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 **v**

Abstrakt **vii**

1 Dokumentation und Struktur **1**

2 Motivation und Zielsetzung **3**

3 Erste Schritte: Das Grundgerüst **4**
3.1 Allgemein 4
3.2 Wer ist Pepper? 4
3.3 Projektmanagement 5
3.4 Persona 6
3.5 Datensammlung 7

4 Konzept: Pepper App für die Hochschule **9**
4.1 Die Vorüberlegungen 9
4.2 Entwicklungsumgebungen für Pepper 11
4.2.1 Choregraphe 11
4.2.2 Android Studio 12
4.2.3 ROS 12
4.3 Entwicklugnsumgebung 13
4.4 Hard- und Softwarespezifikationen 13

5 Implementierung: Pepper App **14**
5.1 Das Grundgerüst 14
5.2 Programmcode 17
5.2.1 MainActivity 17
5.2.1.1 onCreate 18
5.2.1.2 On Destroy 19
5.2.1.3 Focus Gained 19
5.2.1.4 Focus Lost 19
5.2.2 Weitere Funktionen 19
5.2.2.1 Fragments 19
5.2.2.2 Chatbot und Topics 21
5.2.2.3 Webview 22
5.2.2.4 HumanAwareness 23

5.2.2.5 Personenerkennung . . . 24
5.2.2.6 No Interaction Timer . . . 24

6 Implementierung: Anwendungsfälle 25
6.1 Mensa . . . 25
6.1.1 Script um Mensadaten aufzurufen . . . 25
6.1.2 Weiterleitung zum NodeJS Server . . . 29
6.1.3 Bereitstellen der Mensadaten auf dem Server . . . 29
6.1.4 Anzeigen der Mensadaten in Android Studio . . . 30
6.2 Stundenplan . . . 35
6.2.1 Anzeigen des Stundenplans in Android Studio . . . 35
6.3 Routenfinder . . . 36
6.3.1 Anzeigen des Gebäudeplans und des Raumfinders in Android Studio 36

7 Node - Express Webanwendung 40
7.1 Allgemein . . . 40
7.2 Überblick: Webanwendung . . . 41
7.2.1 Web Interface . . . 42
7.2.2 API . . . 44
7.2.3 Datenbank . . . 45
7.3 Endpunkte und Responses der Webanwendung . . . 46
7.3.1 Web Interface Endpunkte / Routes . . . 46
7.3.1.1 Startseite . . . 46
7.3.1.2 Dashboard . . . 46
7.3.1.3 Dashboard: Detailansicht . . . 46
7.3.1.4 Dashboard: Datenabfrage . . . 47
7.3.1.5 Login . . . 47
7.3.1.6 Logout . . . 47
7.3.2 API Endpunkte / Routes . . . 48
7.3.2.1 Speicherung von Emotionsdaten . . . 48
7.3.2.2 Speicherung des verwendeten Anwendungsfalles . . . 48
7.3.2.3 Speicherung von nicht verstandenen Phrasen . . . 49
7.3.2.4 Speicherung von zusätzlichen Daten . . . 49
7.3.2.5 Abfrage von Daten aus der DB mittels SQL Query String 49
7.3.2.6 Abfrage des Stundenplans . . . 50
7.3.2.7 Abfrage des Mensaplans als JSON . . . 50
7.3.2.8 Abfrage des Mensaplans als Bild . . . 50
7.3.2.9 File Server . . . 51
7.3.2.10 Abfrage des Kurspreises einer Kryptowährung . . . 51
7.3.2.11 Abfrage der IP-Adresse des Clients . . . 51
7.3.3 Responses und Status Codes . . . 51
7.4 Implementierung . . . 53
7.4.1 Allgemein . . . 53
7.4.2 Error-Handling . . . 55
7.4.3 Versionen und Spezifikationen . . . 55
7.5 Installation und Konfiguration . . . 57
7.6 Möglichkeiten der Erweiterung . . . 58

8 Skripte und Erweiterungen **60**
8.1 Allgemein 60
8.2 Skripte zur Generierung von Dummy Konversationen 60
8.3 Skripte für die Bereitstellung des Mensaplans 61
8.4 Skripte für die Bereitstellung des 3D-Navigators 62
8.4.1 Anforderungen 62
8.4.2 Vorhandene Ressourcen 62
8.4.3 Entwicklung und Implementierung des 3D-Navigators 65
8.4.3.1 Planung der Videodaten-Struktur 65
8.4.3.2 Sammeln der Videodaten 66
8.4.3.3 Planung der Datenstruktur für die Routeninformationen 70
8.4.3.4 Sammeln der Wegroutendaten 72
8.5 Sonstiges 75

9 Big Data und Ausblick **76**
9.1 Allgemein 76
9.2 Datenanalyse mit Python und Jupyter Notebook 76
9.2.1 Ideen und mögliche Erweiterungen der Pepper Hochschul-Anwendung 78
9.2.2 Google Dialogflow 78

10 Abschließende Worte **80**

Abbildungsverzeichnis

4.1 Skizze: Konzeptplanng der Anwendungsfälle 10
4.2 Diagramm: Anwendungsfall-Diagramm 10
4.3 Diagramm: Architektur der Anwendung 11

5.1 Grundgerüst Android Studio 14
5.2 MipMapping [6] 16
5.3 Android Activity-Lifecycle [5] 18
5.4 Pepper Main Menu 20

6.1 Mensaplan - cropped 28
6.2 Mensadaten - API 30
6.3 Mensaplan auf Hopper 32
6.4 Mensa Dialog 34
6.5 Gebäudeplan - Pepper 37
6.6 Routenfinder 39
6.7 Routenfinder Dialog 39

7.1 Zusammenhang: Pepper App und Backend 40
7.2 Komponenten der Webanwendung 41
7.3 Webansicht des Admin Dashboards Teil 1 42
7.4 Webansicht des Admin Dashboards Teil 2 43
7.5 Ansicht einer bestimmten Konversation 43
7.6 Hierarchie des Stammverzeichnisses der Webanwendung 53

8.1 Beispiel: Zur Verfügung stehende Campuspläne 62
8.2 Beispiel: AStA Campusplan 63
8.3 Beispiel: 3D-Berlin, 3D Navigator API 64
8.4 Beispiel: Terminal für Kaufhäuser 64
8.5 Vorgehen: Entwicklung des 3D-Navigators 65
8.6 Datenstruktur: Wegrouten Videos 66
8.7 Screenshot: Grafische Oberfläche des Download-Skripts 69
8.8 Screenshot: Ausschnitt der durch das Skript erstellten Videos 69
8.9 Diagramm: ER-Modell der Wegrouten Informationen 71
8.10 Datenstruktur: Abstrahierung der Wegrouten Informationen 71
8.11 Screenshot: Google Chrome Netzwerk Analyse Tool 72
8.12 Screenshot: Grafische Oberfläche des Metadata-Skripts 75

9.1 Diagramm: Emotion je Wochentag 77
9.2 Diagramm: Geschlecht je Wochentag 77

9.3 Diagramm: UseCase je Wochentag 77
9.4 Diagramm: Lineare Regression zwischen Dialog Zeit und Alter 77

Abstrakt

Diese Dokumentation ist im Rahmen des Bachelorprojektes von Benjamin T. Schwertfeger, Jacob Menge und Kristian Kellermann entstanden und dient neben der Dokumentation eines 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 8 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: Das Grundgerüst
- *Kapitel 4*: Die Pepper App für die Hochschule
- *Kapitel 5*: Node - Express Webanwendung
- *Kapitel 6*: Skripte und Erweiterungen
- *Kapitel 7*: Big Data und Ausblick
- *Kapitel 8*: Abschließende Worte

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: Das grundgerüst	Was ist das Fundament unseres Softwareprojektes?
4. Die Pepper App für die Hochschule	Wie kann Pepper der Hochschule helfen?
6. Node - Express Webanwendung	Welche Prozesse laufen im Hintergrund?
6. Skripte und Erweiterungen	Welche Programme haben wir zusötzlich geschrieben?
7. Big Data und Ausblick	Was können wir mit den gesammelten Daten anstellen?
8. 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 vor allem auch die Hochschule einen Mehrwert davon erhält.

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. Diese Anwendungsfälle sind zum einen die App für Pepper, welche der Informationsgewinnung im Bezug auf die Hochschule und das Studentenleben dient, zum anderen wollen wir mit dieser App Daten über die geführten Gespräche sammeln, um das Potential eines solchen Roboters aufzuzeigen.

Kapitel 3

Erste Schritte: Das Grundgerüst

3.1 Allgemein

Wir haben bisher 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 haben 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 Pepper?

Der Roboter Pepper ist ein Humanoid, der wie ein kleiner Junge aussieht und sehr fröhlich und vertrauensvoll wirken soll. Er ist relativ klein, hat einen runden Kopf mit großen Augen und auf seiner Brust befindet sich ein Tablet. Er wurde von dem Unternehmen Softbanks Robotics entwickelt und so konzipiert, dass er als Roboter-Gefährte oder auch persönlicher Roboter dienen kann.

Pepper kann sich über alle möglichen Themen unterhalten und gegebenenfalls auch Informationen oder Animationen auf seinem Tablet anzeigen. Außerdem besitzt er die Fähigkeit, sich in jede Richtung fortzubewegen und verschiedene Gesten durchzuführen. Somit wird er hauptsächlich im Tourismusbereich eingesetzt, wo er sich mit Menschen unterhalten kann.

Das Besondere an ihm ist, dass er für zahlreiche Anwendungen programmiert werden kann, sodass er fast in jedem Bereich einen Nutzen findet. Zum Beispiel könnte er in Verkaufsräumen eingesetzt werden, wo er verschiedene Produkte beschreibt oder als Kunden bei der Beratung unterstützt. Man kann ihn auch in Bereichen wie der Erziehung oder

im Gesundheitswesen einsetzen, wo er die Menschen animiert und kleinere Gegenstände wie Tabletten transportieren kann.

Pepper besitzt neben seinem Tablet auch zwei Kameras, die sich jeweils in seinen Augen befinden. Diese Kameras werden durch Gesichtserkennungssoftware untersützt, sodass er den Menschen vor ihm erkennen und fokussieren kann. Außerdem besitzt er Sensoren, mit denen er Distanzen misst. Zum Beispiel kann er sehr gut einschätzen, wie weit eine Person oder ein Hindernis von ihm entfernt ist. So ist Pepper auch in der Lage zu überprüfen, ob die Person näher kommt und ein Diskurs mit ihm sucht oder sich entfernt und das Gespräch verlässt. An seinem Körper befinden sich viele weitere Sensoren, mit denen beispielsweise auch bemerkt, wenn er berührt wird.

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 fünf Repositories von uns angesammelt, welche folgende Themen beinhalten:

- Pepper Anwendung
- Node Webanwendung für das Dashboard
- Dokumentation
- Skripte und Erweiterungen
- Websocket Server für ein Realtime Dashboard

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

Der Node Webanwendung auf welchem wir in Kapitel 7 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 Daten zum Stunden- und Mensaplan.

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

Für Skripte wie der Generierung von Dummy Daten oder auch zur Speicherung und Organisation von Webinhalten und Tests haben wir das Repository "Backend-Services" angelegt.

Ein Realtime Dashboard, welches Daten nicht persistent, so wie die Node Webanwendung speichert, wird ebenfalls separat versioniert.

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.md`, welche Anforderungen und erste Schritte aufzeigen, um einen schnellen Einstieg zu ermöglichen.

3.4 Persona

Damit wir Pepper mit einer Vielzahl an sinnvollen Antworten auf die verschiedensten 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 Hochschulassiszenz einsetzten wollen, haben wir es mit folgenden Personas zu tun:

- Studenten
- Dozenten
- Angestellte der Hochschule
- Besucher (zukünftige Studenten & Interessenten)

Hauptsächlich soll Pepper für Studenten 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.

Studenten des höheren Semesters sind Themen wie Raumfindung oder Stundenplan eher uninteressant. Dafür könnten Themen wie der Mensaplan oder Informationen über die Hochschule und über Pepper selbst interessant sein. Außerdem wird Pepper in der Lage sein, mit dem Nutzer ein kurzes Gespräch (Small Talk) zu führen. Somit ist Pepper auch für weitere Altersgruppen und Personengruppen interessant. Vor allem Personen, die technikaffin sind oder das Thema Roboter einfach spannend finden, werden Pepper ebenfalls ansprechend finden, weswegen wir für diese Personengruppe passende Gesprächsthemen und lustige Animationen vorbereiten werden.

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 in der Praxis einsetzten und verschiedene Daten sammeln.

3.5 Datensammlung

Sobald der Anwendungsfall Hochschule in Pepper integriert ist und er für den praktischen Test 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 und Zeit des Gespräches
- Alter und Geschlecht
- Emotion / Stimmung
- Mimic

Diese Daten sollen während des Gesprächs zwischen Pepper und dem Anwender gesammelt werden und zu unserer Webanwendung gesendet werden. Dort werden diese anschließend weiter verarbeitet und in einer auf MySQL basierenden Datenbank abgespeichert.

Um die Daten erheben 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 Anwender freude bei der Benutzung des Roboters?
- Hatte der Anwender Schwierigkeiten, sich mit dem Pepper zu unterhalten oder verlief alles reibungslos?
- Welche Personengruppe hat mit Pepper am meisten gesprochen?
- Wie lange dauert ein durchschnittliches Gespräch?

- Gab es Fragen, die nicht beantwortet werden konnten?
- Welcher Anwendungsfall wurde wie oft verwendet?
- Gab es Systemfehler oder Abstürze?
- Was ist die optimale Ansprech Distanz zwischen Anwender und Pepper?
- In welchem Semester und / oder Studiengang ist der Anwender?

Kapitel 4

Konzept: Pepper App für die Hochschule

4.1 Die Vorüberlegungen

Nachdem wir uns darauf geeinigt haben, 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 wenig Einsatzgebiete ergeben. Wir gehen jedoch davon aus, zu einer neuen Normalität zurück zu gelangen, 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:

Abbildung 4.1: Skizze: Konzeptplanng der Anwendungsfälle

Auf Grundlage der diese Grundfunktionalitäten haben wir damit begonnen die Anwendungsfälle weiter zu spezifizieren in dem wir diese in einem Anwendungsfall-Diagramm zusammen getragen haben.

Abbildung 4.2: Diagramm: Anwendungsfall-Diagramm

Wir haben hierbei die nutzbaren Funktionalitäten für die jeweiligen Akteure der Software voneinander getrennt und in Beziehung zueinander gesetzt. Die Anwendung haben wir in zwei verschiedene Grundsysteme voneinander getrennt. Pepper selbst bietet Funktionalitäten für die Akteure der linken Seite des Diagramms, dies können Studenten, Mitarbeiter, Dozenten oder Besucher der Hochschule sein. Rechts sind die serverseitigen Funktionalitäten der Anwendung darstellt, welche Beispielsweise durch uns Entwickler selbst oder einem zugeteiltem Admin des Systems zur verwendbar sind. Es wäre an dieser Stelle auch denkbar, dass hier an den Daten interessierte Personen der Hochschule, wie beispielsweise Werbeagenturen usw. Zugriff auch diese Funktionalitäten erhalten.

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, Besuchern Informationen über die Hochschule, das Leben auf dem Campus, sowie allgemeine Empfehlungen für die Stadt Bremerhaven ausgeben kann.

Die geplante Struktur der Anwendung lässt sich mit der unten dargestellten Abbildung möglichst einfach verdeutlichen.

Abbildung 4.3: Diagramm: Architektur der Anwendung

Die Architektur ist in drei verschiedene Systeme unterteilt, welche in der mit den Pfeilen dargestellten Beziehung zueinander stehen. Hierbei ist der Bereich des Backend-Service ausschließlich dafür da, Daten zu sammeln oder zu generieren, welche dann an den Server gesendet werden. Der Server besteht im groben aus dem Node Server, dem Web Server sowie der Datenbank. Im Gegensatz zu dem Backend-Service steht Pepper mit Server stehen ständig in einem asynchronen Austausch miteinander.

4.2 Entwicklungsumgebungen für Pepper

Um das Verhalten von Pepper zu programmieren, kann entweder die Entwicklungsumgebung Android Studio oder Choregraphe verwendet werden. Im Folgenden werden kurz auf die einzelnen Softwares eingehen und erklären, wieso wir uns für Android Studio entschieden haben. Anschließend gehen wir noch auf eine weitere Entwicklungsumgebung ein, die aber nicht mehr von Softbanks unterstützt wird.

4.2.1 Choregraphe

Mit der Choregraphe Entwicklungsumgebung können komplexe Verhaltensmuster in Pepper integriert werden, ohne eine Zeile Code zu schreiben. Zusätzlich kann das Verhalten aber auch mit der Programmiersprache Python gesteuert werden. Der Choregraphe besitzt mehrer Editoren, mit denen sich zum Beispiel Animationen erstellen lassen können oder Gesprächsfunktionen einbinden lassen. Die Software besitzt

ebenfalls einen Emulator mit einem virtuellen Roboter, wo die Applikation getestet werden kann (mehr online: [1]).

4.2.2 Android Studio

Mit Android Studio können Applikationen für Googles Open-Source Betriebssystem "Andriod" entwickelt werden. Sie ist so konzipiert, dass auch Nicht-Programmierer damit arbeiten können, da sie Editoren anbietet, die das Erstellen von Layouts oder Themes vereinfacht. Die Software basiert auf IntelliJ IDEA 14 und bietet die Programmiersprachen Java und Kotlin. Beim Erstellen einer neuen Applikation kann zwischen diesen beiden Programmiersprachen ausgewählt werden.

Die Software besitzt ihren eigenen SDK Manager, worüber die Pepper SDK heruntergeladen werden kann, um eine Entwicklungsumgebung für Pepper zu schaffen. Mit diesen Entwicklungspaketen kommen viele verschiedene Methoden und Funktionen hinzu, worüber sich das Verhalten von Pepper steuern und programmieren lässt. Zusätzlich bietet das Pepper SDK einen Emulator an, wo die Applikation auf einem virtuellen Pepper getestet werden kann und einen weiteren Editor, mit denen Animationen erstellt werden können. Das Tablet von Pepper kann komplett mit den Funktionen und Editoren von Android Studio editiert werden.

Sowohl Android Studio als auch Choregraphe können bestimmte Komponenten von Pepper nicht ansteuern oder verändern. Hierzu gehört zum Beispiel die Kamera von Pepper, da die einzelnen Funktionen wie die Bilderkennung fest verankert sind. Es kann lediglich nur auf die Daten zugegriffen werden, die Pepper selber mit seinen Kameras erzeugt. (Quelle: [2])

4.2.3 ROS

Es gab vor beginn unseres Projektes die Möglichkeit, Pepper mit ROS (**R**obot **O**perating **S**ystem) zu entwickeln. ROS ist ein Betriebssystem, das speziell für die Steuerung und Programmierung von Robotern entwickelt wurde. Anders als bei Android Studio oder Choregraphe, besitzt man hier jegliche Freiheiten, den Roboter so zu programmieren, wie man möchte. Zum Beispiel hätten wir mit ROS die Möglichkeit gehabt, auf die Kamera oder Sensoren von Pepper zuzugreifen. Dies hätte den Vorteil gehabt, dass wir unsere eigene Bild, oder Spracherkennungsalgorithmen für Pepper entwickeln könnten. Softbanks hat sich jedoch ab einer früheren Version dazu entschieden, die Kompatibilität und Anbindung mit ROS zu stoppen. Hätten wir die Möglichkeit gehabt, ROS zu verwenden, hätten wir uns definitiv dafür entschieden.

4.3 Entwicklugnsumgebung

Wir haben uns dazu entschieden, Android Studio als Entwicklungsumgebung zu verwenden, da wir mit der Pepper SDK eine Vielzahl an Möglichkeiten besitzen, Pepper für jede Situation anzupassen. Außerdem wurde es uns auch von den Masterstudenten empfohlen und wir konnten deren Applikation als Referenz verwenden. Für unsere Applikation verwenden wir Java statt Kotlin, da wir in Java schon einige Konzepte in früheren Projekten umgesetzt haben.

4.4 Hard- und Softwarespezifikationen

Nachfolgend ist eine Tabelle, welche noch einmal systematisch unsere verwendeten Softwares aufzeigt.

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.

Kapitel 5

Implementierung: Pepper App

Nachdem die Einrichtung von Android Studio und die Planung der Hochschul-Anwendungen für Pepper abgeschlossen waren, ging es nun darum, Pepper auf unseren Anwendungsfällen zu programmieren. Der erste Schritt war es, sich mit der Programmstruktur von Pepper vertraut zu machen. Da wir den kompletten Code für die Touristik App der Masterstudenten als Referenz verwenden konnten, haben wir uns schnell ein Bild von der Struktur machen können.

5.1 Das Grundgerüst

(Abb. 5.1) Sobald ein neues Projekt in Android Studio erstellt wird, wird nachfolgende Ordner Struktur generiert.

Abbildung 5.1: Grundgerüst Android Studio

/manifests

In dem Ordner `/manifests` befindet sich die konfigurations Datei AndroidManifest.xml. In dieser Datei können verschiedene allgemeine Einstellungen zu Pepper getroffen werden. Die wichtigsten Einstellungen sind die `permissions`. Dies sind Berechtigung-Parameter, mit denen bestimmte Komponenten von Pepper oder Verbindungen erlaubt werden können. Da wir im Code sehr oft Request an den Node-Server geschickt haben oder Webseiten auf dem Tablet von Pepper angezeigt haben, mussten wir zuerst die Verbindung mit dem Internet in der AndroidManifest erlauben. Auch das Zugreifen auf den Externen Speicher von Pepper kann in dieser Datei erlaubt und konfiguriert werden. Des Weiteren können alle wichtigen Pfade zu weiteren Config Dateien und der MainActivity Klasse gesetzt werden.

/java

In `/java` befinden sich alle unsere Java Klassen und somit der komplette Code der Pepper Anwendung. Sobald ein neues Projekt in Android Studio erstellt wird, wird die MainActivity Klasse automatisch generiert. Diese Klasse ist auch das Herzstück von Pepper, da hier der komplette Code gestartet, gesteuert und beendet wird. Hierzu werden wir im späteren Verlauf mehr erzählen. Die Verzeichnisse, welche mit "andoirdTest" und "test" beschriben sind, werden für das Testen der Applikation verwendet. Diese haben wir in unserem Projekt nicht verwendet, daher gehen wir auf diesen Bereich nicht weiter ein.

/assets

In `/assets` können verschiedene Daten abgelegt werden, auf die im Code zu jeder Zeit zugegriffen werden kann. Zum Beispiel können JSON Dateien abgelegt und im Code gelesen und verändert werden.

ressourcen

In `/res` finden sich die Ressourcen der Anwendung. Diese sind wie folgt gegliedert:

/res/Anim

In dem ersten Unterordner `/anim` befinden sich die Animationen von Pepper. Er besitzt die Möglichkeit, jedes Gelenk zu bewegen, um wie ein Mensch zu wirken. Die Pepper SDK besitzt eine eigene GUI, um Peppers Bewegungen zu konfigurieren und neue Animationen zu erstellen. In dem Editor befindet sich bereits eine große Auswahl an vordefinierten Bewegungen. In unsere Applikation haben wir bis auf die vordefinierte "idle Animation" (Standardbewegung, wenn Pepper untätig ist) keine weiteren Animationen verwendet.

/res/drawable

In dem `/drawable` Ordner befinden sich alle Bilder, die auf dem Tablet angezeigt werden können.

/res/layout

Der `/layout` Ordner beinhaltet alle XML-Dateien, die für die Graphische Darstellung auf dem Tablet verwendet werden. Android Studio bietet hier ebenfalls eine GUI mit verschiedenen Komponenten zur Auswahl und eine Echtzeit Darstellung des Layouts.

/res/mipmap

In `/mipmap` (MIP = multum in parvo = vieles im Kleinen) befinden sich vorberechnete und optimierte Bildfolgen, die die Darstellungsqualität von Texturen verbessern, wenn diese kleiner als in der ursprünglichen Größe dargestellt werden. (vgl. Abb. 5.2) Dadurch, dass Texturen mithilfe der MipMaps verkleinert dargestellt werden, benötigt das Anzeigen von Bildern auf dem Tablet weniger Rechenzeit (mehr: [4]).

Abbildung 5.2: MipMapping [6]

/res/raw

Der `/raw` Ordner beinhaltet alle Topic Dateien der Anwendung und werden im Chatbot zur verbalen Interaktion mit dem Anwender verwendet. In den Topics können Gesprächsthemen vordefiniert und mit verschiedenen Funktionen verbunden werden. Die Syntax eines Topics ist relativ simpel gehalten und selbsterklärend, sodass es uns nicht schwer fiel, viele verschiedene Gesprächsthemen für unsere Anwendung zu entwickeln.

/res/values

In `/values` befinden sich alle XML-Dateien, die Werte für die verschiedenen Ressourcen bereitstellen. Dies sind zum Beispiel Werte wie Farben, Stilen oder Dimensionen.

/res/xml

`/xml` wurde manuell erstellt und beinhaltet hauptsächlich weitere XML-Dateien, die für Konfigurationen verwendet wurden. Hierauf werden wir im späteren Verlauf näher drauf eingehen.

gradle

Im unteren Bereich befinden sich alle wichtigen Gradle Konfigurationsdateien. Gradle ist ein Build-Toolkit, um den Build-Prozess von Android Studio zu automatisieren und zu verwalten. Sobald der Code ausgeführt wird, werden zuerst alle Gradle Einstellungen ermittelt, Bibliotheken und Frameworks installiert und anschließend wird der Code kompiliert. Gradle erstellt anschließend eine APK (Android App Bundles), womit die Androidanwendung erstellt und gestartet werden kann. (Quelle: [3]) In der `/build.gradle` Datei können benutzerdefinierte Build-Konfigurationen definiert werden. Hier kann zum Beispiel die Versionen der Pepper SDK verwaltet werden und außerdem können externe Frameworks, die nicht in der Standardbibliothek von Java enthalten sind, implementiert werden.

5.2 Programmcode

5.2.1 MainActivity

Die `MainActivity` ist die Hauptklasse unserer Anwendung, da hier alle Prozesse gesteuert werden. Die Klasse wird mit der `RobotActivity` Klasse erweitert, die mehrere Methoden bereitstellt, um das Verhalten von Pepper zu programmieren (vgl. Abb. 5.3). Außerdem implementiert sie das `RobotLifecycleCallbacks` Interface, die bestimmte Lebenszyklus-Methoden voraussetzt und bereitstellt. Diese können wie folgt zusammengefasst werden:

Abbildung 5.3: Android Activity-Lifecycle [5]

Die *Android Activity Lifecycle* umfasst Methoden, welche in Android Studio als Standard bereitgestellt werden. Mit der qiSDK und deren `RobotLifecycleCallbacks` kommen die `Robot Lifecycle` Methoden hinzu, die speziell für Pepper entiwckelt worden sind. Alle diese Methoden werden automatisch aufgerufen, sobald Pepper einen bestimmten Punkt oder eine bestimmte Situation erreicht hat. (Dies sind Callback Methoden, welche auf bestimmte Events reagieren.)

Die komplette Applikation läuft auf einem Hauptprozess, der UI Thread genannt wird. Die einzigen Ausnahmen, die nicht auf diesem Thread laufen, sind die Robot Lifecycle Methoden und natürlich auch asynchrone Funktionen.

Im Folgenden werden die wichtigsten `Lifecycle-Methoden` erläutern:

5.2.1.1 onCreate

Die `onCreate` Methode ist der erste Einstiegspunkt, sobald die Pepper Anwendung gestartet wird. Sie wird auch nur einmalig ausgeführt. Damit die Callbacks der RobotLifecycleCallbacks Implementation überhaupt funktioniert, ist der erste Schritt, das registrieren der QISDK. Des Weiteren haben wir in der `onCreate` Methode alle erforderlichen Initialisierungen durchgeführt, wie zum Beispiel das Hinzufügen des Ressourcenordners oder die Spracheinstellung, die Pepper's Chatbot verwenden soll. Mit `setContentView` können wir bestimmen, was auf Pepper's Tablet angezeigt wird.

Hier werden die XML-Dateien aus dem Layout Ordner geladen und in der ContentView initialisiert.

5.2.1.2 On Destroy

Die `onDestroy` Methode wird ausgeführt, sobald die Anwendung beendet wird. Sie wird hauptsächlich dafür verwendet, um Ressourcen wie Threads freizugeben. In unserer Anwendung müssen wir hier zusätzlich die QISDK unregistriert, damit keine weiteren Lifecycle-Aufrufe auftreten.

5.2.1.3 Focus Gained

Die `onRobotFocusGained` Methode ist ein Lifecycle exklusiv für Pepper. Diese wird aufgerufen, sobald Pepper eine Person im Fokus hat. Pepper richtet anschließend sein Kopf zu dieser Person und hält stets Blickkontakt. Selbst wenn die Person sich bewegt, verfolgt Pepper diese Person solange mit seinem Kopf, bis er den Fokus verliert. In dieser Methode werden alle Komponenten initialisiert, die zur Interaktion mit dem Anwender verwendet werden. Hierfür wird die Variable `QiContext` übergeben, die es erlaubt, bestimmte Aktionen durchzuführen, Ressourcen bereitzustellen und verschiedene Dienste abzurufen. Hauptsächlich wird hier der Chatbot mithilfe des `QiContext` bereitgestellt, sodass Pepper direkt mit dem Anwender verbal interagieren kann. Da der Chatbot stetig ausgeführt wird, während der Anwender mit Pepper spricht, wird dieser asynchron ausgeführt, um den Thread dieser Methode nicht zu unterbrechen. Zu dem Thema Chatbot werden wir im späteren Verlauf mehr erzählen.

5.2.1.4 Focus Lost

Sollte Pepper seinen Fokus zu dem Anwender verlieren, wird die `onRobotFocusLost` Methode aufgerufen. Diese Methode wird hauptsächlich dazu verwendet, um die initialisierten Komponenten aus der `onRobotFocusGained` Methode wieder zu beenden (Quelle: [8]).

5.2.2 Weitere Funktionen

5.2.2.1 Fragments

Beim Entwickeln der Applikation haben wir die verschiedenen Pepper Projekte von Softbanks als Referenz verwendet. Diese bieten auf GitHub zahlreiche Beispielprojekte an, wie bestimmte Strukturen oder Funktionen implementiert werden. Eine Funktion, die wir als sehr hilfreich betrachtet haben, war das Aufteilen der Applikation in einzelne Fragmente. Hierbei haben wir uns an dem Template von softbankrobotics orientiert: https://github.com/softbankrobotics-labs/App-Template

Diese Fragmente besitzen ihre ganz eigenen Gesprächsthemen, Darstellungen und Funktionen. Der größte Vorteil an Fragmenten ist, dass die Topics voneinander getrennt werden können. Sollte eine Anwendung zum Beispiel sehr viele Topics besitzen, ist die Gefahr groß, dass Pepper etwas falsch versteht und eine Antwort zu einem anderen Gesprächsthema gibt.

Zu Beginn der Implementierung haben wir unser Projekt in drei verschiedene Fragmente aufgeteilt. Das Hauptmenü ist das erste Fragment, wo allgemeine Dinge gefragt werden können und mit Pepper Small Talk geführt werden kann (vgl. Abb. 5.4). Auf seinem Tablet haben wir ein Hauptmenü erstellt, wo per Knopfdruck das Fragment gewechselt werden kann. Der Anwender hat hier aber auch die öglichkeit, verbal das Fragment zu wechseln. Die anderen beiden Fragmente wurden anschließend getriggert, sobald danach gefragt oder der jeweilige Button gedrückt wurde. Das zweite Fragment beinhaltete alles rund um das Thema Hochschule und Bremerhaven. Hier konnte Pepper über alles ausgefragt werden, was mit diesen beiden Thema zutun hatte. In dem letzten Fragment waren unsere Anwendungsfälle implementiert. Sollte der Anwender sich den Stunden- oder Mensaplan anzeigen lassen wollen, so hätte er zuerst auf dieses Fragment wechseln müssen. Letztendlich haben wir unsere finale Applikation nicht in Fragmente aufgeteilt, sodass zu jeder Zeit nach jedem Thema gefragt werden konnte, ohne das Fragment wechseln zu müssen.

Abbildung 5.4: Pepper Main Menu

Eine Applikation, die aus vielen umfangreiche Themen besteht, zieht den größten Nutzen aus der Fragmentfunktion. Themen, die sich stark voneinander unterscheiden, werden somit voneinander getrennt gehalten und die Applikation spart somit auch an Ressourcen.

5.2.2.2 Chatbot und Topics

Wie bereits erwähnt, wird der Chatbot in der `onRobotFocusGained` Methode aufgerufen. Sobald Pepper eine Person fokussiert, wird der Chatbot als asynchrone Instanz bereitgestellt. Bei der Initialisierung stellen wir zuerst die Sprache, die Pepper beim Sprechen verwenden soll, auf Deutsch und geben anschließend alle Topic Dateien an, die Pepper verwenden soll. In unserer Applikation verwenden wir hauptsächlich die Topics `background` und `concepts`. Das `concepts` Topic dient als Erweiterung für das `background` Topic, da hier mehrere Wörter zu einem Klasse zusammengefasst werden.

```
concept:(hallo)[hey hallo moin na "guten Tag"]
```

Diese können anschließend in dem background Topic wie folgt verwendet werden:

```
u:(~hallo {Pepper})   Hey, schoen dich zu sehen
```

Nun kann Pepper mehrere Variationen von Begrüßungen verstehen.

In den Topics gibt es viele weitere hilfreiche Funktionen, die es zum Beispiel ermöglichen, die aktuelle Uhrzeit oder das Datum als Antwort mitzugeben.

```
u:(~welche-uhrzeit) ^currentTime
u:(~datum) Heute ist der ^currentDate
```

Es ist ebenfalls möglich, dass Peppers eine zufällige Antwort zu einer bestimmten Frage gibt. Diese kann wie folgt definiert werden:

```
    u:([Erzaehl Sag] {mir} {ein} Witz) ^rand[
        "Kommt ein Pferd in die Bar. Fragt der Barkeeper, Hey warum
        so ein langes Gesicht"
        "Wenn Chuck Norris einen Windows Update macht, akzeptiert
        Microsoft seine Bedingungen."
        "129 Prozent der Leute uebertreiben voellig"
    ]
```

Pepper sucht sich hier eine zufällige Antwort aus und erwidert diese auf die Frage “Erzähl mir einen Witz”.

Der größte Vorteil an den Topics ist das Aufrufen einer Methode zu einer bestimmten Frage. Hierbei können einzelne Wörter, die Pepper in der Frage erkannt hat, als Variable mitgeben werden aber auch das übergeben einer Variable von der Methode zur Antwort ist möglich.

```
    u:({was} {gibt{s}} {es} {zu} essen {am} _[~week]) $day=$1
```

```
    ^execute(VariableExecutor, qiVariableMensa, $day) $day gibt
    es entweder $qiVariableMensa.
```

Mit `execute` wird der Methodenaufruf gestartet. Die Methode, die aufgerufen wird, befindet sich in der Klasse `VariableExecutor`. In dieser Klasse befindet sich eine `runWith` Methode, die automatisch ausgeführt wird, wenn die `execute` Funktion in dem Topic erreicht wurde. Die einzelnen Argumente in dieser Funktion sind String Parameter, die wir der Methode `runWith` mitgeben. Das Argument `qiVariableMensa` wird dazu verwendet, um mittels `switch case` Befehlen unterscheiden können, welche Frage getriggert wurde. Alle weiteren Argumente dienen als weitere String Parameter, die wir der Methode übergeben wollen. Sobald die Methode erreicht wurde, gibt es die Möglichkeit, die mitgegeben Parameter umzuschreiben und der Antwort in dem Topic wieder zurückzugeben.
`getCurrentChatBot().setQiVariable(variableName, 'Ich bin jetzt die neue qiVariable');`

5.2.2.3 Webview

Die Webview ist eine nützliche Funktion von Android Studio, die es erlaubt, ganze Webseiten auf dem Tablet von Pepper abzubilden. Man könnte sie ungefähr mit IFrames von HTML vergleichen. In Android Studio funktioniert die Implementierung ungefähr genau sowie die `setContentView` Funktion, da sie ebenfalls auf dem Ui-Thread ausgeführt werden muss. Doch davor müssen noch ein paar wichtige Einstellungen vorgenommen werden, bis diese Funktion implementieren kann. Als Erstes muss die Verbindung mit dem Internet in der `AndroidManifest.xml` erlaubt werden. Wie bereits erwähnt, müssen dafür `permissions` geschrieben werden.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission
.ACCESS_NETWORK_STATE" />
```

Nachdem der Internetzugriff in unserer Applikation erlaubt war, ist der nächste Schritt das Erstellen einer Konfigurationsdatei, um die Netzwerksicherheit zu gewähren. Mit dem Befehl: `<base-config cleartextTrafficPermitted="true" />` stellen wir sicher, dass nur Webseiten mit `https` aufgerufen werden dürfen. Sollte der Webviewer über einen Klartext-Netzwerkverkehr wie z.B HTTP kommunizieren, könnte dies das Risiko erhöhen, dass bestimmte Inhalte manipuliert oder abgehört werden. Standardmäßig ist diese Option auf `false` gesetzt. (Quelle: [7])

Als Nächstes muss das Layout erstellt werden und eine WebView muss als Komponente mit einer eindeutigen Id hinzugefügt werden. Diese Id kann im Code mithilfe der Funktion `findViewById` rausgesucht und mit einem WebView Objekt initialisiert werden. Wenn die Webseite, die wir auf dem Tablet widerspiegeln wollen, Javascript verwendet, müssen wir diese zusätzlich in den Einstellungen der WebView, mittels `webSettings.setJavaScriptEnabled(true)`, erlauben. Mit `web.loadUrl('https:...')`

kann anschließend unsere URL von der Webseite, die wir verwenden wollen, in die WebView geladen werden.

Nun würde sich normalerweise der Standardbrowser des Tablets öffnen, sollte der Anwender in der Webview eine Weiterleitung zu einer anderen Webseite öffnen. Um dies zu verhindern, muss die WebViewClient verändert werden, sodass ein weiterer Webview geöffnet wird, falls dieser Fall eintrifft. Dies kann wie folgt implementiert werden:

```
web.setVebViewClient(new Callback())
try{
    ma.runOnUiThread(() -> {
        ma.setContentView(R.layout.webtest);
        WebView web = (WebView) ma.findViewById(R.id.webView);
        WebSettings webSettings = web.getSettings();
        webSettings.setJavaScriptEnabled(true);
        web.setWebViewClient(new Callback());
        web.loadUrl(url);
    });
}catch (Exception e){
    e.printStackTrace();
}

private class Callback extends WebViewClient {
    @Override
    public boolean shouldOverrideKeyEvent(WebView view, KeyEvent event){
        return false;
    }
}
```

Wir hatten längere Zeit das Problem, dass der Webviewer nicht auf dem Emulator funktioniert hat. Sobald dieser im Code erreicht wurde, kam eine Fehlermeldung, dass die GPU zu leistungsschwach wäre, um diese Aktion durchzuführen. In der älteren Version von Android Studio hat der Webviewer jedoch noch funktioniert. Mit der jetzigen Version ist es zurzeit nicht möglich, die Webview auf dem Emulator auszuführen und muss deshalb auf einen richtigen Pepper getestet werden.

5.2.2.4 HumanAwareness

Damit Pepper weiß, wann er einen Anwender fokussieren soll, gibt es die Möglichkeit, auf die `humanAwareness` Funktion zuzugreifen. Diese ist mit den Kameras und Distanzsensoren von Pepper verbunden. Die Kameras von Pepper befinden sich in seinen beiden Augen und sind mit einer Gesichtserkennung ausgestattet. Die `humanAwareness` Funktion bietet nun eine Vielzahl an Einstellungsmöglichkeiten, wann eine Person von Pepper erkannt wird und wann nicht. Es kann zum Beispiel eingestellt werden, ab welcher Distanz eine Person erst erkannt werden soll. Ist die Distanz sehr niedrig eingestellt, wird Pepper nur nach Personen suchen, die sich in seinem nahen Umfeld

befinden. Auch die Distanz, ab wann er den Fokus verlieren soll, kann eingestellt werden. Sollte Pepper einen Anwender fokussiert haben, ignoriert er alle weiteren Personen in seinem Umfeld. Mit der `humanAwareness` können auch Gruppen von Menschen erkannt werden und Pepper kann zuordnen, ob es sich um eine Familie handelt oder ein Freundeskreis. Sollte zum Beispiel eine Gruppe aus jungen und älteren Menschen vor Pepper stehen, ordnet Pepper sie zu einer Familie zu.

5.2.2.5 Personenerkennung

Peppers Kamera besitzt verschiedene KI basierte Anwendungen, um bestimmte Eigenschaften des Anwenders ermitteln zu können. Dies dient dafür, verschiedene Daten über den Anwender zu sammeln, um daraus zum Beispiel das Nutzerverhalten zu analysieren.

Folgende Daten können über den Anwender gesammelt werden:

- Alter
- Geschlecht
- Emotionen

Diese Eigenschaften sind von Softbanks fest vorgegeben und können nicht erweitert werden. Pepper kann anhand der Emotionen erkennen, ob der Anwender glücklich, neutral, genervt oder traurig ist.

5.2.2.6 No Interaction Timer

Nachdem Pepper bereits mehrere Interaktionsmöglichkeiten und Screens auf seinem Tablet besaß, hatten wir das Problem, dass wir nicht mehr von irgendeinem Screen zum Hauptmenü wechseln konnten. Wenn ein Anwender sich zum Beispiel den Mensaplan auf dem Tablet anzeigen lässt, wäre er nicht mehr in das Hauptmenü zurückgekommen.

Aus diesem Grund haben wir Pepper beigebracht, ins Hauptmenü zu wechseln, sobald der Anwender "Zurück" sagt. Nun konnte zu jeder Zeit während des Gesprächs in das Hauptmenü gewechselt werden. Sollte der Anwender jedoch das Gespräch mit Pepper verlassen, während er sich nicht im Hauptmenü befindet, so würde der nächste Anwender den Screen sehen, der zuletzt verwendet wurde. Um dieses Problem zu lösen, haben wir eine Klasse geschrieben, die den Screen automatisch zurücksetzt. Sollte über eine vordefinierte Zeit keine Interaktion mit Pepper stattgefunden haben, so wird automatisch zum Hauptmenü gewechselt.

Kapitel 6

Implementierung: Anwendungsfälle

Im Folgenden werden wir auf die Implementierung unserer Anwendungsfälle eingehen. Hierbei gehen wir zuerst auf die Planung und anschließend auf die Bereitstellung der Daten in unserem Server und in externe Scripts ein. Anschließend erläutern wir die Implementation und Weiterverarbeitung der einzelnen Anwendungsfälle in Android Studio.

6.1 Mensa

Der Anwendungsfall für die Mensa ist in zwei Teile aufgeteilt. Der erste Teil ist das Anzeigen des Mensaplans auf dem Tablet von Pepper. Sollte der Anwender explizit nach dem Mensaplan fragen, muss Pepper in der Lage sein, eine Übersicht des Mensaplans der aktuellen Woche anzuzeigen. Der zweite Teil ist das verbale Beschreiben der Mensaangebote. Hierbei soll Pepper für jeden Wochentag erzählen können, welche Angebote in der Mensa zurzeit zur Verfügung stehen. Die Mensa der Hochschule Bremerhaven bietet jeden Wochentag immer zwei Angebote an. Das zweite Angebot ist dabei immer vegetarisch. Pepper muss dementsprechend in der Lage sein, beide Angebote beschreiben zu können.

Um die verschiedenen Daten über die Mensa Angebote abrufen zu können, verwenden wir unseren Server, der diese Daten bereitstellt. Der Server bekommt diese Daten aus einem externen Python Script.

6.1.1 Script um Mensadaten aufzurufen

Um die Daten auf dem Server anzeigen zu lassen, wird ein Python Script verwendet, dass sich diese Daten aus der offiziellen Hochschule Bremerhaven Mensa Webseite zieht.

Webseite: https://www.stw-bremen.de/de/cafeteria/bremerhaven

Auf dieser Webseite befinden sich die aktuellen Angebote für die nächsten fünf Tage in einer Tabelle und den aktuellen- sowie nächsten Wochenplan im PDF-Format. Mit dem Script werden die einzelnen Daten aus der Webseite gelesen und als JSON gespeichert. Das Lesen der Daten aus der Webseite wird mit Hilfe des Frameworks `BeautifulSoup` durchgeführt. Dieses Framework erlaubt es, XML- und HTML-Dokumente zu `parsen`.

```
menulist = soup.select('tbody')

menu, offer1, offer2 = {}, [], []
day: [str] = [
    'Montag', 'Dienstag', 'Mittwoch', 'Donnerstag', 'Freitag'
    ]
```

Die Mensa Angebote befinden sich auf der Webseite in einer Tabelle, weshalb wir auf alle `tbody`-Elemente mit der Funktion `soup.select` zugreifen. Diese werden als Array in die Variable `menulist` initialisiert. Anschließend werden die Variablen deklariert, aus denen später der JSON string entstehen soll.

```
for i in range(10):
    tmp = (str(menulist[i]).split('description">',1)[1]).rsplit
    ("</td><td",2)[0]
    tmp = tmp.replace('\n',').replace('\r',').replace('a1',')
    .replace('amp;',')
    tmp = re.sub('<sup>.*?</sup>', |, tmp)
    if(i%2==0): offer1.append(tmp)
    else: offer2.append(tmp)
    menu'"day'], menu['offer1'], menu['offer2'] = day, offer1, offer2

with open(folder_location+"mensadata.json", "w+") as f:
    json.dump(menu, f, ensure_ascii=False)
```

Da sich die Daten in den `tbody`-Elementen schwer lesen lassen, werden diese mit mehreren `split` und `replace` Funktionen in lesbare Mensa Angebote umgeformt und in den einzelnen Variablen eingefügt. Die If-Bedingung wird verwendet, um das erste- und zweite Angebot voneinander zu trennen. Anschließend wird das `menu` Objekt mit den keys `day`, `offer1` und `offer2` und den Values der jeweiligen Arrays initialisiert. Zum Schluss wird das `menu` Objekt in JSON mit der `json.dump` Funktion umgewandelt.

Um den aktuellen Wochenplan im Script herunterzuladen und weiterzuverarbeiten, wird die Programmbibliothek Poppler vorausgesetzt. Poppler wird in Unix-ähnlichen Betriebssystemen dazu verwendet, PDF-Dateien anzuzeigen.

```
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')
menucard = ((str(soup.select("a[href$='/print']")[0])
```

```
.rsplit(' target',1)[0]).split('href=',1)[1])
.replace('"',|)
urllib.request.urlretrieve(menucard, f'{folder_location}Mensaplan.pdf')
```

Mit den ersten beiden Zeilen des Codes wird der komplette HTML-Inhalt der Webseite in die Variable `soup` initialisiert. Da wir die URL des aktuellen Wochenplans benötigen, muss dieser aus dem HTML Code gelesen werden. Der Standort dieser URL ist im HTML Code fest verankert, weswegen wir mit Hilfe von `soup.select`, die URL herausfiltern und mit mehreren `split` und `replace` Funktionen zu einer gültigen URL umformen können. Dieser Zwischenschritt ist nötig, da sich die URL jede Woche ändert. Anschließend wird die URL aufgerufen und als `Mensaplan.pdf` abgespeichert.

```
img = convert_from_path(f'{folder_location}Mensaplan.pdf', 500)[0]

area = (0, 200, 4134, 2800) # R T L B
cropped_img = img.crop(area)

area = (0, 5200, 4134, 5800)
cropped_img2 = img.crop(area)
mergeImgs([cropped_img, cropped_img2])
.save(folder_location+'images/mensaplan.png', 'JPEG')
```

Im nächsten Schritt wird mit der Funktion `convert_from_path` das PDF in ein Bildformat konvertiert und in die Variable `img` installiert. Hierfür wird das Framework `pdf2image` verwendet, das vorab mit `pip` installiert werden muss.

Nun ist das Bild jedoch im Hochformat und kann sehr schlecht auf dem Tablet von Pepper angezeigt werden. (vgl. Abb. 6.1) Somit haben wir das Bild in zwei Teile geschnitten und im Querformat zusammengefügt. Um das Bild zu schneiden, haben wir das Framework `PIL` verwendet. Nun kann genau angegeben werden, wie das Bild geschnitten werden soll. In der Variable `area` haben wir jeweils immer vordefiniert, wie weit das Bild in welcher Richtung ausgeschnitten werden soll. Die Reihenfolge der Richtungen lautet wie folgt: (rechts oben links unten) (0, 200, 4134, 2800)

Zuerst haben wir den oberen Teil des Mensaplans ausgeschnitten, der die einzelnen Angebote enthält und anschließend die Legende, die sich im unteren Bereich der PDF befindet.

Nun müssen beide Bilder zusammengefügt werden. Dies wurde in der Methode `mergeImgs` durchgeführt.

```
def mergeImgs(imgs):
    min_img_width = min(i.width for i in imgs)

    total_height = 0
    for i, img in enumerate(imgs):
```

```
6
7     if(img.width > min_img_width):
8         imgs[i] = img.resize((min_img_width, int(img.height /
9         img.width * min_img_width)), Image.ANTIALIAS)
10    total_height += imgs[i].height
11
12    img_merge = Image.new(imgs[0].mode, (min_img_width, total_height))
13    y = 0
14    for img in imgs:
15        img_merge.paste(img, (0, y))
16
17        y += img.height
18    return img_merge
```

In dieser Funktion wird zuerst die Breite und Höhe des neu zusammengefügten Bildes ermittelt. Sollten die beiden Bilder eine unterschiedliche Breite besitzen, wird das breiteste Bild verkleinert. Dies geschieht in der For-Schleife, wo jedes einzelne Bild überprüft wird. Anschließend wir ein neues `img_merge` Objekt erzeugt, das mit der minimalsten Breite und addierten Höhe der beiden Bilder initialisiert wird. Zum Schluss werden die beiden Bilder in das Objekt eingefügt und heruntergeladen.

Studierenden werk BREMEN

Speiseplan Cafeteria Bremerhaven (KW 48: 29.11. - 03.12.2021)

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
Angebot 1	Nürnberger Bratwürstchen auf Sauerkraut mit Kartoffelpüree[1, g] Stud.: 2,80€ | Bed.: 4,50€	Ofengemüse mit Basilikum-[f] Sojajoghurt-Dip und Rosmarinkartoffeln Stud.: 2,50€ | Bed.: 4,50€	Rindergulasch[1, a, g] " Ungarische Art " mit Paprika und Spiralnudeln[a1] Stud.: 2,80€ | Bed.: 4,50€ AT	Tandoori Seelachs[d], Biryanireis, Joghurt-Dip[g] Stud.: 2,80€ | Bed.: 4,50€	Rote Beete Puffer[a1] mit Kartoffelsalat[j] Stud.: 2,50€ | Bed.: 4,50€
Angebot 2	Roter Linsen Dal[1, i] Stud.: 1,50€ | Bed.: 3,30€	Mango-Kürbissuppe Stud.: 1,50€ | Bed.: 3,30€	Chilikartoffeln mit Blattsalaten in Frenchdressing[j] Stud.: 1,70€ | Bed.: 3,50€	Tagliatelle[a1] mit Lauch-Sojasahne-Sauce[f] und gehackten Walnüssen[h3] Stud.: 1,70€ | Bed.: 3,50€	Gnocchi[a1, c] in Kräuterrahm mit Gemüsestreifen[j] Stud.: 1,70€ | Bed.: 3,50€

AT artgerechte Tierhaltung · Fisch · Geflügel · Lamm · mensaVital · Rindfleisch · Schweinefleisch · Plant based = Vegan · Vegetarisch · Wild

Die im Wochenplan ausgewiesene Zusatzstoff- und Allergenkennzeichnung [1] kann tagesaktuell abweichen. Bitte beachten Sie die Tagesaushänge. Änderungen vorbehalten. Änderungen vorbehalten

Abbildung 6.1: Mensaplan - cropped

6.1.2 Weiterleitung zum NodeJS Server

Das Script (vgl. 6.1.1) befindet sich auf dem Hopper und wird mit `crontab` jeden Montag einmal ausgeführt. Es ist wichtig, dass das Script nur jeden Montag ausgeführt wird, da die Mensa Angebote nur für die nächsten fünf Tage zur Verfügung kann. Sollte das Script zum Beispiel an einem Mittwoch ausgeführt werden, so würden die Tage Montag und Dienstag fehlen.

```
    # m h dom mon dow command
    0 0 * * 1 /usr/bin/python3 ~/getMensaPlan.py
```

Das erzeugte JSON Objekt und das zugeschnittene Bild des Mensaplans werden anschließend in den `static` Ordner vom Node Server eingefügt.

6.1.3 Bereitstellen der Mensadaten auf dem Server

Mit dem folgenden Code werden die Daten aus den jeweiligen "/static" Ordnern an die Webseite gesendet:

```
router.get('/docker-hbv-kms-http/api/v1/mensadata', (req, res) => {
    const filePath = `${__dirname}/../static/data/mensadata.json`;
    const jsonData = JSON.parse(fs.readFileSync(filePath, 'latin1'));
    res.send(jsonData);
});

router.get('/docker-hbv-kms-http/api/v1/mensadata/img', (req, res) => {
    const filePath = `${__dirname}/../static/images/mensaplan.png`;
    const img = fs.readFileSync(filePath);
    res.writeHead(200, {
        'Content-Type': 'image/png'
    });
    res.end(img, 'binary');
});
```

(vgl. Abb. 6.2) In folgendem Pfad befindet sich nun unser JSON Objekt, mit allen Mensaangeboten der aktuellen Woche.
https://informatik.hs-bremerhaven.de/docker-hbv-kms-http/api/v1/mensadata

informatik.hs-bremerhaven.de/docker-hbv-kms-http/api/v1/mensadata

```
{
  "day": [
      "Montag",
      "Dienstag",
      "Mittwoch",
      "Donnerstag",
      "Freitag"
  ],
  "offer1": [
      "HÃ¤hnchenbrust \" Piccata MilanesÃ© \" mit Spaghetti und Sauce Pomodoro",
      "Makkaroni-Tomaten-Brokkoli-Auflaufmit SchmelzÃ¼berbacken",
      "Gebratene Frikadelle mit gestovtem Kohlrabi, Bratenjus und RÃ¶stzwiebelpÃ¼ree",
      "Gebratene Champignons mit RÃ¶sti-Ecken und Aioli",
      "Raviolini Pomodoremit italienischem GemÃ¼se"
  ],
  "offer2": [
      "Asiatische Suppe mit ChinagemÃ¼se und Glasnudeln",
      "Kartoffeltaschenmit FrischkÃ¤se gefÃ¼llt,dazu Tomatenrahm und GemÃ¼sereis",
      "Pikante KÃ¼rbissuppe",
      "Fischfrikadellemit Remouladensauce und RÃ¶stkartoffeln",
      "Hausgemachte KÃ¤se-Lauchsuppe"
  ]
}
```


Abbildung 6.2: Mensadaten - API

Der aktuelle Wochenplan ist in png-Format unter folgendem Link erreichbar:
https://informatik.hs-bremerhaven.de/docker-hbv-kms-http/api/v1/mensadata/img

6.1.4 Anzeigen der Mensadaten in Android Studio

Wie bereits erwähnt, soll Pepper mit Hilfe von Sprache in der Lage sein, den Mensaplan auf sein Tablet anzuzeigen und Informationen über die verschiedenen Angebote jedes Wochentags zu geben. Um sich den Mensaplan anzeigen zu lassen, haben wir zuerst ein neues Gesprächsthema in unserem Haupt-Topic erstellt.

```
u:(["Ich hab{e} hunger" "Zeig {mir} {den} Mensaplan"] )
^execute(VariableExecutor, qiVariableMensa, Plan) Hier hast du eine
Uebersicht ueber den Mensaplan
```

Sobald der Anwender nach dem Mensaplan fragt, wird die `runWith` Methode mit den Parametern `qiVariableMensa` und `Plan` in der `VariableExecutor` Klasse aufgerufen.

```
case('qiVariableMensa'):
String day = params.get(1);
if(day.equals('Plan')){
    ...
}
```

In dieser Methode wird anschließend nach dem switch case `qiVariableMensa` gesucht. Da wir einen zweiten Mensaanwendungsfall besitzen, müssen wir diese beiden Fälle voneinander unterscheiden können. Aus diesem Grund wurde ein weiterer String Parameter mit dem Namen `Plan` mitgegeben, um über einer If-Bedingung diese beiden Fälle zu trennen. Nun gibt es mehrere Möglichkeiten, das Bild auf dem Tablet von Pepper anzeigen zu lassen. Die einfachste Methode wäre über den Webviewer. Hierfür hätten wir lediglich die URL benötigt, um die Webseite mit dem Bild auf dem Tablet anzeigen zu lassen. Die Bildgröße des Mensaplans besaß jedoch eine viel zu große Auflösung (3200 x 4134 px), sodass nur ein Teil des Bildes auf dem Tablet angezeigt werden konnte. Auf dem Node Server oder in Android Studio könnte die Auflösung theoretisch manuell umgestellt werden, was wir jedoch vermeiden wollen.

Wir haben uns anschließend für eine andere Lösung entschieden, wo das Bild in Android Studio als Bitmap vom Node Server heruntergeladen wird. Eine Bitmap ist eine Zahlenmatrix von einem Bild, wo jeder Pixel die jeweiligen Farbinformationen enthält.

```
1  String url_str = "https://informatik.hs-bremerhaven.de/
2  docker-hbv-kms-http/api/v1/mensadata/img";
3  InputStream srt = new URL(url_str).openStream();
4  final Bitmap bitmap = BitmapFactory.decodeStream(srt);
5
6  ma.runOnUiThread(() -> {
7      ma.setContentView(R.layout.mensa_layout);
8      ImageView imageView = (ImageView) ma.findViewById(R.id.iMensa);
9      imageView.setImageBitmap(bitmap);
10 });
```

In diesem Code wird das Bild in einen InputStream initialisiert. Ein InputStream ist eine Folge von Bytes, womit sich Daten lesen lassen können. Der InputStream wird mithilfe der `BitmapFactory.decodeStream` Funktion in eine Bitmap umgewandelt. Anschließend kann eine Imageview mittels der Id mit der Bitmap initialisiert werden.

Nun kann Pepper auf die Frage “Zeig mir den Mensaplan” ein Bild von dem aktuellen Mensaplan auf sein Tablet zeigen.

Abbildung 6.3: Mensaplan auf Hopper

Als Nächstes wird ein weiteres Gesprächsthema erstellt, dass Pepper erlaubt, auch verbal Informationen über die Mensa Angebote zu geben.

```
concept:(week) [Montag Dienstag Mittwoch Donnerstag
Freitag Heute Morgen Uebermorgen]

u:({was} {gibt{s}} {es} {zu} essen {am} _[~week]) $day=$1
^execute(VariableExecutor, qiVariableMensa, $day) $day gibt
es entweder $qiVariableMensa

u:({was} {gibt{s}} {es} {am} _[~week] {zu} essen) $day=$1
^execute(VariableExecutor, qiVariableMensa, $day) $day gibt
es entweder $qiVariableMensa
```

Hier wird die Variable `day` mit dem Wochentag in `_[~week]` initialisiert, dass in der Frage genannt wurde. Wenn der Anwender zum Beispiel fragt: “Was gibt es am Mittwoch zu essen”, wird Mittwoch in die Variable `day` initialisiert. In dem Konzept wurden nicht nur Wochentage, sonder auch Wöter wie Heute, Morgen und Übermorgen hinzugefügt. Somit kann der Anwender auch fragen, was es zum Beispiel morgen zu essen gibt. Anschließend wird wieder die `runWith` Methode aufgerufen mit den Parameter `qiVariableMensa` und dem Wochentag, der genannt wurde. In der VariableExecutor Klasse wird nun die Antwort vorbereitet, die Pepper zu der Frage geben soll. Hierfür wird die `getOffer` Methode in der HelperCollection Klasse aufgerufen.

```
HttpURLConnection con = getConnection("https://informatik.
hs-bremerhaven.de/docker-hbv-kms-http/api/v1mensadata");

BufferedReader in = new BufferedReader(new InputStreamReader(
    con.getInputStream()));
String inputLine;
StringBuffer response = new StringBuffer();
```

```
while ((inputLine = in.readLine()) != null) response.append(inputLine);
in.close();

JSONObject myResponse = new JSONObject(response.toString());
String tmp = myResponse.getString("offer1");
String [] offer1 = tmp.split("\",\"", -1);
tmp = myResponse.getString("offer2");
String [] offer2 = tmp.split("\",\"", -1);
```

Zuerst werden die Daten aus dem Server aufgerufen und mithilfe des `BufferedReader` in das `StringBuffer` Objekt `response` initialisiert. Anschließend wird ein JSON Objekt aus dem `response` erstellt. Für uns war es leichter, die Daten in einem Array zu speichern, weshalb wir zwei Arrays mit den jeweiligen Mensa Angeboten initialisiert haben.

```
Calendar calendar = Calendar.getInstance();
int curday = calendar.get(Calendar.DAY_OF_WEEK) - 2;
```

Nun wird mit der Calendar Bibliothek von Java, der aktuelle Wochentag ermittelt und als Zahl in curday gespeichert. Wäre der aktuelle Wochentag ein Montag, so würde curday mit einer 0 initialisiert. Sollte es ein Mittwoch sein, dann wäre curday eine 2 usw. Somit können wir die Arrays `offer1` und `offer2` mit dem genauen Index des aktuellen Wochentags ermitteln.

```
String [] daylist = {
    "Montag", "Dienstag", "Mittwoch", "Donnerstag", "Freitag"
};
String [] spcdays = {"Heute", "Morgen", "Uebermorgen"};
String answer = "";

for(int i = 0; i < spcdays.length; ++i){
    if(day.equals(spcdays[i])){
        if((curday + i) > 5){
            answer ="Am Wochenende ist die Mensa leider geschlossen";
            break;
        }
        answer = offer1[curday + i].replaceAll("\"", "").replace(
            "[","").replace("]",""
        );
        answer += " oder " + offer2[curday + i].replaceAll("\"", "")
            .replace("[","").replace("]",""
        );
        break;
    }
}
```

In der For-Schleife wird das Array `spcdays` iteriert und mit `if(day.equals(spcdays[i]))` wird anschließend überprüft, ob der Anwender das Angebot für Heute, Morgen oder Übermorgen wissen möchte. Außerdem wird überprüft, ob der Anwender mit der Frage Morgen oder Übermorgen den Freitag überschreitet. Sollte der Wochentag zum Beispiel ein Freitag sein und der Anwender würde fragen, was es morgen in der Mensa zu essen gibt, dann wird Pepper antworten, dass die Mensa am Wochenende geschlossen ist. Sollte der Tag in der Woche sein, so wird die Antwort in der Variable `answer` mit den beiden Angeboten initialisiert.

```
for(int i = 0; i < daylist.length; ++i){
    if(day.equals(daylist[i])){
        answer = offer1[i].replaceAll("\"", "").replace("[","")
            .replace("]","");
        answer += " oder " + offer2[i].replaceAll("\"", "")
            .replace("[","").replace("]","");
        break;
    }
}
```

Diese For-Schleife überprüft, ob der Anwender das Mensaangebot für einen bestimmten Wochentag wissen will. Dementsprechend wird dann die Antwort initialisiert.

```
String offer = HelperCollection.getOffer(day);
ma.getCurrentChatBot().setQiVariable(variableName, offer);
```

Nachdem nun alle Antworten vorbereitet wurden, wird die Variable “qiVariableMensa” mit der Antwort initialisiert und in den Topic zurückgegeben.

Abbildung 6.4: Mensa Dialog

6.2 Stundenplan

Der zweite Anwendungsfall ist das Anzeigen des Stundenplans für jeden Studiengang und für jedes Semester.

6.2.1 Anzeigen des Stundenplans in Android Studio

Nachdem die Stundenplandaten im Server bereitgestellt wurden, war der nächste Schritt das Anzeigen des Stundenplans in Android Studio. Hierfür wurde ein neues Gesprächsthema in dem Haupt-Topic erstellt.

```
u:({so} zeig{e} {er} {mir} {den} Stundenplan {fuer} {den} Studiengang
_~Studiengaenge {[im "fuer das"]} semester _~semester {sofort} {bitte})

$course=$1 $semester=$2 Alles klar ^execute(VariableExecutor,
timetable_course_semester, $course, $semester)
```

Damit der Anwender den Stundenplan einsehen kann, erwartet Pepper zwei Wörter. Der Anwender muss in seiner Frage den Studiengang und das Semester nennen. Erst dann wird der Methodenaufruf gestartet. Somit werden die drei Parameter `timetable_course_semester`, `course` und `semester` mitgegeben.

```
final String timetableurl = "https://informatik.hs-bremerhaven.de/
    docker-hbv-kms-http/api/v1/timetable?course="
    + course_ + "&semester="
    + semester_ + "&htmlOnly=true";

ma.runOnUiThread(() -> {
  ma.setContentView(R.layout.webtest);

  WebView web = (WebView) ma.findViewById(R.id.webView);
  WebSettings webSettings = web.getSettings();
  webSettings.setJavaScriptEnabled(true);
  web.setWebViewClient(new Callback());
  web.loadUrl(url);
});
```

Auf dem Node Server erwartet die API, die wir für den Stundenplan erstellt haben, bestimmte Parameter zu dem Studiengang und dem Semester. Hierfür können wir unsere Parameter, die in der Frage ermittelt wurden, in die Variable `timetableurl` einsetzen und erhalten die URL zu dem richtigen Stundenplan. Dieser kann anschließend mit Hilfe des Webviewers auf dem Tablet von Pepper angezeigt werden.

6.3 Routenfinder

Der dritte und letzte Anwendungsfall ist die Navigationshilfe in der Hochschule Bremerhaven. Sollte der Anwender einen bestimmten Raum oder ein Gebäude suchen, soll Pepper auf seinem Tablet eine genaue Wegbeschreibung geben können.
Videos in /var/www/hbv-kms abgelegt

6.3.1 Anzeigen des Gebäudeplans und des Raumfinders in Android Studio

Da wir die einzelnen Videodateien auf dem Hopper abgelegt haben, brauchen wir für diesen Anwendungsfall die Kommunikation mit dem Node Server nicht. Die JSON Datei, die mit dem Python Script erzeugt wurde, haben wir in den /Assets Ordner in Android Studio abgelegt, um sie später im Code verwenden zu können.

Wie bei allen anderen Anwendungsfällen auch, ist der erste Schritt das Erstellen der einzelnen Gesprächsthemen. Zuerst werden wir den Fall abdecken, wenn sich der Anwender den Gebäudeplan anzeigen lassen will.

```
u:([zeig gib] {mir} {bitte} {den} [Gebaeudeplan Campusplan Gelaende])
^execute(VariableExecutor, qiVariableNav, Plan) Hier hast du eine
    Uebersicht ueber das Hochschulgelaende
```

```
String nav = params.get(1);
if(nav.equals("Plan")) {
    try {
        ma.runOnUiThread(() -> {
            ma.setContentView(R.layout.campus_plan);
        });
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

(vgl. Abb. 6.5) Sobald der Anwender nach dem Gebäudeplan fragt, wird ein vordefiniertes XML-Layout mit einem Bild von dem Gebäudeplan auf dem Tablet angezeigt. Das Bild wurde vorher heruntergeladen und in den Ordner `drawable` gelegt.

Abbildung 6.5: Gebäudeplan - Pepper

Als Nächstes haben wir das Abrufen der Videodateien zu den einzelnen Räumen der Hochschule in Android Studio implementiert.

```
concept:(rooms) [C006 C007 ... Z5150 Z5160 ]

u:(["{Kannst du {mir} [sagen erzaehlen]} Wo [befindet ist] {sich} der
Raum _~rooms {befindet}"])
    $room=$1 Sofort ^execute(VariableExecutor, qiVariableNav, $room)
```

Pepper muss in der Lage sein, alle Räume in der Hochschule zu kennen, um genau herausfiltern zu können, welchen Raum der Anwender sucht. Das Concept `rooms` beinhaltet alle Räume der Hochschule und wird somit in der Frage im Topic verwendet. Sollte der Anwender nach einen Raum fragen, den es nicht gibt, wird Pepper die Frage erst gar nicht verstehen und nicht antworten.

Sollte Pepper den Raum kennen, wird ein Methodenaufruf mit den Parametern `qiVariableNav` und der Raumnummer (`room`) durchgeführt.

```
u:(Zeig mir {bitte} einen _~handicapped Weg [zu in] {dem}
Raum _~rooms)
    $handicap=$1 $room=$2 Alles klar ^execute(VariableExecutor,
    qiVariableNav, $room, $handicap)
```

Wenn der Anwender explizit nach einem barrierefreien Weg zu einem Raum fragt, wird der gleiche Methodenaufruf mit dem zusätzlichen Parameter `handicap` durchgeführt.

```
String room = nav;
String handicapped = "M0000";
if(params.size() == 3) handicapped = "M0001";
```

In der `VariableExecutor` Klasse wird als Erstes überprüft, ob der Anwender nach einem barrierefreien Weg zu dem gewünschten Raum gefragt hat oder nicht. Mit der If-Bedingung wird die Länge der Parameter überprüft, die in der Methode mitgegeben wurde. Wie bereits erwähnt, wird ein zusätzlicher Parameter `handicap` zusammen mit den Parametern `qiVariableNav` und `room` mitgegeben, wenn nach dem barrierefreien Weg gefragt wurde. Somit muss nur die Parameterlänge verglichen werden, um entscheiden zu können, welcher Weg angezeigt werden soll.

```
BufferedReader reader = new BufferedReader(new InputStreamReader(
    ma.getAssets().open("route_metadata.json")));

String response = new String();
for (String line; (line=reader.readLine())!=null; response+=line);

Map jsonJavaRootObject = new Gson().fromJson(response, Map.class);
String path = ((Map)((Map)(jsonJavaRootObject.get(room)))
    .get(handicapped)).get("video_path").toString();
```

Mit der `getAssets().open("route_metadata.json")` Funktion, kann die JSON Datei, die sich in dem `/assets` Ordner befindet, geöffnet und mit dem `BufferedReader` ausgelesen werden. Als Nächstes wird mit Hilfe des `Gson` Frameworks, ein Map Objekt erzeugt, dass alle Informationen der JSON Datei bereitstellt. In der Variable `path`, wird der Wert `video_path` mit Hilfe der genauen Raumnummer herausgefiltert. Hier wird auch die Abfrage mitgegeben, ob der Weg Barrierefrei sein soll oder nicht.

```
String urlVideo ="https://informatik.hs-bremerhaven.de/hbv-kms/"+path;
try{
    ma.runOnUiThread(() -> {
        ma.setContentView(R.layout.webtest);
        WebView web = (WebView) ma.findViewById(R.id.webView);
        WebSettings webSettings = web.getSettings();
        webSettings.setJavaScriptEnabled(true);
        web.setWebViewClient(new Callback());
        web.loadUrl(urlVideo);
    });
}
```

(vgl. Abb. 6.6) Nachdem der genau Pfad zum Video ermittelt wurde, kann nun die URL zu dem Video erstellt und mit dem Webviewer angezeigt werden.

Abbildung 6.6: Routenfinder

(vgl. Abb. 6.7) Zusätzlich zu dem Video soll Pepper dem Anwender mit zusätzlichen Details antworten. Er gibt an, in welchem Gebäude sich der gesuchte Raum befindet, wie weit dieser von seinem Standpunkt entfernt ist und wie viel Zeit benötigt wird, um diesen Raum zu erreichen.

Abbildung 6.7: Routenfinder Dialog

Kapitel 7

Node - Express Webanwendung

7.1 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 7.1: 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 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 wird unser Quellcode hitnerlegt sein.

7.2 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 7.2: 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 7.4).

7.2.1 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 7.4).

Abbildung 7.3: Webansicht des Admin Dashboards Teil 1

(Abb. 7.3) 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 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 7.3.1.3 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 (vgl. Abschnitt 8.2).

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.

Abbildung 7.4: 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. 7.4) 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 7.5: Ansicht einer bestimmten Konversation

(Abb. 7.5) 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 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.

7.2.2 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 (7.3) und Implementierung (7.4) 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[*context*]Data

(vgl Abschnitt 7.3.2.9) 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 7.3.2.5) 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 7.3).

7.2.3 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 7.4.3 zu entnehmen.

7.3 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 7.3.1 zu finden. Alle weiteren Endpunkte, welche zumeist nur Daten entgegen nehmen oder Informationen in Form von JSON zurück geben, sind in Abschnitt 7.3.2 aufgelistet.

7.3.1 Web Interface Endpunkte / Routes

.....

7.3.1.1 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 7.3.1.5).

Methode: GET

Endpunkt: /docker-hbv-kms-http/

Parameter: keine

.....

7.3.1.2 Dashboard

Beschreibung: Liefert einem angemeldetem Admin die Ansicht des Dashboards (vgl. Abb. 7.3 und 7.4).

Methode: GET

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

Parameter: keine

.....

7.3.1.3 Dashboard: Detailansicht

Beschreibung: Dient der visuellen Übermittlung der Detailansicht einer Konversation (vgl. Abb. 7.5). 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

7.3.1.4 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

7.3.1.5 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

7.3.1.6 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 7.3.1.1).

Methode: GET

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

Parameter: keine

. . . . . .

7.3.2 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.

7.3.2.1 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

. . . . . .

7.3.2.2 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

7.3.2.3 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

7.3.2.4 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

7.3.2.5 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

7.3.2.6 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)

7.3.2.7 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

7.3.2.8 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

. . . . . .

7.3.2.9 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

. . . . . .

7.3.2.10 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")

. . . . . .

7.3.2.11 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

. . . . . .

7.3.3 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 7.1: Status Codes und deren Bedeutung

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

7.4 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.

7.4.1 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 in Abschnitt 7.5).

Unsere Anwendung besitzt jedoch nicht all zu viele Views, bis auf das zuvor gezeigte Admin Dashboard (vgl. Abschnitt 7.3.1.2), sowie die Detailansicht der einzelnen Daten der Konversationen (vgl. Abschnitt 7.3.1.3). 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 7.3.1.5) erreichbar und beinhaltet nur ein Formular zur Eingabe des Benutzernamens, sowie des dazugehörigen Passwortes.

Abbildung 7.6 zeigt den Inhalt des Root-Verzeichnisses unserer Webanwendung. Im Folgenden wird auf die einzelnen Inhalte eingegangen.

```
~/repositories/NodeJS_Server4Pepper | main
ls
README.md          deploy.sh          middleware         package.json       static
activateDocker.sh  keys               node_modules       routes             test
config.js          lib                package-lock.json  server.js          views
```

Abbildung 7.6: 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.

/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 nterschiedlichen 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 7.3 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 7.2.2).

/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 7.3.2.5). Auch die Festlegung des Ports findet dort statt.

7.4.2 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 7.1 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.

7.4.3 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 vermeidbar, sofern die Server der Hochschule keine Schwierigkeiten haben.

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 7.2: 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 Betriebssystemen Ubuntu, macOS12+ und MS Windows 7+ läuft, sofern alle benötigten Pakete, Module, Tools und Sprachen installiert sind.

7.5 Installation und Konfiguration

In Abschnitt 7.4.1 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 den 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` festgelegt und startet die Anwendung im Entwicklermodus.

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.

7.6 Möglichkeiten der Erweiterung

Da wir neben diesem Projekt auch für das KI - Lab arbeiten und dort ab und zu gewisse Vorführungen der bisherigen Fährigkeiten 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 auf Github (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 8

Skripte und Erweiterungen

8.1 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 deren 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 9 genauer ein.

8.2 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 7 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.

8.3 Skripte für die Bereitstellung des Mensaplans

8.4 Skripte für die Bereitstellung des 3D-Navigators

Der Campus unserer Hochschule ist ziemlich weitläufig und unübersichtlich. Dazu kommt, dass mit der Zeit einige Gebäude von der Hochschule als Lernräume angemietet wurden, welche nicht direkt als solche zu erkennen sind. Aufgrund dessen erweist es sich, vor allem für Studenten der ersten Semester, sowie Besucher der Hochschule als äußerst schwierig, sich auf dem Campus zu orientieren.

8.4.1 Anforderungen

Die Benutzer des Roboters sollen die Möglichkeit haben, ihn nach einem Raum oder Ort zu fragen woraufhin Pepper eine genaue Wegbeschreibung als Antwort geben soll. Dabei ist es von Vorteil, dem Nutzer eine visuelle Beschreibung zu bieten, da die nicht nur einfacher zu verstehen ist, sondern auch mögliche Sprachbarrieren ausschließt. Dafür soll das Tablett von Pepper als visuelles Kommunikationsmittel für die Wegbeschreibung dienen. Über akustische Mitteilungen soll Pepper zudem eine geschätzte Dauer des Weges, sowie die Entfernung zu dem gesuchten Ort geben. Außerdem ist es wichtig, dass Pepper auch in der Lage ist, barrierefreie Wege für den Benutzer zu beschreiben, da unsere Hochschule auch barrierefreie Wege anbietet und körperlich beeinträchtigte Menschen nicht von einem solchen System nicht ausgeschlossen werden sollen.

8.4.2 Vorhandene Ressourcen

Bei der Suche im Web sind die ersten Ergebnisse für eine Orientierungshilfe oder einem Campusplan, die von der Hochschule Bremerhaven unter der offiziellen Internetseite der Hochschule Bremerhaven bereitgestellten Geländepläne. Diese bestehen aus einem 2-Dimensionalen Campusplan, wie in Abbildung 8.1) zu sehen, sowie einen 3-dimensionalen Campusplan (Abb. **??**).

Abbildung 8.1: Beispiel: Zur Verfügung stehende Campuspläne

Darüber hinsaus wird, auch vom AStA Bremerhaven, ein Campusplan bereitgestellt.

Abbildung 8.2: Beispiel: AStA Campusplan

Alle diese Karten haben gemeinsam, dass sie einem zwar einen Überblick über das Campusgelände bieten, wodurch eine, nach einem bestimmten Raum suchende, Person zwar in der Lage wäre, das passende Haus zu finden, allerdings ist es dort angekommen, immer noch notwendig, den richtigen Raum, mit der entsprechenden Nummer zu finden. Ein weiteres Problem dieser Gebäudepläne ist auch, dass es zwar anhand des Buchstaben vor der zu suchenden Raumnummer möglich ist, das richtige Gebäude zu finden, jedoch nicht, das Haus zu finden, in dem sich beispielsweise die Mensa oder die Bücherei befindet. Da wir vorhaben, Pepper den Weg zu einem bestimmten Ort oder Raum genau beschreiben zu lassen, wäre es mit einem solchen Campusplan ein ziemlich großer Aufwand visuelle Markierungen, in Form von Linien, in die Pläne abbilden zu lassen. Außerdem gibt damit immer noch das Problem, die genaue Route innerhalb eines Gebäudes zu bestimmen, vor allem dann, wenn barrierefreie Wege gewünscht sind. Zudem müssten wir die sprachlich ausgegebenen Informationen zu den Wegrouten selbst erzeugen, was einen sehr großen Aufwand darstellen würde.

Aufgrund dieser genannten Gründe, eignet sich also ein solcher Gebäudeplan relativ schlecht für unser Vorhaben, Pepper sinnvoll und effizient für die 3D-Navigation auf dem Campusgelände zu nutzen. Wir haben uns deswegen für eine, von 3D-Berlin, zur Verfügung gestelltes, schlüsselfreies API-System entschieden. Diese API wurde von Prof. Dr.-Ing. Peter Ritzenhoff 2014 im Auftrag der Hochschule Bremerhaven von der 3D-Derlin VR Solutions GmbH erstellt. Sie besteht aus einem in 3-dimensional, visualisierten Modell der Hochschule Bremerhaven, in welchem die Räume anhand der

Gebäudepläne Maßgetreu animiert wurden.

Abbildung 8.3: Beispiel: 3D-Berlin, 3D Navigator API

Dieses API-System wurde genau für solch einen Anwendungsfall, wie wir ihn erstellen wollen, entwickelt. Für gewöhnlich werden diese wie von 3d-Berlin entwickelten 3D-Navigation in großen Gebäudekomplexen und Kaufhäusern verwendet, um sie dann auf digitalen Endgeräten wie Smartphones oder Digitalterminals für Besucher zur Orientierung zu Verfügung zustellen (Abb. 8.4).

Abbildung 8.4: Beispiel: Terminal für Kaufhäuser

Die Implementierung eines solchen 3D-Navigators stellt also eine innovative Anwendung dar. Wir verfügen mit dieser API über die Möglichkeit, einen Start- und Endpunkt als Anfrage an die API zu schicken und daraufhin per HTTP ein passendes Video, mit dem genauen Ablauf der Route als Antwort zu bekommen. Ein weiterer Vorteil ist, dass die wir die Distanz in Meter, sowie die geschätzte Dauer für eine Route zwischen zwei Räumen abfragen können, wodurch wir diese Informationen von pepper verbal ausgeben zur Verfügung stellen können, während das Tablet ein Video für die Wegbeschreibung abspielt. Anschließend zeigt jedes von dieser API ausgelieferte Video, einen QR-Code mit einem Link zu dem entsprechenden Video. Dieser QR-Code lässt sich von einem

Benutzer mithilfe eines Smartphones einscannen, wodurch sich die Route auch nach Antreten des Weges problemlos nachvollziehen lässt. Außerdem ist die API in der Lage, ein Video mit der Wegbeschreibung für einen barrierefreien Weg zu erstellen, womit die Nützlichkeit des 3D-Navigators noch mehr Anwendungsbereiche abdeckt.

8.4.3 Entwicklung und Implementierung des 3D-Navigators

Wir haben uns dazu entschieden, die Daten des 3D-Navigators auf dem Hochschulserver Hopper bereitzustellen. Dies bietet den Vorteil, dass wir unabhängig von der Verfügbarkeit der 3D-Berlin API sind. Zudem sind wir so in der Lage, die Struktur der benötigten Daten selbst zu definieren, um sie möglichst effizient von Pepper abrufen zu lassen. Zunächst haben wir das Vorgehen für die Erstellung des Navigators geplant, wobei wir es für sinnvoll erachten, das Vorgehen in 5 verschiedene Schritte aufzuteilen. Die Abbildung zeigt die chronologische Vorgehensweise bei der Entwicklung, sowie die primär verwendete Programmiersprache, so wie weitere benötigte Technologien, um das Ziel des jeweiligen Schrittes zu erreichen.

Abbildung 8.5: Vorgehen: Entwicklung des 3D-Navigators

8.4.3.1 Planung der Videodaten-Struktur

Die Strukturierung der Videos soll so erfolgen, dass es bei der fertigen Implementierung für die Kommunikation zwischen Pepper und dem Server möglichst unproblematisch, sowie effizient ist, das passende Video zu der benötigten Anfrage abzurufen. Außerdem ist es aufgrund der speziellen Sonderzeichen innerhalb einiger Raumnamen nicht möglich, die Namen der Räume direkt innerhalb der Deklaration der Videodateien zu verwenden. Die Struktur der Dateinamen besteht deswegen aus drei notwendigen Informationen, diese sind dabei der Übersicht halber mit einem "-" Zeichen voneinander getrennt und bilden immer einen, nur einmalig vorkommenden, Namen zur Wiedererkennung des jeweiligen Videos. In der ersten Sektion befindet sich die ID für den Startpunkt der Route und in der zweiten die des zu erreichenden Endpunktes. Die Definition dieser Identifikationsnummern sind dieselben ID-Namen, wie sie die API von 3D-Berlin verwendet hat. Das hat den Vorteil, dass wir im Nachhinein besser nachvollziehen können, aus welcher Anfrage das entsprechende Video ursprünglich stammt. Zudem

wäre es ineffizient, eine neue Nummerierung der ID's anzulegen. Der dritte Abschnitt des Videonamens beschreibt, ob es sich um ein Video mit einem Barrierefreien Weg oder um einen gewöhnlichen Weg handelt. Dabei steht die Bezeichnung "M0000" für den gewöhnlichen Weg und "M0001" für einen Weg unter Berücksichtigung der Barrierefreiheit. Somit gibt es für jedes Video also zwei verschiedene Varianten. Die untere Abbildung soll eine Abstrahierung dieser Datenstruktur verdeutlichen.

Abbildung 8.6: Datenstruktur: Wegrouten Videos

Bei dem Format der Videos haben wir uns für `mp4` entschieden. Diese Videodateien verfügen über eine gute Browserkompatibilität und bieten trotz einer verhältnismäßig geringen Speichergröße trotzdem eine möglichst hohe Bildqualität. Da wir bei jeder Anfrage einer Wegbeschreibung das entsprechende Video per HTTP vom Server an den Roboter senden, eignet sich dieses Format also gut, da wir so eine entsprechend geringe Verzögerung beim Übertragen und Anzeigen des Videos erreichen. Wir haben uns deswegen auch für eine Bildbreite von 306px und eine Bildhöhe von 544px entschieden, wodurch wir eine durchschnittliche Größe von 1.5MB pro mp4-Datei kommen. Insgesamt soll Pepper in der Lage sein ca. 500 verschiedene Wegbeschreibung zu abzubilden, wodurch sich ein Datensatz von ca. 1.000 mp4-Videos mit insgesamt 845MB Kapazität ergibt.

8.4.3.2 Sammeln der Videodaten

Zunächst haben wir uns einen Überblick darüber verschafft, welche Daten wir mithilfe der 3D-Berlin API erhalten können. Als geeignetes Werkzeug für das automatisierte Arbeiten mit Requests haben wir uns für die Programmiersprache Python entschieden. Python bietet uns die Möglichkeit, schnell und zielorientiert, ein effizientes Skript zu entwickeln, um die benötigten Daten nicht nur auszulesen, sondern auch zu formatieren und sie später direkt in unsere gewünschte Datenstruktur einzubinden. Das Pythonskript `createvideodata.py` ist Teil unserer Backend Anwendungen und dient dazu, die erforderlichen Anfragen für den Download der Videos zu konstruieren. Anschließend werden diese ausgeführt und mit dem richtigen Dateinamen zu versehen. Ein paar relevante Ausschnitte des Skripts möchten wir im Folgenden erläutern. Um genau zu wissen, welche Wegrouten wir erzeugen können, rufen wir zunächst eine Liste der Video-IDs ab.

Wir bekommen diese in Form einer JSON-Datei als Antwort zurück und initialisieren die Variable “videos” mit ihr.

IST DAS WIRKLICH RELEVANT? das ist doch python grundkurs haha

```
import requests
video_data = requests.get(...).json()
```

Wir definieren eine Variable `directory`, welche den Zielpfad der zu speichernden Videos angibt. Die Projekt-ID `project` initialisieren wir, um später mögliche Änderungen für weitere Projekte vornehmen zu können. Die hier gewählte ID “100011” ist die des Projektes der Hochschule Bremerhaven. Zusätzlich initialisieren wir die Domain zu der API von 3d-Berlin in der variable “url”.

```
directory, project = 'videos', '100011'
video_format, start_point = '544x306', 'L00P1133'
url = f'https://cdnguide3dcom.blob.core.windows.net/videos/{project}'
```

Den relevantesten Teil des Skripts stellt die folgende for-Schleife dar. Hier iterieren wir über die in der zuvor eingelesenen JSON-Datei angegebene Liste der Räume, zu welchen es Informationen in Form einer Wegbeschreibung gibt. Wir leiten daraus ab, dass wir zu jedem dieser Einträge auch ein Video von der API anfordern können. Dazu lesen wir bei jedem Eintrag eines Videos die dazugehörige ID des Endpunktes der Route, sowie den mit dieser ID verknüpften Namen des Raumes ein und speichern sie jeweils in einer Variable zwischen. Danach übergeben wir die zu der Wegroute entsprechenden Informationen an unsere zuvor definierte Funktion zum Erstellen des Requests. Wir rufen dabei diese Funktion jeweils einmal für den normalen weg und für den Barrierefreien weg auf.

Das ist wenig aussagen, einfach eine for-schleife, in der eigentlich nur variablen an funktionen übergeben werden, die hier gar nicht gezeigt werden. Den kram würde ich (Benjmain) gar nicht zeigen. Nur schriftlich 2-3-4 Sätze und fertig.

```
        for video in range(len(videos['data']['list'])):
            endpoint = videos['data']['list'][video]['point']
            room = videos['data']['list'][video]['name']
            print(f' DOWNLOAD VIDEO ROUTES TO ROOM: {room}')
            print("accessibily_mode: False")
            get_video(url=url,
                      startpoint=startpoint,
                      endpoint=endpoint,
                      accessibily_mode=False,
                      format=video_format)
            print("accessibily_mode: True")
            get_video(url=url,
                      startpoint=startpoint,
                      endpoint=endpoint,
                      accessibily_mode=True,
                      format=video_format)
```

Ist ein Request erfolgreich ausgeführt wurden, so speichern wir die erhaltene MP4-Datei zwischen und schreiben sie in das vorab gewählte Verzeichnis für die Videos.

Das kann man lassen

```
1    open(f'{directory}/{video_name}', 'wb').write(response_object.content)
```

Als letztes haben wir das Skript mit Hilfe des “PySimpleGUI” Frameworks noch um eine Grafische Oberfläche erweitert. Mit diese sind wir in der Lage, nachhaltig das Verzeichnis zum Speichern der Videos über den windows-explorer auszuwählen. Zudem haben wir einen Ladebalken implementiert, welcher den Fortschritt des Downloads anzeigt. Abbildung 8.7 zeigt die von uns erstellte grafische Anwendung und den Output des Skripts:

Abbildung 8.7: Screenshot: Grafische Oberfläche des Download-Skripts

Ist das Skript durchgelaufen haben wir die Videos erfolgreich in die geplante Struktur und mit der korrekten Deklaration der Dateinamen in ein Verzeichnis gespeichert.

Das ist auch irgendwie unnötig

Abbildung 8.8: Screenshot: Ausschnitt der durch das Skript erstellten Videos

8.4.3.3 Planung der Datenstruktur für die Routeninformationen

Die Strukturierung der Metainformationen soll so gestaltet werden, dass es für Pepper und den Server möglichst einfach ist, das passende Video zu der erfolgten Anfrage abzurufen. Die Metainformationen über die Wegbeschreibungen zu einem Raum setzen sich aus folgenden Attributen zusammen:

Attribute einer Wegroute		
Beschreibung	Kommentar	Datentyp
Art des Weges	Es gibt zu jeder Wegbeschreibung zwei verschiedene Arten, das Ziel zu erreichen. Dabei bilden wir jeweils einen barrierefreien und einen gewöhnlichen bzw. schnell zu erreichenden Weg an. Obwohl es nur zwei verschiedene Zustände für die Art eines Videos gibt, haben uns hierbei bewusst für einen String und nicht für einen booleschen Wert entschieden. Der Grund dafür ist, dass sich bei der Erstellung des Skriptes () gezeigt hat, dass wir so keine Datentypen bei der Strukturierung eines Requests umwandeln müssen. Außerdem haben wir so die Möglichkeit über genügend Agilität zu verfügen, sollten wir dieses Skript zukünftig für ein Projekt werden wollen, welches mehr als zwei Modi verwendet.	String
Videopfad	Dieser Parameter stellt die direkte Verknüpfung zu der entsprechenden Daten mit der dazugehörigen Rauminformation dar. So können wir möglichst präzise das passende Video mit der richtigen Wegbeschreibung ermitteln.	String
Ort	Der Ort ist eine kurze Beschreibung, in welchem Haus und welcher Etage sich der Raum befindet. Eine solche Ortsbeschreibung wäre zum Beispiel “Haus C Erdgeschoss”. Mit dieser Umschreibung kann Pepper dem Benutzer noch eine zusätzliche Information für eine verbale Interaktion bieten.	String
Distanz	Bei der Distanz geben wir die Entfernung, ausgehend vom Startpunkt der Wegbeschreibung bis zum Ziel an. Wir verwenden dabei Meter als Einheit und runden diese für eine bessere Verständlichkeit auf ganze Meter auf.	Integer
Dauer	Die Dauer geben wir in Minuten an. Sie beschreibt die benötigte Zeit vom Startpunkt bis zum Ziel, ausgehend von der durchschnittlichen Schrittgeschwindigkeit.	Integer

Anschließend haben wir begonnen, die zur Verfügung stehenden Eigenschaften der Wegrouten sinnvoll miteinander zu verknüpfen.Dabei sei vorausgesetzt, dass die Informationen immer zu genau einem Video gehören und dabei möglichst einfach zu finden sein sollen. Außerdem gibt es zu jedem Raum, genau zwei verschiedene Wegbeschreibungen, da wir zwischen dem normalen und dem barrierefreien Weg unterscheiden müssen. Dies hat zur Folge, dass zu jeder Wegroute, zwei verschiedene Eigenschaften zur Verfügung

stehen müssen, welche sich jedoch die gleiche Art und Struktur der Datentypen teilen. Diese Beziehungen der Daten zueinander haben wir der Übersicht halber in einem Entity-Relationship-Modell charakterisiert.

Abbildung 8.9: Diagramm: ER-Modell der Wegrouten Informationen

Mit Hilfe dieses Datenmodells, konnten wir ein passendes Datenformat für unseren Anwendungsfall wählen. Wir haben uns dabei für die JavaScript Object Notation entschieden, da dies uns den Vorteil bietet, Datenstrukturen in lesbarer Form abzubilden. Außerdem verfügen wir so über ein Datenformat, welches eine hohe Unabhängigkeit von Programmiersprachen bietet. Dies erweist sich für unsere Architektur als äußerst vorteilhaft, da wir zwischen unseren Anwendungen und Entwicklungsabläufen verschiedene Programmiersprachen verwenden. Die endgültig geplante Struktur der JSON-Datei haben wir unten in der Abbildung 8.10 abstrahiert dargestellt.

Abbildung 8.10: Datenstruktur: Abstrahierung der Wegrouten Informationen

8.4.3.4 Sammeln der Wegroutendaten

In der Dokumentation von 3D-Berlin wird ausschließlich beschrieben, wie die entsprechenden Anfragen für die geforderten Videos zu definieren sind. Aus diesem Grund sind wir so vorgegangen, dass wir mithilfe des Netzwerkanalysetools unseres Browsers die Antworten der gesendeten Anfragen, während der Nutzung des 3D-Navigators beobachtet haben. Dabei konnten wir genau ermitteln, welche Anfragen notwendig sind, um weitere Daten über die Videos zu erhalten und wie diese in ihrer Struktur aufgebaut werden müssen.

Auf dem Bild kann man nichts erkennen. Das ist viel zu klein. Weglassen?

Abbildung 8.11: Screenshot: Google Chrome Netzwerk Analyse Tool

Auch hier haben wir uns für die Programmiersprache Python entschieden. Dafür haben wir innerhalb unseres Backends ein weiteres Skript, `createmetadata.py`,angelegt. Das Skript beschafft sich die Informationen, welche wir unter der API 3D-Berlin finden konnten und wandelt diese in für uns brauchbare Daten um. Außerdem werden diese anschließend in der Datei `routedata.json` gespeichert. Im Folgenden gehen wir auf ein paar relevante Abschnitte des Skripts genauer ein und erläutern diese.

Zu Beginn werden zwei verschiedene Requests definiert. Der Erste erzeugt das Request-Objekt für den normalen Weg und der Zweite generiert das gleiche Request-Objekt als barrierefreie Route.

```
r_M0 = get_request(mode='M0000',
                   project=project,
                   startpoint=startpoint,
                   endpoint=endpoint)
r_M1 = get_request(mode='M0001',
                   project=project,
                   startpoint=startpoint,
                   endpoint=endpoint)
```

Die Funktion “getrequest()” bekommt als Übergabeparameter einen String “mode” übergeben. Dieser Parameter gibt an, um welche Variante der Route es sich handelt.

Außerdem wird eine Projekt-ID “project” übergeben. Über diese ID lässt sich das von uns an die API angeforderte Projekt bestimmt. Wir haben diesen Parameter bewusst variabel implementiert, um das gesamte Skript flexibel für einen möglichen weiteren Anwendungsfall im Zusammenhang mit 3D-Navigations APIs von 3d-Berlin zu gestalten. Es werden zudem ein Start “startpoint” sowie ein Endpunkt “endpoint” an die Funktion übergeben. Diese geben an, von welchem Startpunkt aus, welches Ziel zu erreichen sein soll. Es sei dazu gesagt, dass bei jeder Route immer von demselben Startpunkt ausgehen. Dies hat einerseits den Grund, dass der notwendige Datensatz an MP4-Dateien sonst exponentiell steigen würde und wir somit nicht in der Lage während diese Speicherkapazität auf dem Hopper-Server zu gewährleisten. Ebenso setzen wir für unseren Anwendungsfall des Roboters als Hochschulassistenz voraus, dass dieser sich immer an dem gleichen Ort befindet. Es handelt sich bei dem Startpunkt um das Foyer in Haus K. Wir haben auch diesen Parameter dennoch variabel gelassen, um unsere Anwendung so flexibel wie möglich zu entwickeln.

```
1    def get_request(mode, project, start_point, end_point):
2        return requests.get(
3            url=f'https://services.guid3d.com/route/cors/index.php?\
4            project={project}&start={start_point}&end={end_point}\
5            &mode={mode}&redirect=duration&format=none',
6            allow_redirects=True
7        )
```

```
def get_request(mode: str, project: str, startpoint: str, endpoint: str):
    url = f'https://services.guide3d.com/route/cors/index.php?project=' \
          f'{project}&start={startpoint}&end={endpoint}' \
          f'&mode={mode}&redirect=duration&format=none'
    r = requests.get(url, allow_redirects=True)
    if r.status_code == 200:
        print('Request was Success!')
        print(f'HTTP status code is: {r.status_code}. ')
    else:
        print('something went wrong!')
        print(f'HTTP status code is: {r.status_code}')
    return r
```

Als Nächstes wird über die Liste der Videos iteriert, um den Request für die Anfrage der Routeninformation für genau jedes Video zu erstellen.

```
for d in data['data']['list']:
```

Innerhalb dieser Schleife lesen wir als Erstes den Raumnamen und aus der Beschreibung des Ortes ungewollte Zeichen aus dem String heraus. Dies ist notwendig, da es sonst später in der Android-Applikation erfolgen müsste und eine überflüssige Prozessbelastung darstellen würde. Danach werden die Gleitkommazahlen der Dauer und der Distanz einer Route auf ganze Zahlen, als Integer aufgerundet. Außerdem werden die Sekunden in Minuten umgerechnet.

folgendes wichtig? man müsste sagen wo di vairbalen initialisiert sind

```
    new_json_struct.update(
        {
            f'{room_name}: {
                'M0000':{
                    'path': f'{v_path}M0000.mp4',
                    'location': location,
                    'distance': distance_M0,
                    'time': duration_M0,
                },
                'M0001':{
                    ...
                },
                {...}
            }
        }
    )
```

```
tmp_json_struc = {f'{room_name}': {'M0000': {'video_path': f'{video_path}M0000.mp4',
                                             'location': f'{location}',
                                             'distance': f'{distance_M0}',
                                             'time': f'{duration_M0}'},
                                   'M0001': {'video_path': f'{video_path}M0001.mp4',
                                             'location': f'{location}',
                                             'distance': f'{distance_M1}',
                                             'time': f'{duration_M1}'}}}
new_json_struc.update(tmp_json_struc)
```

Es wird nun ein temporäres JSON-Objekt für die jeweilige Wegroute erzeugt. In dieser wenden wir die zuvor geplante Struktur der Informationen an und weisen die Values den entsprechenden Key zu. Danach wird jedes Objekt an die endgültige JSON-Datei angeknüpft.

```
tmp_json_struc = {f'{room_name}': {'M0000': {'video_path': f'{video_path}M0000.mp4',
                                             'location': f'{location}',
```

Wir haben auch für diese Anwendung nachträglich eine grafische Oberfläche implementiert, um das Ausführen des Programms übersichtlicher und nachhaltiger zu gestalten. Dabei haben wir die Oberfläche so gestaltet, dass wir zwei verschiedene Konsolen für die Outputs anzeigen lassen. Die Linke Ausgabe ist die des Programmablaufs. Auf der rechten Ausgabe sind die hinzugefügten JSON-Objekte aufgelistet. In der folgenden Abbildung ist ein Durchlauf des Programms zu sehen. Es gibt die Möglichkeit ein Zielverzeichnis und den namen für die zu erstellende Datei zu wählen. Werden diese beiden Eigenschaften nicht zuvor gesetzt und mit einem Klick auf die “Submit” Schaltfläche bestätigt, so bekommt die Datei den namen “routedata” und wird in das Verzeichnis `video` geschrieben. Die Abbildung unten zeigt das laufendene Programm:

Abbildung 8.12: Screenshot: Grafische Oberfläche des Metadata-Skripts

8.5 Sonstiges

Kapitel 9

Big Data und Ausblick

9.1 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 7.3, 7.4 und 7.5 gesehen. Doch dort ist nicht mehr, als die bloße Einsicht möglich.

9.2 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 7.3.2.5 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 8.2 generiert worden sind.

Abbildung 9.1: Diagramm: Emotion je Wochentag

Abbildung 9.2: Diagramm: Geschlecht je Wochentag

Abbildung 9.3: Diagramm: UseCase je Wochentag

Abbildung 9.4: Diagramm: Lineare Regression zwischen Dialog Zeit und Alter

Die Abbildungen 9.1 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.

9.2.1 Ideen und mögliche Erweiterungen der Pepper Hochschul-Anwendung

9.2.2 Google Dialogflow

Dialog Flow, ist eine von dem Konzern Google betriebene Plattform, welche das verarbeiten und verstehen von menschlicher Sprache ermöglicht. Sie bietet den Vorteil, dass die Verarbeitung der Sprache nicht mehr auf dem eigenen System stattfinden, sondern an einen von Google bereitgestellten Server gesendet wird. Dieser verarbeitet die Daten und sendet sie anschließend wieder zurück. Dialog Flow hat dabei den Vorteil, dass sich die Interaktion der Dialoge dynamisch auf der von Google erstellten Browser-Plattform bearbeiten lassen. Eine Implementierung in das Pepper System, könnte so das produktive Arbeiten stark verbessern. Es ist außerdem möglich Lernraten zu definieren, durch welche Pepper, ohne das erneute Aufspielen einer Applikation dazu lernen könnte. Pepper wäre mit einer solchen Architektur in der Lage, selbst und dynamisch dazuzulernen und seine Interaktion und das Verstehen von Wörtern und Sprache ständig weiter zu optimieren. Wir haben für diese Funktionalität ein grafisches Konzept erstellt, welches den Ablauf dieser theoretischen Funktionalität verdeutlichen soll: **Hier Bild noch machen tun**

Obwohl es eine von SoftBanks-Robotik offiziell bereitgestellte Schnittstelle für eine Anbindung an die Google-Dialogflow Plattform gibt, haben wir uns nicht nur aus zeitlichen Gründen gegen die Entwicklung einer solchen Funktionalität entschieden. Dabei gab es für uns zwei entscheidende Faktoren. Erstens empfinden wir es als äußerst unpassend, unser bereits entwickeltes System, welches dem Administrator den Vorteil bietet, die gesammelten Daten selbst zu besitzen, mit einer Anwendung zu verknüpfen, durch welche diese an einen Großkonzern weitergegeben werden. Das würde für uns bedeuten, dass wir einen unserer bedeutendsten Schwerpunkte dieser Arbeit verlieren, und zwar das ermöglichen gesammelte Daten selber zu verwalten und beliebig in eigene Dateiformate zu konvertieren und weiterzuverarbeiten. Außerdem ist es so, dass Google die Dialogflow Plattform nur für zahlende Kunden in Form eines Abo-Modells anbietet, wobei hier Mikrobeträge pro Request an die Plattform bezahlt werden müssen. Es wäre also alleine schon ein Kostenfaktor, eine Anwendung, während des Entwicklungsprozesses mit diesem System zu testen. Trotzdem möchten wir der Vollständigkeitshalber auf die Möglichkeit der Anbindung an Google Dialog hinweisen, auch wenn sie sich nicht mit den Ethischen-Vorstellungen unserer Softwareanwendung vereinbaren lassen.

Kapitel 10

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] aldebaran. Choregraphe overview. http://doc.aldebaran.com/1-14/software/choregraphe/choregraphe_overview.html, letzter Zugriff am 01.02.2022.

[2] Stephan Augsten. Was ist Android Studio? https://www.dev-insider.de/was-ist-android-studio-a-605428/, letzter Zugriff am 01.02.2022.

[3] NA. Configure your build. https://developer.android.com/studio/build, letzter Zugriff am 10.02.2022.

[4] NA. MipMaps. https://wiki.delphigl.com/index.php/MipMaps#:~:text=Mip%20Mapping%20(MIP%20%3D%20multum%20in,Aufl%C3%B6sung%20des%20Bildes%20in%20Pixeln), letzter Zugriff am 10.02.2022.

[5] NA. Android activity lifecycle. https://developer.softbankrobotics.com/sites/default/files/repository/64_rst_pepper/_build/html/_images/robot_life_cycle.png, letzter Zugriff am 15.02.2022.

[6] NA. Mipmap. https://en.wikipedia.org/wiki/Mipmap#/media/File:Mipmap_Aliasing_Comparison.png, letzter Zugriff am 15.02.2022.

[7] NA. ¡application¿. https://developer.android.com/guide/topics/manifest/application-element#usesCleartextTraffic, letzter Zugriff am 15.02.2022.

[8] SoftBank Robotics. Mastering Focus and Robot lifecycle. https://developer.softbankrobotics.com/pepper-qisdk/principles/mastering-focus-robot-lifecycle, letzter Zugriff am 11.02.2022.