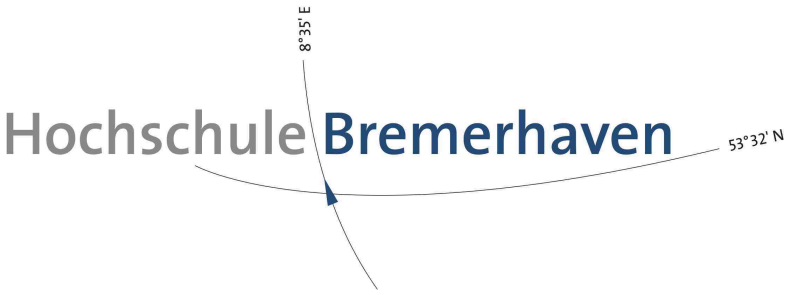
Bachrlorprojekt: Pepper



Eingereicht von:

Benjamin Thomas Schwertfeger 36036

Kristian Kellermann 35751

Jacob Menge xxxxx

Kursleitung: Prof. Dr. Nadja Petram

Technik für Wirtschaftsinformatik

Hochschule Bremerhaven

Eigenständigkeitserklärung

Hiermit erklären wir, dass wir die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Alle sinngemäß und wörtlich übernommenen Textstellen aus fremden Quellen wurden kenntlich gemacht.

Name: _____ Unterschrift: _____
Ort, Datum: _____

Name: _____ Unterschrift: _____
Ort, Datum: _____

Name: _____ Unterschrift: _____
Ort, Datum: _____

Inhaltsverzeichnis

Abbildungsverzeichnis

Abstrakt

Diese Dokumentation ist im Rahmen des Bachelorprojektes von Benjamin T. Schwertfeger, Jacob Menge und Kristian Kellermann entstanden und dient neben der Dokumentation des selbst gewählten Projektes ebenfalls als Anforderung für das Bestehen des Bachelorstudiums.

Die Studenten haben sich dazu entschieden, eine Anwendung für den humanoiden Roboter Pepper, welcher sich derzeit im Besitz der Hochschule Bremerhaven befindet zu entwickeln. Hierzu werden verscheidene Anwendungsfälle implementiert, welche durch eine selbst implementierte Web-Schnittstelle in Form einer Backend Webanwendung unterstützt wird.

Ziel ist es, dem Roboter das Interagieren mit Menschen beizubringen, sodass sinnvolle Interaktionen zwischen Mensch und Maschine zustande kommen. Zudem sollen mit Hilfe der Webanwendung zusätzlich dynamisch Informationen an Pepper wetergegeben und von ihm abgerufen werden, sodass Informationen zu Interaktionen, wie Dauer, Intensität und Erfolg des Gespräches gespeichert und ausgewertet werden können.

Dieses Projekt wird von Frau Prof. Dr. Nadja Petram begleitet. Es wurdne keine Rahmenbedingungen festgelegt.

Kapitel 1

Dokumentation und Struktur

Diese Dokumentation ist in XXX Kapitel gegliedert, welche sich mit den einzelnen Aspekten unseres Projektes auseinandersetzen. Die Stuktur sieht wie folgt aus:

- *Kapitel 1*: Dokumentation und Struktur
- *Kapitel 2*: Motivation und Zielsetzung
- *Kapitel 3*: Erste Schritte und Installation
- *Kapitel 4*: Anwendungsfall - Hochschule
- *Kapitel 5*: Anwendungsfall - XXX
- *Kapitel 6*: Das Backend
- *Kapitel 7*: Möglichkeiten der Erweiterung
- *Kapitel 12*: Abschließende Wort

Zu sehen ist, dass wir nach der Erläuterung unserer Struktur und des Grundes für diese Arbeit von vorn bis hinten durch unser Projekt gehen. Nachdem wir unser Grundgerüst und unsere ersten Schritte einschließlich Installation dargelegt haben, werden wir zwei Anwendungsfälle zum Einsatz von Pepper Aufzeigen, sowie die von uns Implementierten Funktionen aufzeigen.

Ebenfalls werden wir die Herangehensweise, sowie die technische Umsetzung und Hürden, mit denen wir zu kämpfen gehabt haben erläutern, um alle Aspekte der Implementierung von Software für den Roboter Pepper abzudecken.

Nachfolgend haben wir die Fragestellungen festgehalten, auf welche wir in den jeweiligen Abschnitten eingehen werden.

Kapitel	Fragestellung
1. Dokumentation und Strktur	Wie ist diese Arbeit aufgebaut?
2. Motivation und Zielsetzung	Was wollen wir hiermit erreichen?
3. Erste Schritte und Installation	Was ist das Fundament unseres Softwareprojektes?
4. Anwendungsfall - Hochschule	Wie kann Pepper der Hochschule helfen?
6. Node - Express Webanwendung	Welche Prozesse laufen im Hintergrund?
6. Skripte und Erweiterungen	
7. Skripte und Erweiterungen	
12. Abschließende Worte	Was haben wir aus diesem Projekt mitgenommen?

Im Anschluss dieser Dokumenation ist das Literaturverzeichnis mit allen Referenzen zu finden.

ES GIBT REPOSITORIES, IN DENEN ALLES EINSEHBAR IST, WENN DIE NICHT MEHR ZUR VERFÜGUNG STEHEN, HAT DIE HOCHSCHULE DEN GESAMTEN KRAM. DER QUELLCODE IST WEITESGEHEND DOKUMENTIERT WIR VERZICHTEN HIER GRÖßTENTEILS AUF DIE DARSTELLUNG DES QUELLCODES, DA ES ZU VIEL WÄRE. AN MANCHEN STELLEN ZEIGEN WIR GEWISSE DINGE AUF, DIES IST ABER NUR EIN KLEINER TEIL UND KANN IN DEN ABGEGEBENNEN DATEIN UND ODER AUF GITHUB EINGESEHEN WERDEN

Kapitel 2

Motivation und Zielsetzung

Wie alle Bachelorstudierende kommt irgendwann die Zeit, in der es darum geht, ein Projekt zu finden, mit welchem man sich über einen längeren Zeitraum beschäftigt, um dies im Rahmen des Studiums schriftlich niederzulegen. Wir drei sind von der Welt der Programmierung begeistert und haben von Frau Dr. Petram das Angebot erhalten, bei ihr das Bachelorprojekt durchzuführen. Da wir im Laufe unserers Studiums viele Kurse bei ihr besucht haben, unter anderem “KI - maschinelles Lernen“ und “Big Data“ waren wir sofort begeistert, als wir gehört haben, mit dem humanoiden Roboter Pepper arbeiten zu dürfen.

Wir haben von ehemaligen Masterstudierenden eine Vorführung einer laufenden Anwendung bekommen und haben uns natürlich direkt gefragt, was wir denn tolles entwickeln können, damit nicht nur wir, sondern auch die Hochschule den meisten Mehrwert davon erhalten.

Aufgrund dieser Fragestellung haben wir uns dazu entschieden, dass wir zwei Anwendungsfälle zum Einsatz des Peppers implementieren wollen, um diese auch mit Hinblick auf den Tag der Informatik vorstellen zu können.

Neben der Entwicklung dieser Anwendungsfälle wollen ebenfalls unser Verständnis für die Thematik KI und Big Data ausbauen, indem wir ein Backend schaffen, welches mit Hilfe von Pepper Daten sammelt, um diese in einem späteren Prozess anderweitig zu verwenden.

Kapitel 3

Erste Schritte: Das Grundgerüst

3.1 Allgemein

Wir haben bisher noch kaum Berührung mit Robotern im Alltag erlebt und daher ist unsere Erfahrung im Bereicht der Roboterprogrammierung sehr begrenzt. Zum Glück haben uns ehemalige Masterstudierende einen schnellen Einstieg ermöglicht, denn diese habe eine Anwendung für das Unternehmen “Erlebnis Bremerhaven“ entwickelt und uns deren Quellcode zur Verfügung gestellt.

Wir haben sehr schnell gemerkt, dass uns dies zwar eine gute Hilfe für den Anfang ist, jedoch gibt es mittlerweile deutlich effizientere Methoden einzelne Komponenten miteinander zu verknüpfen.

3.2 Wer ist dieser Pepper?

3.3 Projektmanagement

Um dieses Projekt erfolgreich durchzuführen und zielorientiert zur arbeiten, treffen wir uns wöchentlich in der Hochschule und Besprechen unsere neuesten Ideen und Implementierungen. Auch zwischendurch stehen wir im Kontakt, um Probleme und Schwierigkeiten schnell zu beheben.

Damit wir gemeinsam an einer Codebasis arbeiten können, haben wir zu Anfang unseres Projektes die Organisation HBV-Pepper auf GitHub angelegt. Jegliche Beteiligung, sowie verschiedene Versionen können dort eingesehen werden. Zum Ende des Projektes haben sich hier 4 Repositories von uns angesammelt, welche folgende Themen beinhalten:

- Pepper Anwendung
- Backend Serveranwendung
- Dokumentation

- Sonstiges

Das erste Repo beinhaltet unsere lauffähige Anwendung, welche die von uns implementierten Anwendungsfälle, sowie sonstige Funktionalitäten für den Roboter Pepper beinhaltet.

Der Backend Serveranwendung auf welchem wir in Kapitel ¡HIER LINK SETZEN¿ eingehen, ist unsere Schnittstelle, mit welcher wir Informationen, die Pepper durch seine Interaktionen mit der Umgebung sammelt, abfangen und speichern. Zudem Versorgen wir Pepper mit Informationen, Beispiel hierfür sind Informationen zum Stunden- und Mensaplan.

Das Repositorie zur Dokumenation beinhaltet alle zu diesem Dokument gehörenden Dateien. Da wir diese mit LaTeX anfertigen, lohnt es sich auch dies zu versionieren.

Für kleine Skripte, Tests und andere zu keinem festen Thema dazugehörigen Dokumente haben wir das Repositorie “Sonstiges“ angelegt.

Wir werden in diesem Dokument neben der Implementierung auch auf das Herunterladen, Installieren und Einrichten unserer Software eingehen, damit Studierende, die dieses Projekt weiterführen wollen, die Möglichkeit haben, mit einem etwas fortgeschrittenem Projekt zu starten. Darüber hinaus befindet sich in jedem Stammverzeichnis eine Readme, welche Anforderungen und erste Schritte aufzeigen, um einen schnellen Einstieg zu ermöglichen.

3.4 Persona

Damit wir Pepper mit allen möglichen Antworten auf alle möglichen Fragen vorbereiten können, müssen wir uns zuerst überlegen, welche Personengruppe mit Pepper sprechen wird. Auf dieser Basis können wir verschiedene Gesprächsthemen entwickeln, die für die Personengruppe interessant sein könnten.

Da wir Pepper als Hochschul Assistenzen einsetzten wollen, haben wir es mit folgenden Personas zu tun:

- Student
- Dozent
- Angestellter der Hochschule
- Besucher (Zukünftige Studenten, Interessenten)

Hauptsächlich soll Pepper für Studenten und insbesondere für Erstsemestler eingesetzt werden, die zum Beispiel nicht wissen, wo sich ein bestimmter Raum oder ein Gebäude befindet oder wann und wo die nächste Vorlesung stattfindet. Hierbei soll es auch die Möglichkeit geben, den kompletten Stundenplan für jeden Studiengang und jedes (zur

Zeit verfügbare) Semester anzeigen zu lassen, um auf jede Situation eine Antwort parat zu haben.

Ein weiterer Faktor ist der Altersintervall mit dem Pepper zu tun haben wird. Ältere Studenten, die sich auch in einem höheren Semester befinden, sind Themen wie Raumfindung oder Stundenplan eher uninteressant. Dafür könnten Themen wie der Mensaplan oder Informationen über die Hochschule und über Pepper selber interessant sein. Außerdem wird Pepper in der Lage sein, mit dem User ein kurzes Smalltalk Gespräch zu führen. Somit ist Pepper auch für alle Altersgruppen interessant. Diese Themen könnten für alle weitere Personas ebenfalls interessant sein. Vor allem Personen, die technikaffin sind oder das Thema Roboter einfach spannend finden, werden Pepper ebenfalls verwenden, weswegen wir für diese Personengruppe passende Gesprächsthemen und lustige Animationen vorbereiten.

Für Dozenten und Angestellte der Hochschule sind Themen wie der Stundenplan und die Raumfindung eher uninteressant und der Mensaplan wieder interessanter. Diese beiden Personas werden Pepper sehr wahrscheinlich am wenigsten verwenden.

Für Besucher sind Themen wie Informationen über Studiengänge und allgemein zur Hochschule eher interessanter als alle anderen Themen.

Um genau sagen zu können, welche Personengruppe sich mit Pepper unterhalten und über welche Themen sie sprechen, müssen wir Pepper zuerst sehr oft in der Praxis einsetzten und verschiedene Daten sammeln. Dazu werden wir im Punkt “Datensammlung” näher eingehen.

3.5 Datensammlung

Sobald alle Anwendungsfälle in Pepper integriert sind und er für den Praktischen Nutzen einsatzbereit ist, haben wir uns überlegt, verschiedene Daten zu sammeln um eine Art Feedback zu erhalten und diese graphisch darzustellen.

Folgende Daten sollen gesammelt werden:

- Distanz
- Alter
- Geschlecht
- Emotion/Stimmung
- Mimic
- Dialog Zeit
- ...

Diese Daten sollen während des Gesprächs zwischen Pepper und dem User gesammelt werden und zu unserem Node Server gesendet werden. Auf dem Node Server werden diese anschließend weiter verarbeitet und vorerst in einer MySQL Datenbank abgespeichert.

Um die Daten sammeln zu können, bietet Softbanks praktischerweiße verschiedene KI orientierte Funktionen an, wie zum Beispiel die Gesichts,- oder Alterserkennung.

Diese Daten sollen dafür eingesetzt werden, um Pepper in Zukunft weiter zu optimieren und folgende Fragen zu beantworten:

- War das Gespräch erfolgreich und konnte Pepper helfen?
- Hatte der User freude bei der Benutzung des Roboters?
- Hatte der User Schwierigkeiten, sich mit dem Pepper zu unterhalten oder verlief alles reibungslos?
- Welche Personengruppe hat mit Pepper am meisten gesprochen?
- Wie lange verläuft ein Gespräch im Durchschnitt?
- Gab es Fragen, die nicht beantwortet werden konnten?
- Welcher Anwendungsfall wurde wie oft verwendet?
- Gab es Systemfehler?
- Was ist die optimale Ansprech Distanz zwischen User und Roboter?
- Welches Semester und/oder Studiengang ist der User

Kapitel 4

Die Pepper App für die Hochschule

HIER ERKLÄREN, DASS WIR VON ROS NICHTS WUSSTEN UND DIES MIT UNSEREM PEPPER NICHT MÖGLICH IST - ABER AUFPASSEN, DASS DIES NICHT MIT DEM ERSTEN ABSATZ IM NODE KAPITEL IN KONFLIKT TRITT

4.1 Die Vorüberlegungen

Nachdem wir uns darauf geeinigt haben, mindestens einen Anwendungsfall für den Roboter Pepper zu entwickeln, und sich dieser auf den Alltag an der Hochschule Bremerhaven beziehen soll, haben wir uns damit konfrontiert gesehen, uns Gedanken darüber zu machen, wie genau eine Interaktion zwischen einem Studenten oder einer Lehrkraft und dem Roboter stattfinden kann. Aufgrund der den aktuellen Beschlüssen der Regierung ist das Leben auf dem Campus fast zum Stillstand gekommen, wodurch sich für Pepper wenige Einsatzgebiete ergeben. Wir gehen jedoch davon aus, zu einer neuen Normalität zurück zu gekangen, und werden Pepper dahingehend vorbereiten.

Folgende Fragen sind für uns zentral:

- Wie sieht der Alltag auf dem Campus aus?
- Wann soll Pepper mit wem interagieren?
- Welche Funktionalitäten wollen wir implementieren?
- Wo setzen wir welche Werkzeuge ein?
- Wo fangen wir an?

Wir sind dann ziemlich schnell darauf gekommen, dass wir eine klare Struktur benötigen, um unsere Aufgaben und Ziele klar zu definieren. Daraufhin ist folgende Skizze entstanden, welche grob die Hauptfunktionalitäten unseres Projektes festhält:

HIER WIRKLICH EINE SKIZZE ANFERTIGEN

In Abb. ist zu sehen, dass wir folgende Grundfunktionalitäten definiert haben:

- Stundenplan
- Raum- und Lageplan
- Speisekarte der Mensa
- Smalltalk

Zudem soll die Interaktion mit Pepper für alle möglich sein. Studierende und Lehrende sollen erfragen können, wo welche Vorlesung stattfindet, in der Kaffeteria oder Mensa soll sich Pepper mit Gästen unterhalten, Smalltalk führen oder Empfehlungen für bestimmte Speisen ausgeben. Darüber hinaus wollen wir Pepper das lückenlose Interagieren beibringen, damit er an besonderen Veranstaltungen, wie dem Tag der offenen Tür oder dem Tag der Informatik, Interessenten Informationen über die Hochschule, das Leben auf dem Campus, sowie allgemeine Empfehlungen für die Stadt Bremerhaven ausgeben kann.

4.2 Hard- und Softwarespezifikationen

Da Peppers Tablet auf Android basiert, liegt es auf der Hand, Android Studio als Entwicklungsumgebung zu benutzen. Der Einfachheit halber, ist nachfolgend eine Tabelle, welche die von uns für die Pepper Anwendung benutzte Software mit den entsprechenden Versionsnummern auflsitet.

Tabelle 4.1: Hard- und Softwarespezifikationen

Software / Tool	Version / Spezifikation	Beschreibung
Android Studio	Arctic Fox 2020.3.1 Patch 4	IDE der IntelliJ Plattform
Gradle	7.0.3	Build Tool
Android SDK	12	Android Framework
Pepper API	v1.9 API 23	Entwicklungstools für Pepper mit Emulator, Deploy und Debug Funktion
Android API SDK	31	Framework zum Verbinden und Installieren von Software auf Geräten mit Android OS
Java SDK	1.8	Basis der Programmiersprache Java und dessen Grundfunktionalitäten

Mit abwichenden Versionen kann es zu Kompatibilitätsproblemen kommen.

Der Roboter Pepper selbst, ist XXX groß, wurde am XXX von XXX gekauft und ist seit XXX im Besitz der Hochschule und läuft derzeit mit der Version XXX.

4.3 Implementierung

hier muss auch erwähnt werden, dass pepper selbst das alter, stimmung und geschlecht erkennt

Kapitel 5

Node - Express Webanwendung

5.1 Allgemein

Wir als Studenten haben durch das Studium, aber auch durch Nebenjobs und private Projekte Erfahrungen mit verschiedenen Programmierkonzepten sammeln können. Da wir bei der Entwicklung der Pepper Anwendung jedoch nur zwischen den Sprachen Kotlin und Java entscheiden konnten, haben wir zu Beginn unseres Projektes, das Gefühl gehabt, in unserer Entwicklung, was die Nutzung und Implementierung verschiedener Methoden angeht, sehr eingeschränkt zu sein.

Daher haben wir im Verlauf unseres Bachelorprojektes immer wieder gemerkt, dass mit dem Pepper allein nicht viel anzufangen ist, wenn es um Flexibilität in der Entwicklung geht. Zugriff auf Sensoren, sowie die Kamera und die Mikrophone sind nicht direkt möglich, sondern nur über vordefinierte Funktionen der Aldebaran Bibliotheken. Somit haben wir keine Möglichkeit, selbst Gesichter zu erkennen oder Sprache zu analysieren, geschweige denn mehr als nur mit Pepper reden zu können.

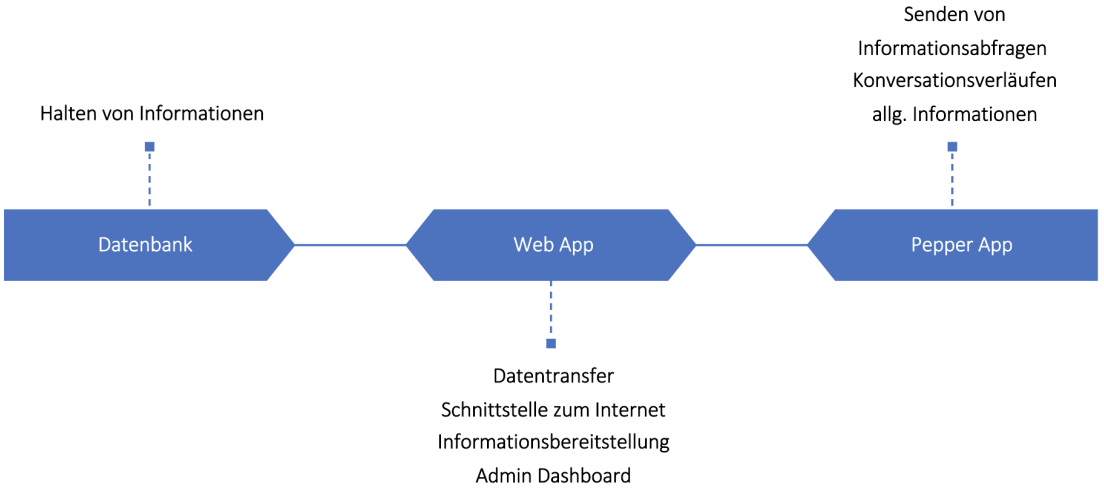


Abbildung 5.1: Zusammenhang: Pepper App und Backend

Bei der Implementierung der Stundenpläne etwa, werden verschiedene Anfragen an die Hochschulserver geschickt und ausgewertet, jedoch ist auch dies in Java nicht wirklich entwicklerfreundlich. Daher sind wir auf die Idee gekommen, einen Webserver aufzusetzen, den Pepper kontaktieren kann, um weiterführende Informationen zu erhalten.

Genau hier war es uns von Vorteil, Erfahrungen mit Node, Datenbanken und API's gesammelt zu haben. Daher ist es ein leichtes Unterfangen gewesen, einen Express Webserver mit Hilfe der Laufzeitumgebung Node aufzusetzen, auf welchem wir vielfältige Möglichkeiten zur Implementierung von komplexen Vorgängen haben.

So hat es sich ergeben, dass wir nicht nur eine Anwendung für Pepper entwickelt haben, sondern einen zweiten Schwerpunkt, welcher sich auf das Sammeln und Auswerten von Daten konzentriert, definiert haben. Diese Daten sollen natürlich von Pepper über unsere Anwendung gesammelt werden.

Das Repository dieser Webanwendung ist derzeit noch öffentlich und unter folgender Adresse erreichbar: https://github.com/ProjectPepperHSB/NodeJS_Server4Pepper. Sollte dieser dieser Link nicht mehr funktionieren, kann sich an die Hochschule gewandt werden, denn auch dort werden wir unseren Quellcode eingereicht haben.

5.2 Überblick: Webanwendung

Bevor wir im Folgenden auf die Implementierung und Spezifikationen unserer Webanwendung eingehen, wollen wir hier kurz die Hauptfunktionalitäten, welche wir in drei Kategorien aufgeteilt haben, aufzeigen.

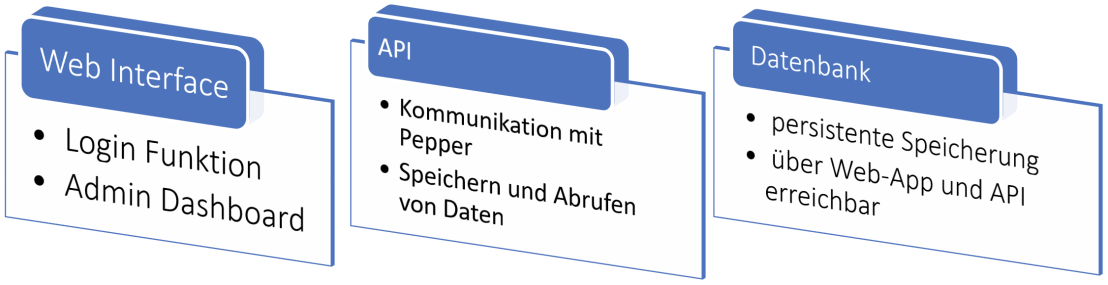


Abbildung 5.2: Komponenten der Webanwendung

Diese Webanwendung, sowie die dazugehörige Datebank laufen derzeit auf dem Hochschulserver Hopper, in einem abgeschirmten Docker-Kontainer, welcher einem Benutzer mit Namen "hbv-kms" gehört. Dies ist ein Benutzer, auf welchen wir gemeinsam als Team zugreifen können. Da die Anwendung über den HTTP Port des Dockers läuft, findet sich der Pfad `docker-hbv-kms-http` in allen Endpunkten dieser Webanwendung wieder.

Diese Anwendung ist jedoch auch auf fast jedem lokalen Rechner ausführbar, sofern eine funktionsfähige MySQL Instanz aktiv ist. (vgl. Abschnitt 5.4)

5.2.1 Web Interface

Das Web Interface dient dazu, den Entwicklern und Betreibern der Webanwendung, Informationen über die gesammelten Daten aufzuzeigen. Da wir es für sinnvoll halten, nicht jedem den kompletten Zugriff auf die gesammelten Daten zu gewähren, haben wir uns dazu entschieden, nur einen Benutzer in unserer Webanwendung anzulegen, welcher Zugriff auf das Admin Dashboard hat. Somit sparen wir uns die Implementierungen zur Registration und Verwaltung von Nutzern (mehr in Abschnitt 5.4).

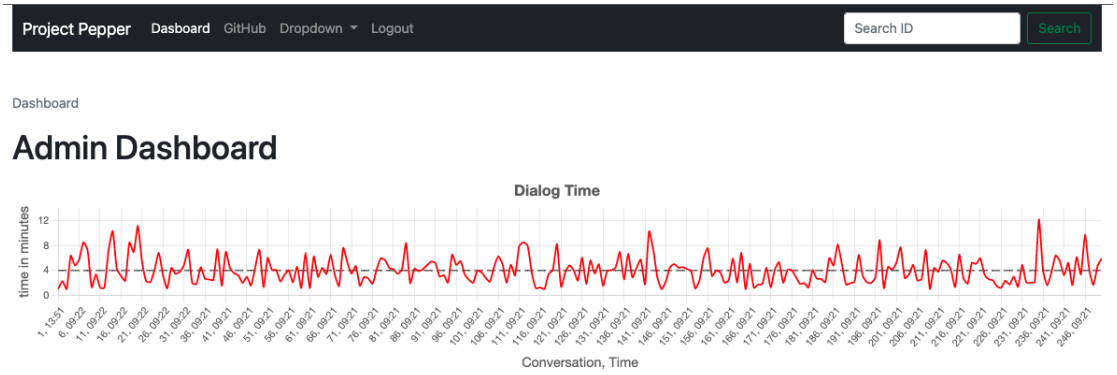
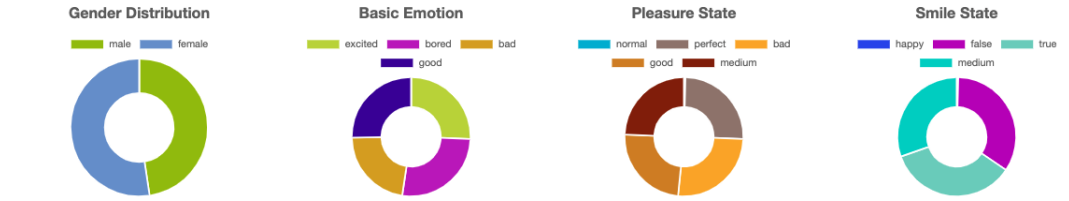


Abbildung 5.3: Webansicht des Admin Dashboards Teil 1

(Abb. 5.3) Sofern sich ein Administrator angemeldet hat, gelangt er in die Ansicht des Admin Dashboards. Dieses besteht aus mehreren Komponenten. Zu sehen ist, dass sich am oberen Bereich der Abbildung eine Menüleiste befindet. Diese Beinhaltet eine Verlinkung zu unseren derzeit öffentlichen GitHub Repositories, sowie weitere Dropdowns, welche bisher noch keine Funktionen besitzen. Dies ist je nach Anwendungsfall und Einsatzgebiet individuell erweiterbar. Die dargestellte Suchleiste funktioniert jedoch. Über diese können nach speziellen Identifikationsnummern einzelner Konversationen und Datenreihen gesucht werden. (vgl. Abschnitt 5.3.1.3 f.)

Des Weiteren ist ein Liniendiagramm abgebildet. Dies zeigt die Dauer der letzten 250 Konversationen, welche von Pepper geführt und anschließend über unserer Webanwendung in die Datenbank gespeichert wurden. Dieses Diagramm ist jedoch nicht aussagekräftig, da wir zur Füllung unserer Datenbank, Skripte zur randomisierten Generierung von Konversationen erstellt und ausgeführt haben. Mehr dazu in **HIER VERLINKUNG ZUM SKRIPTE SUBSECTION**.

Wir liefern immer nur die letzen 250 Konversationen aus, da Pepper, falls er denn mal zum Einsatz kommt, gar nicht so sehr viele Konversationen führen wird, da diese auch eine gewisse Zeit in Anspruch nehmen. Somit ist es nur sinnvoll, sich die nur letzen 250 anzeigen zu lassen, um einen aktuellen Überblick über die vergangenen Interaktionen zu erhalten. Über die Suchleiste kann dennoch jede Konversation gefunden werden.



#	Distance	Age	Gender	Basic Emotion	Pleasure State	Excitement State	Smile State	Dialog Time	TimeStamp
1	0.823 meters	24	male	excited	normal	medium	happy	1m 1s	2022-01-28T13:51:42.000Z
2	0.5363 meters	66	male	bored	perfect	excited	false	2m 17s	2022-01-25T09:22:07.000Z

Abbildung 5.4: Webansicht des Admin Dashboards Teil 2

Zusätzlich zur Darstellung der Konversationsverläufe haben wir uns entschieden, weitere interessante Informationen zu den zur Grunde liegenden Gesprächen visuell hervorzuheben. Somit ist es möglich, sich einen schnellen Überblick über die letzen Interaktionen mit Pepper zu verschaffen.

(Abb. 5.4) Sprechen mehr männliche oder weibliche Personen mit Pepper? Wie ist die Stimmung unserer Akteure und wie verhalten sie sich? Wird gelacht oder sind sie gelangweilt? Diese Informationen können anhand der Donut-Diagramme abgelesen und interpretiert werden. So hat man beispielsweise Auskunft darüber, ob ein neues Update bei den Kunden und Anwendern gut ankommt, oder ob man nicht doch noch etwas ausbessern sollte. Zudem ist auf unseren Anwendungsfall Hochschule bezogen, auch nachvollziehbar, wie die allgemeine Stimmung der Studenten und Interessierten ist.

Unterhalb den Diagrammen beginnt eine Tabelle, welche die ausgewerteten Datenreihen beinhaltet. Jede dieser Reihen führt per Klick auf eine Detailansicht der jeweiligen Konversation und bietet somit einen genaueren Einblick in das geführte Gespärch.

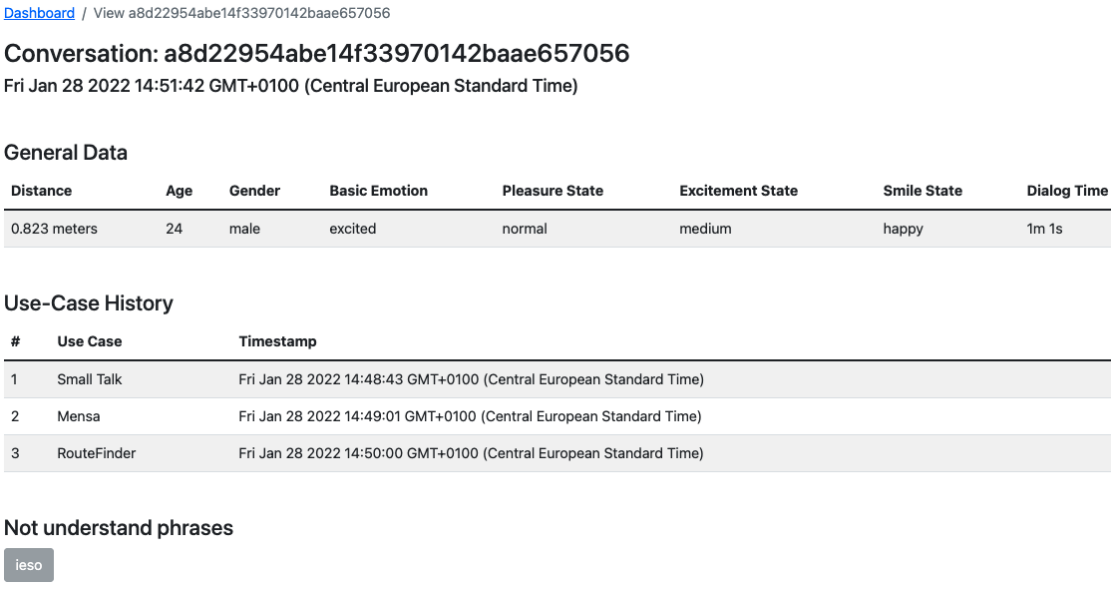


Abbildung 5.5: Ansicht einer bestimmten Konversation

(Abb. 5.5) Jeder Konversation, die Pepper führt, wird eine Identifikationsnummer zugewisen. Dies ermöglicht die dynamische Zuordnung von Inhalten zu einem bestimmten Gespräch. Somit ist eine neben der Einsicht in die allgemeinen Informationen wie dem Alter und Geschlecht des Gespärchspartners möglich, auch Informationen über verwendete Anwendungsfälle und deren chronologischer Reihenfolge zu erhalten. Auch Phrasen, welche Pepper nicht verstanden hat, sei es aufgrund von Akzenten oder falscher Aussprache, werden der Konversation zugewiesen.

Dies alles ermöglicht es uns und dem Anwender der Software, mehr über seine Kunden und Anwender zu erfahren - oder auf die Hochschule bezogen: Die Stimmung der Studenten, sowie des Personals und Interessierten kann somit erfasst werden.

Alle Endpunkte unserer Webanwendung, die für das Admin Dashboard benötigt werden, setzen voraus, dass der Benutzer als Admin angemeldet ist und ein gültiges JSON Web Token in seinem Cookie besitzt.

5.2.2 API

Die Kommunikation mit unserer Webanwendung verläuft nicht nur über das Web Interface, sondern über verschiedene Endpunkte, auch Routes genannt. Über diese können je nach Grad der Authentifizierung verschiedene Inhalte gesendet und empfangen werden.

Um hier nicht zu viele Informationen aus den Bereichen der Endpunkte (5.3) und Implementierung (5.4) vorweg zu nehmen, wollen wir es bei der Erwähnug und Beschreibung der wichtigsten Endpunkte belassen.

- /docker-hbv-kms-http/fileserver
- /docker-hbv-kms-http/api/v1/sql
- /docker-hbv-kms-http/api/save[...]Data

(vgl Abschnitt 5.3.2.9) Wir haben aufgrund der Struktur unserer Serverkonfiguration zunächst Schwierigkeiten gehabt, statische Dateien über HTTP auszuliefern, da sich die Pfade zwischen der lokal laufenden Instanz von denen der auf dem Hopper laufenden unterscheiden. Daher haben wir einen Endpunkt `/docker-hbv-kms-http/fileserver` erstellt, welcher uns dynamisch die gewünschte Datei Pfadunabhängig ausliefern kann. Hierbei wurde auch berücksichtigt, dass nur bestimmte Dateien ausgeliefert werden können. Es wäre fatal, wenn man dies nicht beachtet und in einer Produktivumgebung alle Dateien des Servers, einschließlich der Zertifikate ausliefern würde.

(vgl. Abschnitt 5.3.2.5) Der Endpunkt `/docker-hbv-kms-http/api/v1/sql` ist die Schnittstelle über welche man extern auf Daten in der Datenbank zugreifen kann. Hier ist es erforderlich, authentifiziert zu sein. Dafür muss ein Auth-Token bei jeder Abfrage übermittelt werden, welches mit dem Token, welches in unserer Vebanwendung hinterlegt ist, verglichen wird. Um nicht das Problem des öffentlichen Einsehens unseres Tokens

durch Github haben, arbeiten wir mit .env Dateien, welche während der Initialisierung der Webanwendung geladen wird. Diese .env Datei ist nicht auf Github, jedoch existiert eine Beispieldatei mit namen .example.env, welche das Format und die nötigen Felder vorgibt.

Der letzte oben aufgeführte Endpunkt, steht symbolisch für eine Reihe von Endpunkten, welche für die Speicherung verschiedener Informationen, die Pepper während seiner Gespräche sammelt und an unserern Server übermittelt, verantwortlich ist. Diese Endpunkte sind sowohl über GET , als auch über POST Anfragen ansprechbar und nehmen verschiedene Parameter entgegen. Bisher muss man sich für diese Endpunkte nicht authentifizieren, das heißt, dass jeder in unsere Tabellen schreiben kann, sofern er die richtigen Parameter kennt. Wir haben überlegt dies zu umgehen, indem wir Pepper ebenfalls ein Token generieren lassen, welches als zur Authentifizierung mit der Webanwendung verwendet wird, jedoch würde dies das Problem nur verlagern, da unser gesamter Quellcode öffentlich einsehbar ist. Dies kann, soweit wir wissen, nicht über Umgebungsvariablen in Andriod Studio umgangen werden.

Es gibt natürlich noch viele weitere Endpunkte in unserer Webanwendung, wie zum Beispiel den zur Abfrage von Stunden- oder Mensaplänen. Diese bekommen auch Parameter übergeben, woraufhin diese mit Hilfe verschiedener Methoden die gewünschten Informationen bereitstellen. (vgl. Abschnitt 5.3)

5.2.3 Datenbank

Um die Informationen, welche unser Pepper während seiner Gespräche sammelt auch persistent speichern zu könenn, haben wir uns dazu entschieden, eine MySQL Datenbank zu integrieren. Zunächst haben wir es als fortschrittlicher gehalten, MongoDB zu verwenden, da dies eine JSON-Objekt basierte Speicherung von Daten ermöglicht und durch seine skalierbare und einfache Integration in Expres und JavaScript vielfältige Möglichkeiten bietet.

MongoDB ist jedoch nicht auf dem Hochschulserver installiert und da wir durch verschiedene Kurse schon sehr gute Kenntnisse und Erfahrungen mit MySQL gesammelt haben, war es dann doch nicht so schlimm, auf MongoDB zu verzichten.

Die von der Webanwendung verwendete Datenbank läuft in dem zuvor erwähnten Docker-Kontainer des geteilten Benutzers.

Weitere Informationen und Spezifikationen sind der Tabelle im Abschnitt 5.4.3 zu entnehmen.

5.3 Endpunkte und Responses der Webanwendung

Unsere Webanwendung ist über viele Endpunkte ansprechbar. Nachfolgend sind diese in zwei Gruppen Aufgeteilt. Endpunkte, welche für das Admin Dashboard benötigt werden, sowie als Resultat eine HTML Seite rendern, sind unter dem Abschnitt 5.3.1 zu finden. Alle weiteren Endpunkte, welche zumeist nur Daten entgegen nehmen oder Informationen in Form von JSON zurück geben, sind in Abschnitt 5.3.2 aufgelistet.

5.3.1 Web Interface Endpunkte / Routes

5.3.1.1 Startseite

Beschreibung: Liefert die Startseite der Webanwendung wieder. Auf dieser ist nur der Titel der Anwendung, sowie einen Button, der zur Login Seite führt. (siehe 5.3.1.5)

Methode: GET

Endpunkt: /docker-hbv-kms-http/

Parameter: keine

5.3.1.2 Dashboard

Beschreibung: Liefert einem angemeldetem Admin die Ansicht des Dashboards. (vgl. Abb. 5.3 und 5.4)

Methode: GET

Endpunkt: /docker-hbv-kms-http/dashboard

Parameter: keine

5.3.1.3 Dashboard: Detailansicht

Beschreibung: Dient der visuellen Übermittlung der Detailansicht einer Konversation (vgl. Abb. 5.5). Dies ist nur über den Browser erreichbar, sofern ein gültiges JSON Web Token im Cookie hinterlegt ist.

Methode: GET

Endpunkt: /docker-hbv-kms-http/dashboard/view

Parameter:

Param	Typ	Beschreibung
conversation_id	String	ID der zu abzurufenden Konversation

5.3.1.4 Dashboard: Datenabfrage

Beschreibung: Dieser Endpunkt dient der Abfrage von n Datenreihen, welche auf dem Admin Dashboard dargestellt werden. Dies wird für die Tabelle, sowie für die dortigen Visualisierungen verwendet.

Methode: GET

Endpunkt: /docker-hbv-kms-http/api/v1/getData

Parameter:

Param	Typ	Beschreibung
n	String	Anzahl der abzurufenden Datenreihen

5.3.1.5 Login

Beschreibung: Liefert bei einer GET Anfrage die Login Seite aus. Der POST Endpunkt wird nach Abschicken des Login-Forumulars angesprochen, validiert den Login und leitet den Client bei Erfolg an das Dashboard weiter. Gleichzeitig wird das JSON Web Token im Cookie des Clients hinterlegt.

Methode: GET, POST

Endpunkt: /docker-hbv-kms-http/dashboard/login

Parameter GET: keine

Parameter POST:

Param	Typ	Beschreibung
username_input	String	Benutzername
password_input	String	Passwort

5.3.1.6 Logout

Beschreibung: Löscht das JSON Web Token aus dem Cookie des Benutzers und meldet ihn somit ab. Anschließend wird nach /docker-hbv-kms-http/ weiter geleitet. (vgl. Abschnitt 5.3.1.1)

Methode: GET

Endpunkt: /docker-hbv-kms-http/dashboard/logout

Parameter: keine

5.3.2 API Endpunkte / Routes

Alle Endpunkte, welche nicht über ein GUI angesprochen werden, sind mit "api' in der URI hiterglegt. Anschließend ist die Versionsnummer des Endpunktes anzugeben. Bisher gibt es nur die Version 1.

5.3.2.1 Speicherung von Emotionsdaten

Beschreibung: Dieser Endpunkt dient der Speicherung verschiedener Daten, welche sich im Verlauf einer Konversation mit Pepper ergeben.

Methode: GET

Endpunkt: /docker-hbv-kms-http/api/v1/saveEmotionData

Param	Typ	Beschreibung
identifier	String	Identifikationsnummer der Konversation
distance	String / Float	[optional] Distanz zwischen Pepper und dem Sprecher
age	String / Float	[optional] Alter des Sprechers
gender	String	[optional] Geschlecht des Sprechers
basic_emotion	String	[optional] Generelle Stimmung des Sprechers
pleasure_state	String	[optional] Motivation des Sprechers
excitement_state	String	[optional] Begeisterung des Sprechers
smile_state	String	[optional] Einschätzung der Glücklichkeit des Sprechers
dialog_time	String / Float	[optional] Zeit des Gespräches

5.3.2.2 Speicherung des verwendeten Anwendungsfalles

Beschreibung: Realisierung der Speicherung der in einer Konversation verwendeten Anwendungsfälle.

Methode: GET

Endpunkt: /docker-hbv-kms-http/api/v1/saveUseCaseData

Param	Typ	Beschreibung
identifier	String	Identifikationsnummer der Konversation
use_case	String	Name des Use-Cases

5.3.2.3 Speicherung von nicht verstandenen Phrasen

Beschreibung: Realisierung der Speicherung der von Pepper nicht verstandenen Wörter und Phrasen.

Methode: GET

Endpunkt: /docker-hbv-kms-http/api/v1/saveNotUnderstandPhrases

Param	Typ	Beschreibung
identifier	String	Identifikationsnummer der Konversation
phrase	String	Nicht verstandene Phrase

5.3.2.4 Speicherung von zusätzlichen Daten

Beschreibung: Dieser Endpunkt ermöglicht die Speicherung von weiteren, variablen und nicht vordefinierten Informationen.

Methode: POST

Endpunkt: /docker-hbv-kms-http/api/v1/saveAttributeData

Param	Typ	Beschreibung
identifier	String	Identifikationsnummer der Konversation
data	String / JSON	variable Daten der Konversation

5.3.2.5 Abfrage von Daten aus der DB mittels SQL Query String

Beschreibung: Endpunkt zur Interaktion mit der Datenbank. Ein API-Token wird benötigt. SQL Befehle können übermittelt werden. Aus sicherheitsgründen werden Anfragen, welche folgende Wörter beinhalen, abgelehnt: `CREATE`, `DROP`, `DELETE`, `SHOW`, `USERS`, `INSERT`, `INTO`. Dies gilt für Groß-und Kleinschreibung. Bei erfolgreicher Abfrage wird das Ergebnis des SQL-Statements als JSON zurückgeliefert.

Methode: POST

Endpunkt: /docker-hbv-kms-http/api/v1/sql

Parameter:

Param	Typ	Beschreibung
auth_key	String	API-Token
subject	String	[optional] Art der Abfrage
sql_query	String	Auszuführender SQL Befehl

5.3.2.6 Abfrage des Stundenplans

Beschreibung: Dient der Abfrage von Stundenplänen eines bestimmten Studiengangs. Bei erfolgreicher Abfrage wird ein JSON Objekt, oder einen HTML String zurückgegeben.

Methode: GET

Endpunkt: /docker-hbv-kms-http/api/v1/timetable

Parameter:

Param	Typ	Beschreibung
course	String	Studiengang (Kürzel wie: "WI", "BWL" etc.)
kw	String / Integer	[optional] Kalenderwoche
semester	String / Integer	[optional] Semester
htmlOnly	Boolean	[optional] HTML oder JSON Response (Default: false)

5.3.2.7 Abfrage des Mensaplans als JSON

Beschreibung: Liefert den Mensaplan für die aktuelle Woche als JSON Objekt zurück.

Methode: GET

Endpunkt: /docker-hbv-kms-http/api/v1/mensadata

Parameter: keine

5.3.2.8 Abfrage des Mensaplans als Bild

Beschreibung: Liefert den Mensaplan fur die aktuelle Woche als Bild zurück.

Methode: GET

Endpunkt: /docker-hbv-kms-http/api/v1/mensadata/img

Parameter: keine

5.3.2.9 File Server

Beschreibung: Dieser Endpunkt wird verwendet, Skripte und Stylesheets dynamisch aus dem `static` Verzeichnis abzurufen.

Methode: GET

Endpunkt: /docker-hbv-kms-http/fileserver

Parameter:

Param	Typ	Beschreibung
name	String	Name der angeforderten Datei

5.3.2.10 Abfrage des Kurspreises einer Kryptowährung

Beschreibung: Dieser Endpunkt kann einem den Kurspreis und weitere relevante Informationen zu einem Kryptowährungs-Handelspaar zurückliefern.

Methode: GET

Endpunkt: /docker-hbv-kms-http/api/v1/crypto

Parameter:

Param	Typ	Beschreibung
subject	String	Anforderung (nur "price" ist implementiert)
ksymbolw	String	Symbol (Bsp.: "BTC-USDT")

5.3.2.11 Abfrage der IP-Adresse des Clients

Beschreibung: Liefert die IP Adresse des Clients zurück.

Methode: GET

Endpunkt: /docker-hbv-kms-http/api/v1/ip

Parameter: keine

5.3.3 Responses und Status Codes

Alle An- und Abfragen an unsere Webanwendung liefern verschiedene Antworten / Responses zurück. In der nachfolgenden Tabelle sind die geläufigsten dieser aufgelistet. Da

gewisse Error-Handling-Mechanismen automatisch eine entsprecheende Antwort liefern, kann es zu abweichenden Beschreibungen und weiteren Status Codes kommen.

Tabelle 5.1: Status Codes und deren Bedeutung

Status Code	Bedeutung
200	OK
400	Invalid Parameter
401	Unauthorized
404	Not Found
500	Internal Server Error

5.4 Implementierung

Bei unserer Web App handelt es sich um eine Anwendung, welche mit dem Framework Express für die Laufzeitumgebung Node entwickelt wurde. Express zeichnet sich durch die simple Konfiguration, sowie einfache Implementierung verschiedenster Funktionalitäten aus. Express wird meist für Backendanwendungen verwendet, wir benutzen es jedoch auch, um serverseitiges Rendering zu realisieren. Dies bedeutet, dass Ansichten / Views, welche an den Client ausgeliefert werden, zuvor auf dem Server gerendert worden sind.

5.4.1 Allgemein

Von außen ist unsere Anwendung über die URL https://informatik.hs-bremerhaven.de/docker-hbv-kms-http erreichbar. Sollte sie lokal ausgeführt werden, muss die Domain auf Localhost geändert werden - hier ist dann auch die Angabe des Ports nötig. (mehr dazu in Abschnitt 5.5)

Unsere Anwendung besitzt jedoch nicht all zu viele Views, bis auf das zuvor gezeigte Admin Dashboard (vgl. Abschnitt 5.3.1.2), sowie die Detailansicht der einzelnen Daten der Konversationen (vgl. Abschnitt 5.3.1.3). Darüber hinaus gibt es nur die Startseite und die Login-View. Diese Login-View ist über den Endpunkt `/docker-hbv-kms-http/login` (vgl. Abschnitt 5.3.1.5) erreichbar und beinhaltet nur ein Formular zur Eingabe des Benutzernamens, sowie des dazugehörigen Passwortes.

Abbildung 5.6 zeigt den Inhalt des Root-Verzeichnisses unserer Webanwendung. Im Folgenden wird auf die einzelnen Inhalte eingegangen.

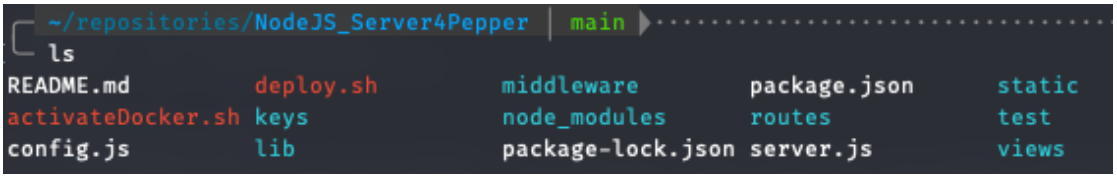


Abbildung 5.6: Hierarchie des Stammverzeichnisses der Webanwendung

/server.js

Der Einstiegspunkt unserer Webanwendung ist das Skript `server.js`. Dieses beinhaltet die Einbindung aller benötigten Dateien und Skripte, sowie die Anwendung der in den Konfigurationsdateien festgelegten Einstellungen (mehr dazu in Abschnitt **??**).

/node_modules, /package-lock.json und /package.json

Da wir Node als Laufzeitumgebung gewählt haben, ist es uns möglich den Node Package Manager (npm) zur Installation verschiedener Module anzuwenden. Alle installierten Module sind in dem Verzeichnis `node_modules` zu finden. Deren Spezifikationen sind in den Konfigurationsdateien `package.json`, sowie `package-lock.json` festgehalten.

package-lock.json ist eine automatisch generierte Datei, welche Node verwendet um Dependencies und Versionen festzuhalten. Aufgrund dieser Konfigurationsdateien ist es nicht nötig, alle im Verzeichnis `node_modules` befindlichen Erweiterungen in das jeweilige GitHub Repository zu laden, denn diese können, nach einer neuen Initialisierung über den Befehl `npm init` anhand der Definitionen in `package-lock.json`, automatisch nachinstalliert werden.

/views

In diesem Verzeichnis befinden sich Templates der Seiten, welche vom Server gerendert und an den Client ausgeliefert werden.

/routes

Der Ordner `routes` beinhaltet die verschiedene Skripte, welche die unterschiedlichen Endpunkte unserer Webanwendung definieren. Innerhalb dieser Skritpe werden auch die Ansichten aus dem `views`-Verzeichnis gerendert und an den Client geschickt. Hier sind alle in Abschnitt 5.3 behandelten Endpunkte definiert. Die jeweiligen Routes implementieren die jeweiligen Funktionalitäten, sind über Requests ansprechbar und interagieren ggf. mit der Datenbank und anderen Services.

/static

Hier finden sich statische Dateien, hierzu gehören Bilder und unabhängige Skripte, sowie Stylesheets. Diese werden über unsere Route `docker-hbv-kms-http/fileserver` ausgeliefert (vgl. Abschnitt 5.2.2).

/middleware

Bei Middlewares handelt es sich um Skripte, welche Abläufe beinhalten, die zwischen anderen Prozessen statt finden. Hierzu gehört bei uns die Authentifizierung von Nutzern, indem das sich dort befindliche Skript `auth.js` eingebunden wird, während ein Nutzer einen beschränkten Endpunkt anspricht. Dies sorgt dafür, dass nur ausgewählte Benutzer bestimmte Endpunkte erfolreich ansprechen können. Alle anderen bekommen die Mitteilung, dass sie nicht die nötigen Berechtigunen besitzen.

/lib

Im Ordner `lib` (aka Libraries) befinden sich alle von uns für die Webanwendung verwendeten Skripte, für welche wir keine genaue Unterkategorie finden konnten. Hierbei handelt es sich um Skripte, welche Anfragen synchron und asynchron durchführen können, sowie Funktionen zur Generierung und Validierung von Passwörtern und deren Hashes.

/keys

Da wir beschränkte Endpunkte haben und unsere Authentifizierung mittels JSON Web Token im Cookie des Clients realisiert ist, muss unsere Anwendung dieses Token auch auf Richtigkeit überprüfen können. Daher haben wir uns dazu entschieden ein Schlüsselpaar zu generieren, welches für die Ver-und Entschlüsselung der JSON Web Tokens verwendet

wird.

Sonstige Dateien

Im Hauptverzeichnis befinden sich weitere Dateien, wie die README.md, welche eine Quick-Start Anleitung bietet, die in dem Repository der Webanwendung dargestellt ist.

Des weiteren befindet sich ein Skript zum Aktivieren des Docker-Kontainers und ein Deploy-Skript, welches für die Aktivierung des Dockers, das Übertragen der Daten in das entsprechende Verzeichnis, sowie für den Start der Webanwendung sorgt.

Im Repository ist auch die Datei `.example.env` zu finden, welche ein Beispiel dafür bietet, wie die `.env` Datei aussehen muss. Diese .env Datei legt Umgebungsvariablen fest, sowie beinhaltet es sensible Daten, wie die Informationen zum Login in die Datenbank, sowie den API-Key zur Nutzung der API über externe Programme / Skripte (vgl. Abschnitt 5.3.2.5). Auch die Festlegung des Ports findet dort statt.

5.4.2 Error-Handling

Auch das Behandeln und Abwenden von Fehlern gehört zu einer guten Webanwendung dazu. Hierbei haben wir alle Endpunkte damit bedacht, entsprechende Fehlermeldungen bzw. Antworten an den Client zu senden. Tabelle 5.1 zeigt unter anderem auch die geläufigsten Fehlermeldungen. Diese werden beispielsweise dann zurückgegeben, wenn ein Nutzer auf einen Bereich zugreift, für welchen er nicht die entsprecheende Berechtigung vorwesein kann, oder fehlerhafte bzw. ungültigte Informationen an einen Endpunkt sendet. Sollten Pfade angesprochen werden, die nicht verfügbar sind, wird der Status Code 404 zurückgegeben. Dies ist in der Datei `server.js` definiert.

Wir loggen unsere Fehler nicht selbst, da dies in diesem Rahmen etwas zu viel wäre, jedoch benutzen wir den Prozessmanager pm2, welcher für uns das Logging übernimmt.

5.4.3 Versionen und Spezifikationen

Unsere Webanwendung basiert auf Node, JavaScript und dem JavaScript Framework Express. Node ist eine Laufzeitumgebung, welche JavaScript außerhalb des Browsers ausführen kann und somit wichtige Prozesse auf dem Server anstatt beim Client ausführt. Node hat einen eigenen Paketmanager, npm (Node Package Manager), mit welchem sich vielfältige Libraries und Frameworks installieren lassen.

Mit npm haben wir zum Beispiel den Prozessmanager pm2 installiert, welcher Prozesse organisiert und bei Abstürzen neu startet. Auch ein Cluster Mode, bei dem Prozesse in mehreren Instanzen laufen können, ist hiermit möglich. Somit ist ein Totalausfall der Anwendung größtenteils vermeidbar.

Es gibt sehr viele weitere Module, welche wir mittels npm installiert haben. Da viele Module von andern Modulen abhängen und npm automatisch die Abhängigen Module mit installiert, wäre es hier nicht sinnvoll, hunderte Module aufzulisten. Deshalb haben wir uns entschieden in folgender Tabelle nur die zwei wichtigsten Module, pm2 und

Express, aufzulisten.

Tabelle 5.2: Hard- und Softwarespezifikationen der Webanwendung

Software / Modul	Version / Spezifikation	Beschreibung
Node	v16.13.1	Laufzeitumgebung
npm	v8.3.2	Paketmanager für Node
MySQL	v15.1 Distrib 10.6.5-MariaDB, for debian-linux-gnu	Datenbank
Express	v4.17.2	JS Framework für Webserver / Webanwendung (Node Modul)
pm2	v5.1.2	Prozessmanager (Node Modul)

Node Module sind mit Hilfe von npm installiert worden und müssen regelmäßig geupdated werden.

Unsere Anwendung ist so konszipiert, dass sie auf den Betriebssystem Linux, macOS und MS Windows läuft, sofern alle benötigten Pakete, Module und Tools installiert sind.

5.5 Installation und Konfiguration

In Abschnitt 5.4.1 haben wir die Datei README.md angesprochen. Diese bietet einen schnellen Überblick über die benötigte Software (Node, npm und MySQL), sowie die Befehle zur Installation alle benöitgten Tools. Im Folgenden werden wir diese Schritte aufzeigen.

Installation

Zu Begin muss sichergestellt werden, dass Node, npm und MySQL installiert sind. Anleitungen hierzu gibt es im Internet. Im besten Fall sollte man die LTS Versionen oder zumindest die aktuellsten Versionen installiert haben. Zudem setzen wir voraus, dass eine voll funktionsfähige Shell mit Bash oder ZSH vorhanden ist. Des Weteren sollte das Projekt schon heruntergeladen sein. Entweder über das Repositoy oder durch die zur Verfügungstellung der Hochschule.

Ist dies geschafft, muss im Root-Verzeichnis der Webanwendung folgender Befehl auf der Kommandozeile ausgeführt werden: `npm install`.

Dies sorgt dafür, dass alle in `package-lock.json` und `package.json` definierten Node Module mit den entsprechenden Versionsnummern installiert werden. Hierdurch wird der Ordner `node_modules` automatisch dem Root Verzeichnis hinzugefügt.

Für diese Anwendung ist es erforderlich, eine MySQL Instanz im Hintergrund laufen zu haben. Des Weiteren muss eine Datenbank für die Anwendung angelegt werden. Diese kann über die Kommandozeile mittels `mysql -e "CREATE DATABASE pepperbackend"` angelegt werden. Der Name “pepperbackend” wurde in den Konfigurationsdateien festgelegt.

Sofern diese Schritte erledigt sind, kann die Anwendung gestartet werden. Möchte man dies auf seinem lokalen rechner über localhost laufen lassen, so lässt sich die Anwendung mittels `npm run dev` auf der Kommandozeile im Root Verzeichnis der Anwendung starten. Dieser Befehl “dev” ist in der Datei `package.json` festgeelegt und startet die Anwendung im Entwicklermodus (für lcoalhost).

Möchte man die Anwendung auf einem Server öffentlich erreichbar starten, so muss man sich mit den gegebenen Portkonfigurationen des Servers auseinander setzen. Sollte alles stimmen, so kann die Anwendung mit `npm run prod` gestartet werden.

Der Unterschied liegt darin, dass man auf seinem lokalen Rechner oft keine verschlüsselten Datenbanken hat und andere Ports benutzt. Daher gilt hier eine strikte Trennung von Entwicklungs- und Produktivumgebung.

Es befindet sich ebenfalls ein Skript `deploy.sh` im Stammverzeichnis der Anwendung, welches die Anwendung auf den Hochschulserver an den richtigen Ort kopiert, abhängigkeiten installiert und den Server startet. Dies muss je nach Umgebung, Benutzer und Server angepasst werden.

Nachfolgend noch einmal die Schritte zur Installation und Start der Anwendung, wobei zu beachten ist, dass die Konfigurationsdateien `.env` und `config.js` angepasst werden müssen, sobald das Projekt heruntergeladen worden ist.

```
~$ git clone https://github.com/ProjectPepperHSB/NodeJS_Server4Pepper.git
~$ cd NodeJS_Server4Pepper
NodeJS_Server4Pepper:~$ npm install
NodeJS_Server4Pepper:~$ mysql -e "CREATE DATABASE pepperbackend"
NodeJS_Server4Pepper:~$ npm run dev
```

Konfiguration

Diese Webanwendung bietet verschiedene Möglichkeiten der Konfiguration. Unsere ist an die Umgebung des Docker-Kontainers des Hochschulservers Hopper angepasst. Es lässt sich jedoch auch für andere Umgebungen anpassen, indem die zuvor erwähngen Konfigurationsdateien `.env` und `config.js` angepasst werden. So lassen sich zum Beispiel die Ports für die Entwicklungs- und Produktivumgebung sowie die Daten zur Authentifizierung mit der Datenbank separat einstellen.

Sonstiges

Der Admin Account, welcher Zugriff auf das Admin Dashboard hat, wir beim initialen Start der Webanwendung automatisch angelegt. Auch alle anderen Tabellen werden erstellt, sofern diese noch nicht in der Datenbank “pepperbackend” hinterlegt sind. Das Passwort des Admin Users wird bei der Erstellung auf dem Terminal ausgegeben. Da wir den Prozessmanager pm2 verwenden und dieser im Produktivmodus den Prozess im Hintergrund startet wird einem das Passwort nicht direkt ausgegeben. Daher muss man sich hierzu mittels `pm2 log` das Logbuch anschauen, in welchem alle Ausgaben der Anwendung aufgeführt werden. Hierzu gehört auch das Passwort.

5.6 Möglichkeiten der Erweiterung

Da wir neben diesem Projekt auch für das KI Lab arbeiten und dort ab und zu gewisse Vorführungen der bisherigen Fähigkeiten von Pepper stattfinden, haben wir uns dazu entschieden, eine andere Variante des Dashboards zu implementieren. Dies ermöglicht die Echtzeitübertragung der von Pepper wahrgenommenen Informationen. Somit können wir mit Hilfe einer weiteren Webanwendung alle Informationen im Browser aufzeigen und zudem verschiedene Diagramme in Echtzeit aktualisieren. Dies bietet die Möglichkeit, den Interagierenden Personen aufzuzeigen, welche Daten in welchem Umfang von Pepper wahrgenommen werden. Da dies nur für die Echtzeitdarstellung gedacht ist, werden diese Daten nicht, wie in unserer zuvor dargelegten Webanwendung, gespeichert. Zu finden ist diese separate Anwendung unter https://github.com/ProjectPepperHSB/WebsocketServer

Realisiert wurde dies ebenfalls mit Hilfe von Express, sowie dem Node Modul Socket.io, welches eine Websocketverbingung zwischen dem Browser (Clientside) und dem Server, bzw. dem Backend erzeugt, über welche dann die von Pepper an die Webanwendung gesendeten Informationen übermittelt werden.

Es wäre denkbar, die Webanwendung weiter auszubauen, verschiedene Benutzerkonten mit unterschiedlichen Berechtigungen anzulegen und diesen nur beschäränkte Einsicht in die Daten zu gewähren. Auch die Steuerung der Pepper App über das Admin Dashboard wäre ein spannendes Projekt.

Ebenfalls ist es mit unserer Implementierung als Grundlage nicht schwer, dieses Sammeln von Daten auf ganz verschiedene Anwendungsfälle zu übertragen, denn die Grundlagen sind auf jeden Fall gegeben, nur die Interaktion mit Pepper müsste je nach Einsatzgebiet angepasst werden.

Kapitel 6

Skripte und Erweiterungen

6.1 Allgemein

Neben der Anwendung für den Roboter Pepper und der Webanwendung haben wir weitere Skripte, hauptsächlich in Python, geschrieben. Diese sind in einem separaten Repository unter https://github.com/ProjectPepperHSB/Backend-Services zu finden. In diesem Repositry befindet sich, wie in jedem anderen unserer Projekte auch, eine README.md, welche einen Einstieg in die Installation und Anwendung der einzelnen Skripte ermöglicht.

Es sind mehrere Verzeichnisse angelegt, welche separate Schwerpunkte beinhalten, auf welche wir in den folgenden Abschnitten genauer eingehen wedern. Auch diese bieten eigene README.md Dateien, welche die Installation, sowie den Zweck aufzeigen.

In jedem dieser kleineren Module befindet sich eine `install.sh`, sowie eine `reguirements.txt` Datei, welche zusammen ausgeführt werden, um die für das Skript benötigten Packages zu installieren, sofern diese noch nicht vorhanden sind.

Es befindet sich auch ein Verzeichnis mit Namen “analysis” in diesem Repository, jedoch gehen wir darauf im Kapiel 7 genauer ein.

6.2 Skripte zur Generierung von Dummy Konversationen

Aufgrund der andauernden Einschränkungen der Regierung, ist es uns nicht möglich, Pepper an der Hochschule im vollem Umfang auszutesten. Somit können wir Pepper und seine Interaktionen nicht an einer großen Menge an Studierenden und Interessierten austesten. Da wir jedoch den Schwerpunkt Big Data mit in unserem Projekt einfließen lassen wollen und unsere Webanwendung, welche wir im Kapitel 5 besprochen haben genau für das Sammeln und zur Verfügung stellen von Daten ausgelegt haben, war es für uns ganz klar, dass wir uns selbst Daten generieren müsssen.

Hierfür wurde ein Skript mit dem Namen `create-dummy-data.py` geschrieben, welches über die Kommandozeile ausgeführt werden kann und mehrere Parameter übergeben

bekommt.

```
~$ python3 create-dummy-data.py -n 1000 --prod
```

Das Flag `n` gibt die Anzahl der zu generierenden Konversationen an. Das zweite Flag `--prod` ist optioanl und sorgt dafür, dass die Daten, welche innerhalb des Skriptes generiert werden, an die URL der laufenden Webanwendung auf dem Hochschulserver gesendet werden. Ohne diesen Flag, werden die Datenreihen an den Localhost geschickt. Sollte dies nicht einwandfrei laufen, so ist die Webanwendung höchstwahrscheinlich nicht aktiv.

Da die `requests` Library in Python nur synchrone Prozesse unterstützt und dies bei 1000 Anfragen etwas Zeit in Anspruch nimmt, haben wir dies mit der Library Joblib parallelisiert, sodass 6 Prozesse gleichzeitg die Generierung der Daten, sowie das Übermitteln an die Webanwendung übernehmen. Aufgurnd der Beschränkungen des Hochschulservers Hopper ist es nicht möglich, noch mehr gleichzeitige Anfragen zu schicken. Dies ist eine Sicherheitsmaßnahme zur Abwendung von DOS Attacken.

6.3 Skripte für die Bereitstellung des Mensaplans

6.4 Skripte für die Bereitstellung des Routenfinders

6.5 Sonstiges

Kapitel 7

Big Data und Ausblick

7.1 Allgemein

Wir haben in unserem Projekt ein großes Augenmerk auf das Sammeln von Daten gelegt. Big Data ist ein Thema, auf welches man in Zeiten wie diesen, in immer mehr Bereichen des Lebens stößt. Sei es die Kontaknachverfolgung, die Digitalisierung sämtlicher gesundheitlicher und behördlicher Dokumente, oder das Sammeln von Kundendaten.

Wir und unser Projekt zählt sich eher zum Letzteren und da wir nun schon auf die Quelle der Daten, Pepper und seine Kommunikation mit den Studenten und Interessierten, sowie auf die Verarbeitung und Speicherung mit Hilfe unserer Webanwendung eingegangen sind, wollen wir nun auch noch einmal aufzeigen, was man mit diesen Informationen anfangen kann. Einen kleinen Ausschnitt haben wir schon in den Abbildungen 5.3, 5.4 und 5.5 gesehen. Doch dort ist nicht mehr, als die bloße Einsicht möglich.

7.2 Datenanalyse mit Python und Jupyter Notebook

Hier kam es uns zu Gute, dass wir Teammitglieder haben, welche sich auch mit Jupyter Notebooks sehr gut auskennen. So ist es passiert, dass wir eine Client Klasse in Python geschrieben haben, welche es uns ermöglicht, durch bloße Eingabe des API Keys, auf Daten in unserer Webanwendung zuzugreifen. Hierfür wird der Endpunkt `/docker-hbv-kms-http/api/v1/sql` angesprochen, an welchen wir unsere SQL Abfragen schicken können und die Ergebnisse als Antwort von unserer Webanwendung zurück bekommen.

Die Client Klasse sorgt automatisch dafür, dass die Anfragen entsrechend formatiert und abgeschickt werden, sowie Fehler ohne Probleme behandelt und mitgeteilt werden. Somit bekommt der Nutzer in seinem Jupyter Notebook mitgeteilt, sofern er einen Fehler in seiner SQL Syntax hat, da wir auch diese Informationen von unserem Server an den Benutzer übermitteln. Nachfolgend ist ein solches Beispiel vorgeführt:

```
[ IN ]  client = Client(API_KEY, sandbox=False)
[ OUT ] {'message': 'Connected!'}
[ IN ]  client.sql_query("select * from users")
```

```
[ OUT ] Exception: 400-Invalid SQL command!
```

Wer sich an Abschnitt 5.3.2.5 erinnert wird bemerken, dass die Tabelle `users` nicht über diesen Endpunkt ansprechbar ist.

Der Parameter `sandbox`, welcher im Konstruktor der Client Klasse verarbeitet wird, gibt an, ob man sich mit der lokalen Instanz der Webanwendung, oder mit der in der Produktivumgebung verbinden möchte.

Hat man nun die Client Klasse instantiiert, ist es mit ihr möglich eine Vielzahl von SQL Abfragen durchzuführen, um an verschiedene von Pepper gespeicherte konversationsdaten zu gelangen.

Mit Hilfe der Bibliotheken Pandas, Matplotlib und weiteren, haben wir innerhalb dieses Notebooks verschiedene Visualisierungen angefertigt. Hierbei ist zu beachten, dass all diese Daten mit Hilfe des Skriptes aus Abschnitt 6.2 generiert worden sind.

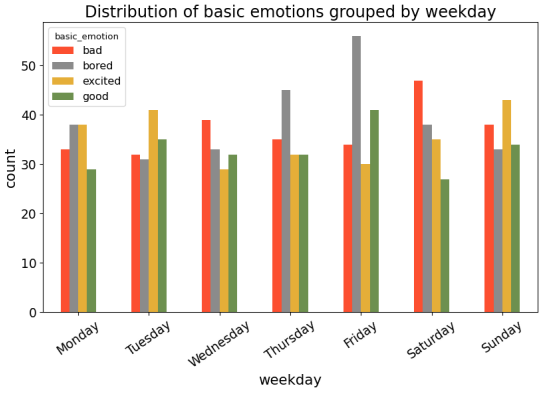


Abbildung 7.1: Diagramm: Emotion je Wochentag

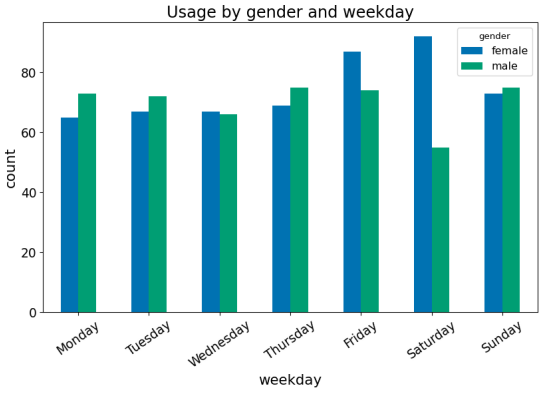


Abbildung 7.2: Diagramm: Geschlecht je Wochentag

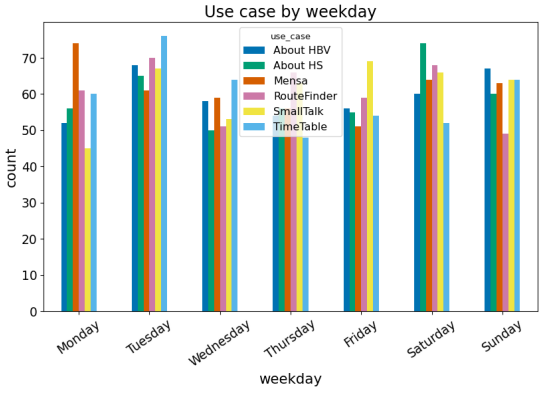


Abbildung 7.3: Diagramm: UseCase je Wochentag

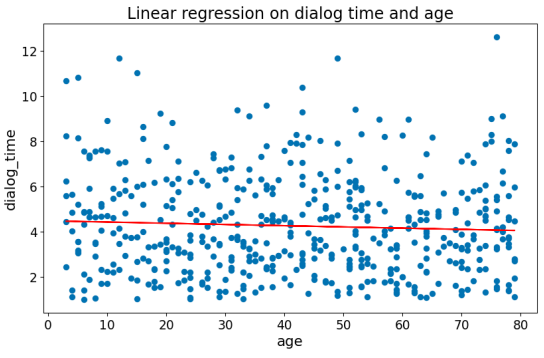


Abbildung 7.4: Diagramm: Lineare Regression zwischen Dialog Zeit und Alter

Die Abbildungen 7.1 ff. geben einen kleinen Vorgeschmack auf die Möglichkeiten der Analyse der von Pepper sammelbaren Daten. in dem Notebook sind weitere Diagramme

zu finden, auch ein simples RNN wurde konstruiert, mit welchem wir das Alter anhand der zur Verfügung stehenden Parameter bestimmen wollten. Jedoch es nicht verwunderlich, dass wir bei einer Genauigkeit von 50% liegen, da diese Daten von uns generiert worden sind.

Diese Schnittstelle zu unserem Webserver ermlglicht es dem Anwender, einen genaueren Einblick in die Daten zubekommen. Zusätzlich zu diesem Notebook, haben wir ein Skript erstellt, welches die Daten aus der Datenbank abruft und die Visualisierungen als PDF Datei speichert. Dies ist in Verbindung mit einem Cronjob, einem wiederkehrenden Prozess sehr nützlich, denn so können wir, aber auch andere, die diese Art der Anwendung nutzen wollen, sich in regelmäßigen Abständen Berichte generieren lassen, welche einen tieferen Einblick in die Daten bieten.

Durch die Anbindung von Pepper an unsere Webanwendung ist es möglich, sämtliche Informationen, während einer Konversation zu speichern und nachzuvollziehen. Hiermit können Unternehmen herusfinden, was ihre Kunden bewegt und in welchen Bereichen man noch an seinem Geschäftsmodell arbeiten muss. Es wäre Denkbar, mehrere Pepper, welche den selben Datensammlungsablauf haben, an verschiedene Unternehmen zu vermieten, womit man als Vermiter ein großes Kontingent an Informationen zu verschiedenen Branchen sammeln und auswerten kann.

Kapitel 8

Abschließende Worte

Wir haben in diesem Projekt sehr viele neue Methoden und Möglichkeiten zur Entwicklung verteilter Anwendungen kennen gelernt. Der Entwurf und die Implementierung einer solch umfangreichen Webseite bzw. Webanwendung hat uns sehr gut gefallen, da wir hier im Vergleich zu vielen anderen Kursen unseres Studiums, Methoden und Werkzeuge aus der Vorlesung direkt anwenden und ausbauen konnten.

Auch die Arbeit im Team war sehr angenehm. Wir haben zum Teil täglich miteinander den Diskurs gesucht, neue Ideen ausgetauscht miteinander gearbeitet. Hierbei konnte jeder etwas von dem Anderen lernen, sodass nach einem Treffen alle etwas schlauer und mit einem guten Gefühl den Jitsi Raum verlassen konnten.

Wir nehmen aus diesem Projekt viele neue Eindrücke und vor Allem neues Wissen mit. Webentwicklung war für uns drei vor diesem Kurs, ein bisher ≫nur≪ interessnates Thema, von dem wir mal gehört haben, mit dem wir aber keine praktischen Erfahrungen teilen konnten. Die intensive Auseinandersetzung mit JavaScript und php hat bisher jedoch bei jedem von uns gefehlt, daher sind wir sehr froh auch dies endlich nachgeholt zu haben.

Literaturverzeichnis

[1] Weihnachtsmann. Softbanks Tutorials? https://dslkfnsfjknksdlfs, letzter Zugriff am irgendwann.