         renderer.render(scene, camera);
196     }
197
198     window.requestAnimationFrame(render);
199 }
200
```

```
201 /**
202  * converts blender coordinates to three.js coordinates
203  * @param blenderCoords Coordinates from Blender
204  * @returns {Vector3} Coordinates for three.js
205  */
206 function blenderToThree(blenderCoords) {
207     let ret = new THREE.Vector3();
208     ret.x = blenderCoords.x * 100;
209     ret.y = blenderCoords.z * 100;
210     ret.z = blenderCoords.y * -100;
211     return ret;
212 }
213
214 /**
215  * converts rad to degrees
216  * @param rad angle in rad
217  * @returns {number} angle in degrees
218  */
219 function radToDeg(rad) {
220     return rad * 180 / Math.PI;
221 }
222
223 /**
224  * converts degrees to rad
225  * @param deg angle in degrees
226  * @returns {number} angle in rad
227  */
228 function degToRad(deg) {
229     return deg / 180 * Math.PI;
230 }
```

Listing A.2: Emulator-Sourcecode

B. Gesprächsprotokoll

Datum	Anwesende Schüler	Betreuer	Thema
07.09.2021	Dominik L., Philipp M.	Spilka R.	Anfangsgespräch
15.09.2021	Dominik L., Philipp M.	Damböck W.	Softwarearchitekur Besprechnung
16.09.2021	Dominik L., Philipp M.	Damböck W.	Spring Boot Einführung
15.10.2021	Dominik L., Philipp M.	Spilka R.	Besprechung bzgl. Potentiometer
29.10.2021	Dominik L., Philipp M.	Spilka R., Damböck W.	Erster Prototyp der Potentiometerhalterung
06.12.2021	Dominik L., Philipp M.	Spilka R., Damböck W.	Besprechung bzgl. Websocketserver
23.12.2021	Dominik L., Philipp M.	Spilka R.	Präsentation des ersten Prototypen des Controllers
19.01.2021	Dominik L., Philipp M.	Spilka R.	Besprechung zwecks Dokumentation
07.02.2021	Dominik L., Philipp M.	Spilka R.	Richtlinien bei Dokumentation

Tabelle B.1.: Gesprächsprotokoll

Abkürzungsverzeichnis

UART	Universal Asynchronous Receiver / Transmitter
SLA	Stereolithografie (3D-Druckverfahren)
FDM	Fused Deposition Modeling
MVC	Model-View-Controller
MVP	Model-View-Presenter
ESP32	Espressif 32
μC	Microcontroller
3D	Dreidimensional
USB	Universal Serial Bus
DMS	Dehnmessstreifen
UV	Ultraviolett
SQL	Structured Query Language
STOMP	Simple Text Oriented Messaging Protocol
WLAN	Wireles Local Area Network
HTML	Hypertext Markup Language
JDBC	Java Database Connectivity
ORM	Object Related Mapping
AOP	Aspect Oriented Programming
REST	Representational State Transfer
URL	Uniform Resource Locator
URI	Uniform Resource Identifier

JPA	Java Persistence API
API	Application Programmer Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
DDD	Domain-Driven-Design
JVM	Java Virtual Machine
POJO	Plain Old Java Object
SSL	Secure Sockets Layer
CLI	Command Line Interface
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IP	Internet Protocol
DDNS	Dynamic Domain Name Service
DNS	Domain Name Service
SDK	Software Development Kit
JDK	Java Development Kit
SpEL	Spring Expression Language
UUID	Universally Unique Identifier

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[1] **Java Spring Framework** https://spring.io

[2] **Python Websockets Library** https://websockets.readthedocs.io/en/stable/

[3] **PostgreSQL** https://www.postgresql.org

[4] **Android Java Framework** https://developer.android.com/docs

[5] **STOMP** http://stomp.github.io/stomp-specification-1.0.html

[6] **Hibernate** https://hibernate.org

[7] **Micropython** https://micropython.org

[8] **ESP32** https://www.espressif.com/en/products/devkits/esp32-devkitc

[9] **three.js Javascript Library** https://threejs.org

[10] **IntelliJ IDE** https://www.jetbrains.com/de-de/idea/

[11] **Spring Initializr** https://start.spring.io

[12] **Maven** https://maven.apache.org

[13] **Spring Security** https://spring.io/projects/spring-security

[14] **Spring Boot Data REST** https://spring.io/projects/spring-data-rest

[15] **Spring Data JPA** https://spring.io/projects/spring-data-jpa

[16] **Thymeleaf** https://www.thymeleaf.org

[17] **Spring Boot Starter Web** https://spring.io/guides/gs/spring-boot/

[18] **Spring Boot Validation** https://www.baeldung.com/spring-boot-bean-validation

[19] **Lombok** https://projectlombok.org

[20] **pip** https://pypi.org/project/pip/

[21] **Tomcat** https://tomcat.apache.org

[22] **TomEE** https://tomee.apache.org

[23] **MVC** https://de.wikipedia.org/wiki/Model_View_Controller

[24] **Seperation of Concerns** https://en.wikipedia.org/wiki/Separation_of_concerns

[25] **DDD** https://de.wikipedia.org/wiki/Domain-driven_Design

[27] **Asynchrone Programmierung** https://www.aleksundshantu.com/wiki/asynchrone-programmierung/

[28] **Dependency Injection** https://de.wikipedia.org/wiki/Dependency_Injection

[29] **Spring JDBC** https://spring.io/projects/spring-data-jdbc

[30] **Spring AOP** https://www.baeldung.com/spring-aop

[31] **REST** https://de.wikipedia.org/wiki/Representational_State_Transfer

[32] **JavaBeans** https://de.wikipedia.org/wiki/JavaBeans

[33] **SSL** https://de.wikipedia.org/wiki/Transport_Layer_Security

[34] **certbot** https://certbot.eff.org

[35] **Webjars** https://www.baeldung.com/maven-webjars

[36] **editor.md** https://pandao.github.io/editor.md/en.html

[37] **AJAX** https://developer.mozilla.org/de/docs/Web/Guide/AJAX

[38] **Websockets** https://de.wikipedia.org/wiki/WebSocket

[39] **asyncio-Library** https://docs.python.org/3/library/asyncio.html

[40] **socket-Library** https://docs.python.org/3/library/socket.html

[41] **uwebsocket-Library** https://github.com/danni/uwebsockets.git

[42] **MVP** https://de.wikipedia.org/wiki/Model_View_Presenter

[43] **Docker** https://www.docker.com

[44] **docker-compose** https://docs.docker.com/compose/

[45] **Reverse-Proxy** https://de.wikipedia.org/wiki/Reverse_Proxy

[46] **nginx** https://www.nginx.com

[47] **IIR-Filter** https://de.wikipedia.org/wiki/Filter_mit_unendlicher_Impulsantwort

[48] **Exponentieller Filteralgoritmus** https://tttapa.github.io/Pages/Mathematics/Systems-and-Control-Theory/Digital-filters/Exponential%20Moving%20Average/Exponential-Moving-Average.html

[49] **Android Studio** https://developer.android.com/studio

[50] **Spring Boot Webserver Github** https://github.com/psykovski-extended/binobo

[51] **Python Websocketserver Github** https://github.com/psykovski-extended/binobo_websocket_server

[52] **Moon Modeler** https://www.datensen.com/data-modeling/moon-modeler-for-databases.html

[53] **Blender** https://www.blender.org

[54] **Autodesk Inventor** https://www.autodesk.de/products/inventor/overview?term=1-YEAR&tab=subscription

[55] **Autodesk Fusion 360** https://www.autodesk.de/products/fusion-360/overview