Weiterentwicklung eines Chatbots

Projektarbeit

„Habláme - das sprechende Faultier“

Hochschule für angewandte Wissenschaften
Würzburg-Schweinfurt

David Artmann
Dominik Hirsch
Kristoffer Schneider

28. August 2015

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Allgemeine Änderungen

1.1 Organisatorisches

1.1.1 Strutktur der Repositories

Das ursprüngliche Projekt bestand aus zwei Teilen: der Android App, als lokaler Client, und einem Backend, dessen Service den Bot beinhaltet. Diese Teile waren bisher zusammen in einem einzigen Repository. Das Zusammenlegen hatte eine gewisse Unübersichtlichkeit zur Folge. Daher haben wir für jedes Teilprojekt ein eigenes Repository angelegt. Die Aufteilung zwischen App und Backend wurde beibehalten, lediglich kam noch ein zusätzlicher Teil hinzu, die Service-API-Library. Damit ist die aktuelle Aufteilung auf Github wie folgt:

- Altes Repository unserer Vorgänger - https://github.com/TeamChatbot/chatbot
- Aktuelle Android App - https://github.com/TeamChatbot/Hablame-Android-App
- Aktuelle Service-API-Library - https://github.com/TeamChatbot/hablame-service-api
- Aktuelles Service-Botbackend - https://github.com/TeamChatbot/Hablame-BotBackend

Durch diese Aufteilung kann besser und unabhängiger an den einzelnen Teilprojekten gearbeitet werden.

1.1.2 Aktuelle Aufgabenverteilung auf die Teilprojekte

Durch das Hinzufügen der Service-API-Library ergibt sich nachfolgend dargestellter Gesamtaufbau.

Jedes Teilprojekt behandelt ein Teilproblem des Gesamtsystems.
Android App:

- Umwandlung gesprochener Sprache in Text (SSpeech-To-Text")
- Umwandlung von Text in gesprochene Sprache ("Text-To-Speech")
- Weiterleiten der Informationen mittels der API-Service-Library an das Backend

API-Service-Library:

- Erstellen der Anfragen an das Backend
- Weitergabe der Backend-Antworten an das aufrufende System

Backend:

- Hosting und Konfiguration des Chatbots
- Auswerten der Serviceanfragen

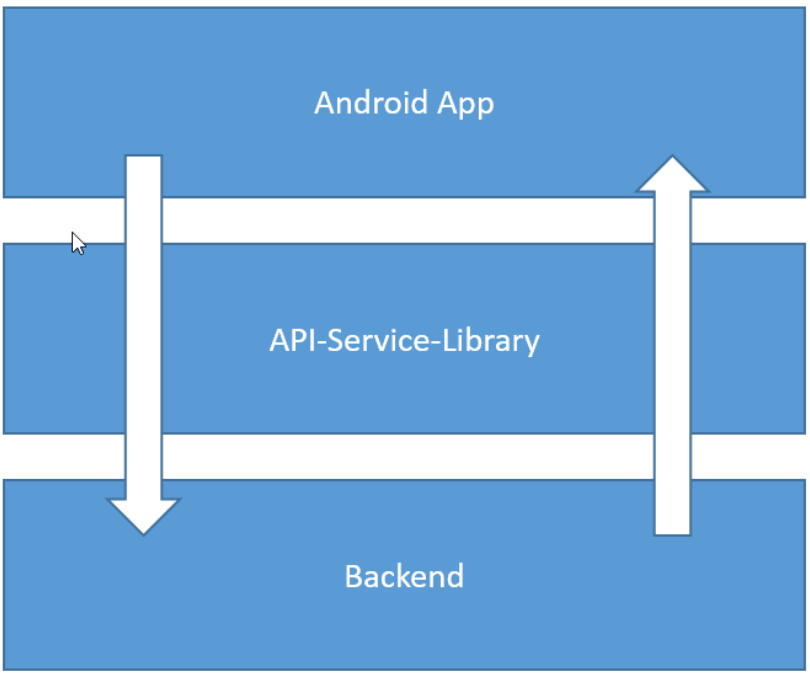

Abbildung 1: Gesamtaufbau Hablame

1.2 Dokumentation im Allgemeinen

Für die Dokumentation gibt es nun mehrere Anlaufstellen. In Redmine gibt es ein integriertes Wiki welches wir bereits mit einigen Informationen zu allgemeine Dingen wie Logindaten und den einzelnen Repositories, aber auch zu genutzten Technologien wie beispielsweise das Git-System gefüllt.[1] Zusätzlich zu dem Wiki gibt es in jedem Repository ein Readme, das nochmal zusätzlich einen Überblick über das dortige Projekt gibt. Die Dokumentation des Programmcodes ist mittels dem Java Tool JavaDocs umgesetzt worden. Die damit generierten Dokumentationsseiten sind im jeweiligen Repository auf Github zu finden.

1.3 Unit Testing

Neu ist auch das, bisher nicht beachtete, automatisierte Testen des Programmcodes. Hierfür wird auf das Java Framework JUnit zurückgegriffen.[2] Durch das Nutzen eines solchen Frameworks ist es möglich automatisiert Programmcode auf die korrekte Ausführung zu testen. Gerade in einem großen System wie das Backend nimmt das einiges an arbeit ab, während es die Wartbarkeit deutlich erhöht.
Aktuell wird es sowohl im Backend, als auch in der Service-API-Library genutzt.
Im Backend werden zusätzlich noch die Mockito Bibliotheken genutzt, da es mit diesen

[1] Projekt Wiki in Redmine: http://194.95.221.229/redmine/projects/hablame/wiki/
[2] Projektseite von JUnit: http://junit.org/

ermöglicht wird zusätzlich noch die Controller-API zu testen, welche den Chatbot von außen über Requests erreichbar und ansprechbar macht.

1.4 Build Management mit Maven

Ein Problem im bisherigen Projekt war, dass Abhängigkeiten und der Buildvorgang nicht zentral geregelt wurden. So wurden genutzte Bibliotheken als Jar-Archive eingebunden. Dies hat zur Folge das diese Bibliotheken zu meist händisch in das Arbeitssystem geladen werden müssen. Um dies zu vereinfachen und den Buildvorgang zu vereinheitlichen wird im Backend sowie der Service-API-Library nun das Build Management Tool Maven verwendet.[3] Somit gibt es jetzt in beiden Projekten einen einheitlichen Punkt für die Konfiguration der Abhängigkeiten und des Buildvorgangs, die pom.xml.

[3]Projektseite von Maven: https://maven.apache.org/

2 Die Service-API-Library

Um die Nutzung des Backends zu vereinfachen wurde ein weiteres Teilprojekt erstellt: die Service-API-Library. Dieses Projekt bietet einen einfach zu handhabenden Client zur Nutzung des Bot-Services. Dafür bietet dieser Methoden zur jeder Schnittstelle des Backends.
Durch dieses Vorgehen ergibt sich der Vorteil, dass somit eine einheitliche, leicht zu nutzende Schnittstelle für Nutzerprogramme geboten wird, welche auch in weiteren Anwendungen integriert werden kann (z.B. in eine Desktopanwendung).
Als Basis für die Rest-Kommunikation wird die Bibliothek UniREST von Mashape verwendet.[4]

[4]Unirest: http://unirest.io/

3 Überarbeitung der Server-Architektur

Im Rahmen der Projektarbeit soll die serverseitige Architektur überarbeitet und mit der Nutzung von modernen Tools und Frameworks ausgestattet werden. Diese Optimierungen werden in den folgenden Unterkapiteln beschrieben.

3.1 Datenbank

Eine Anforderung an das Projekt war es, dass die bisherige Nutzung einer Oracle Datenbank abgelöst wird durch eine frei verfügbare Datenbank. Das Team entschied sich für das freie relationale Datenbankmanagementsystem MySQL in der aktuellsten produktiven Version 5.6.26 zum Zeitpunkt dieser Arbeit. Dies bringt den großen Vorteil mit sich, eine weitverbreitete quelloffene, statt einer proprietären Datenbank Software zu nutzen.

3.2 Einsatz moderner Technologien

Eine weitere Anforderung stellt die Nutzung moderner Frameworks und Technologien dar. Um einerseits einen guten Programmierstil sicher stellen zu können und andererseits die Kollaboration zu verbessern, sowie zukünftigen Projektteams den Einstieg maßgeblich zu erleichtern.

3.2.1 Das Spring Framework

Spring ermöglicht es durch so genannte Annotationen die Persistierung von Objekten unter Zuhilfenahme von Hibernate mit Object-Relational-Mapping zu vereinfachen. Zusätzlich wird das Testen mit Unittests durch JUnit simplifiziert und die Instanziierung von Objekten zur Laufzeit im Singleton-Pattern durchgeführt. Das Speichern und Lesen von Objekten kann unter Spring beispielsweise durch Interfaces mit der @Respository Annotation übernommen werden. So ist es möglich ohne eine SQL Anweisung selbst zu schreiben, Datenbankzugriffe zu realisieren und mit Objekten zu arbeiten. Dies erleichtert das Erstellen der Daten-Zugriffsschicht enorm. Diese Fähigkeit wird durch das Beerben des Interfaces CrudRepository, oder eines davon erbenden Interfaces erworben. Das Beerben eines dieser Interfaces verpflichtet zu den grundlegenden Datenbankoperationen Create, Read, Update und Delete eines spezifischen Objekttyps. Somit können POJOs in der Datenbank direkt abgespeichert werden. Diese werden in Form von separaten Model-Klassen angelegt, mit der @Entity Annotation versehen und sind somit als Datenbank-Entität gekennzeichnet. Weiterhin kann der Tabellenname, sowie einzelne Spaltennamen angegeben werden und Kardinalitäten zu anderen POJOs festgelegt werden, z.B. hat eine Kategorie mehrere Themengebiete.
Die Spring Hierarchie ist ein wesentlicher Bestandteil der neu designten Server-Architektur und soll hier deshalb zusätzlich Einzug erhalten. Wie bereits erwähnt können einzelne Interfaces mit der @Repository Annotation für ORM genutzt werden. Diese Datenbankschicht der Hierarchie ist die unterste der drei und wird von der darüber befindlichen Service-Ebene genutzt. Hier werden Klassen mit der @Service Annotation zum Service

ermächtigt, was dem Spring Kontext mitteilt, dass diese Services zur Laufzeit als Singleton Instanziiert werden. Services nutzen also die darunter liegenden Repositories für den Datenbankzugriff und verarbeiten die Daten weiter, bzw. führen andere gewünschte Operationen aus. Services stellen also Logikschicht dar. Diese Logikschicht wird von der API, den sogenannten Controllern genutzt, um Anfragen, sogenannte Web-Requests in Form eines GET oder POST, weiter bearbeiten zu können. Ein Controller hält die @Autowired Annotierten Services und dieser wiederum die Repositories und diese Objekte werden zur Laufzeit Instanziiert.

3.2.2 Testing mit JUnit

Wenn auch nicht bisher beachtet, ist das Testen von Programmcode einer der wichtigsten Bereiche und stellt sicher, dass bisher Implementiertes auch erwartungsgemäß funktioniert. Deshalb wurde JUnit – ein Framework zum Testen von Java-Programmen – eingeführt und einige Tests bereits implementiert, an denen sich zukünftige Teams orientieren können. Dank der Mockito Bibliotheken ist es möglich, auch die Controller-API zu testen, welche den Chatbot auch von außen über Requests erreichbar und ansprechbar macht.

3.3 Dokumentation

Eine Dokumentation stellt einerseits die Übersichtlichkeit und Nachvollziehbarkeit für Dritte und den Entwickler selbst her und optimiert andererseits die Einarbeitung nachfolgender Projektteams durch wesentlich verkürzte Zeiten enorm. Dies wurde in Form von zwei erwähnenswerten Punkten angestrebt, welche in den nachfolgenden Unterkapiteln erläutert werden sollen.

3.3.1 Javadocs und Codedokumentation

In der neuen Server-Architektur wurde jede implementierte Klasse mitsamt jeder dort realisierten Methode dokumentiert. Das ermöglicht beim Aufruf einer Klasse bzw. Methode eine Anzeige dieser Javadocs, was automatisch über die IDE realisiert wird. Wenn beispielsweise eine neue Extension für den Chatbot realisiert werden soll, kann der Entwickler über die Nutzung von bereits vorhandenen Erweiterungen und vor allem deren Dokumentiertem nachvollziehen wie dies umzusetzen ist.

3.3.2 Redmine

Als einzige bekannte und weit verbreitete Projektmanagement Software, die quelloffen und für den eigenen Gebraucht selbst aufgesetzt werden kann, kam Redmine in Frage, welche auf dem für das Projekt verfügbaren Windows 2008 R2 Server installiert und unter dem bestehenden Apache Webserver konfiguriert wurde. Diese ermöglicht es dem Projektteam über diverse Features effizient an diesem und auch künftigen Projekten zu arbeiten. Redmine enthält beispielsweise ein Ticketsystem, welches für Aufgabenverteilung und Zeitmanagement genutzt werden kann. Außerdem ist es dem Projektleiter

möglich, auf einfache Weise einzusehen, wie der Stand einzelner Aufgaben ist. Ein internes Wiki enthält wichtige Dokumentation über sensible Zugänge und Informationen, sowie Dokumente. Ein Kalender pro Projekt und eine E-Mail Funktion erhöht die Kommunikationsfähigkeit und steigert die Effizienz weiter. Über all diesen Möglichkeit steht natürlich die Zugriffssicherheit, welche ein integriertes Benutzermanagement darstellt, über welches neue Teammitglieder eingepflegt werden können und obsolete als inaktiv markiert, bzw. gelöscht werden können.

4 Überarbeitung der Android Application

4.1 Motivation und Ziel

Da die bisher vorhandene native Android Anwendung mit dem Hauptaugenmerk auf das Prototyping entwickelt wurde und aus diesem Grund kein spezielles Design sein eigen nennen kann, haben wir uns dazu entschieden, die Application von ihrem Status als Prototyp weg weiter zu entwickeln.
Das Hauptaugenmerk sollte auf einer einfachen und verständlichen Bedienung sowie der Erstellung eines individuellen Designs liegen. Bestand die vorhandene App bisher nur aus einer Activity in welcher alle Interaktionen mit dem System ausgeführt wurden, so wurde die neue Anwendung etwas entzerrt und auf zwei Activities aufgeteilt. Auf diese beiden wurde die Nutzerführung und das gesamte Bedienkonzept angepasst und überarbeitet.

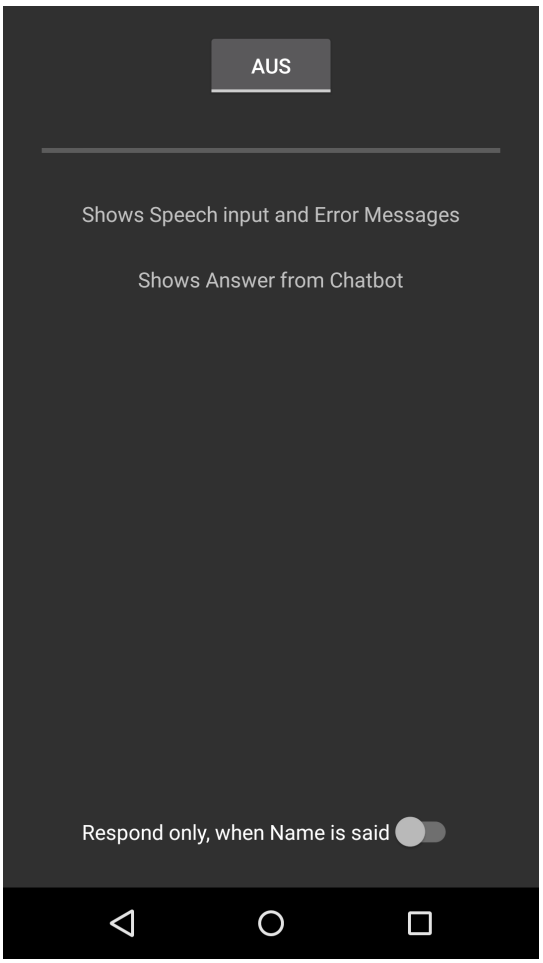

Abbildung 2: UI der bisherigen Android App

4.2 Aufbau

Wie unter 4.1 "Motivation und Ziel“ auf S.11 bereits erwähnt, sollte die Anwendung auf zwei Activities aufgeteilt werden. Desweiteren beinhaltet sie eine Klasse für den zur Spracherkennung notwendigen Service - "RecognitionService.java und eine Klasse in welcher die möglichen Fehler des Service beschrieben werden, ErrorDescription.java.

Bestandteile:

- MainActivity.java
- ConversationActivity.java
- RecognitionService.java
- ErrorDescription.java

4.2.1 MainActivity

Die MainActivity wird bei Programmstart automatisch geöffnet und bildet somit die Landing-Page. Sie besteht aus 3 grundlegenden Bereichen, nämlich dem Header, dem Body und dem Footer. Im Header sind die Schriftzüge "Háblame“ sowie "Das sprechende Faultier“ zu lesen.
Der Body beinhaltet zum einen das Logo und ein editierbares Textfeld welches Nutzereingaben entgegennimmt, in diesem Fall speziell den Nutzernamen. Der Body bietet somit im Gegensatz zum Header eine Interaktionsmöglichkeit für den Nutzer.
Der Footer besteht einzig aus einem Button welcher die komplette Displaybreite nutzt und bei einer onClick-Aktion auf die ConversationActivity weiterleitet. Diese Weiterleitung jedoch funktioniert nur wenn durch den Benutzer ein Username vergeben wurde, ansonsten wird eine Meldung auf dem Bildschirm ausgegeben, das kein Benutzername eingetragen ist.

4.2.2 ConversationActivity

Die ConversationActivity wird durch einen Intent in der MainActivity aufgerufen. Sie besteht ebenfalls aus Header Body und Footer. Änderungen gegenüber der MainActivity sind die Konversationsansicht im Body sowie eine andere Funktion des Buttons im Footer. Dieser bringt den Nutzer zurück in die MainActivity.
Nach aufrufen der ConversationActivity wird der RecognitionService gestartet und der in der MainActivity zuvor eingegebene Nutzername in die Konversationsansicht übernommen.

4.2.3 RecognitionService und ErrorDescription

Der RecognitionService welcher zur Erkennung der Sprache des Nutzers und deren Umsetzung in Text dient - und vice versa, wurde rein funktionell fast im Original übernommen. Einzig die Einbindung des alten Webservice wurde durch die Entwicklung und den Einsatz der Service-API-Library obsolet.
Die Klasse ErrorDescription dient lediglich dem RecognitionService dazu um im Fehlerfall eine detaillierte Fehlerbeschreibung an den Nutzer zurück zu liefern.

4.3 Funktionsweise

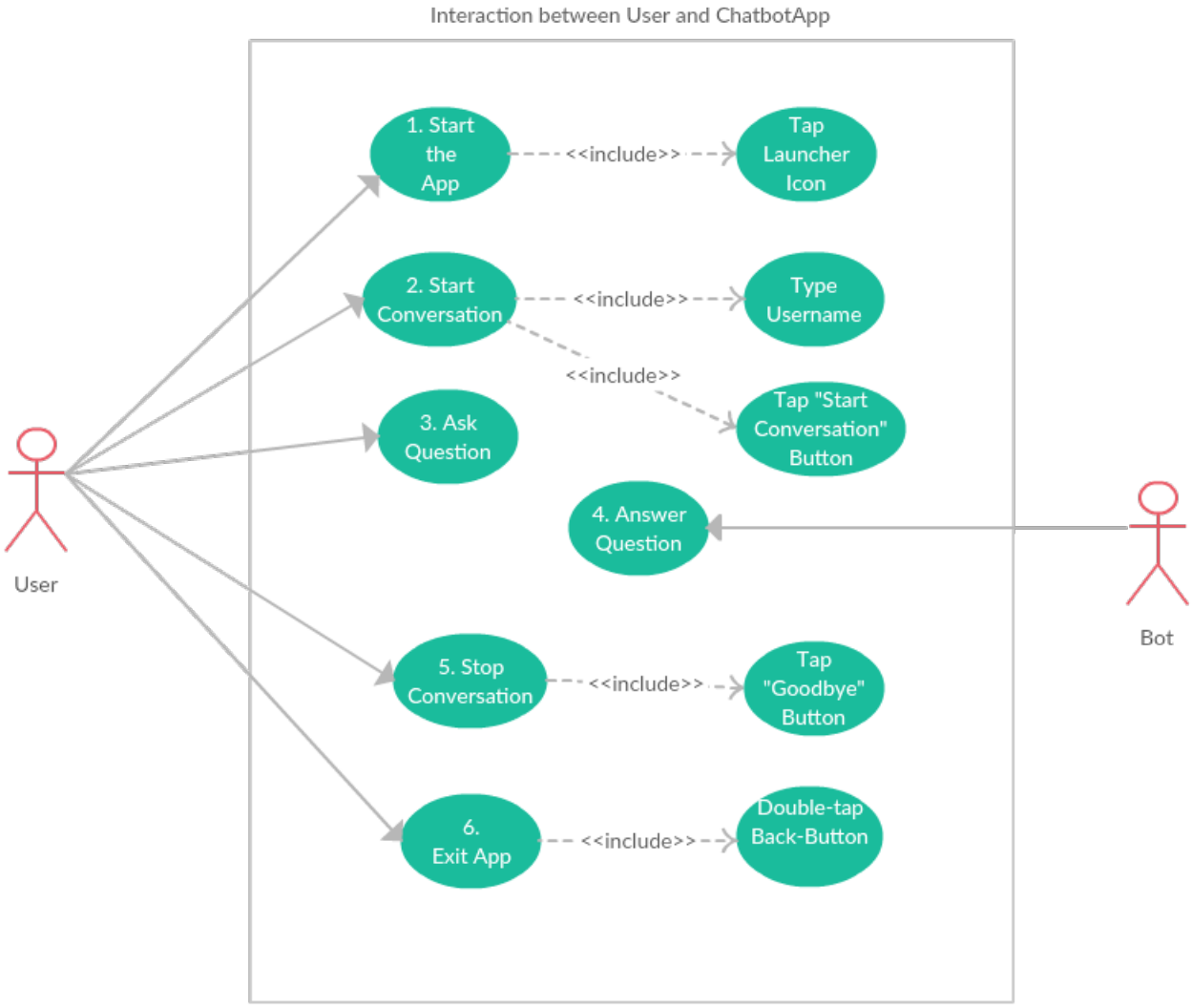

Abbildung 3: Use Case Diagramm zur Nutzung der App

4.4 Designanmerkungen

Kritikpunkte bei der Erstellung:

- Entzerrung der Nutzerführung
- Erstellung eines Logos
- Wiedererkennungswert schaffen
- Individualität

Das UI wurde so erstellt um eine möglichst hohe Darstellbarkeit zu gewährleisten und der Fragmentierung auf Android bei zu kommen. So wurden bis auf das Logo selbst alle Elemente mit Android eigenen Bordmitteln umgesetzt um responsive zu bleiben. Das Logo wurde aus selbigem Grund in allen Auflösungen von ldpi bis xxhdpi umgesetzt.
Eingesetzt wurde die, für kommerzielle sowie private Zwecke, frei nutzbare Schriftart Logobloqo2.ttf von seextwood[5]. Sowie die Grafik "Cartellone con bradipo“ von bradisoft[6], die auf http://www.fotolia.de erworbene Vektorgrafik ist lizenzfrei und zeitlich sowie räumlich ohne Beschränkung nutzbar.

4.4.1 Farben

Alle verwendeten Farben sind dem Google Design Guide entnommen[7].

MainActivity:

- Red 500
- Red 50
- Grey 400

ConversationActivity:

- Green 500
- Red 50
- Grey 400

Um die gestartete Konversation zu verdeutlichen, wurde für die ConversationActivity bewusst eine andere Hauptfarbe gewählt.

[5] http://www.dafont.com/de/logobloqo2.font
[6] https://de.fotolia.com/id/60029011
[7] https://www.google.com/design/spec/style/color.html#color-color-palette

Abbildung 4: UI der neuen Android App

5 Ausblick und Anregungen

Hier sind Anregungen für die Weiterentwicklung und mögliche Zukunftszenarien zu finden:

5.0.1.1 Backend

- TLS Zertifikat für den Webserver und den Tomcatserver
- Mehr Extensions nutzen für externes Wissen (wie Wetter)
- Mehrbenutzerbetrieb (gehaltenes Wissen des Bots pro User)

5.0.1.2 Service-API

-

5.0.1.3 Android Application

- Einstellungsmenü erstellen
- Funktion OnListenForName im Einstellungsmenü ein/ausschaltbar
- Sprachwahl im Einstellungsmenü
- Sprachausgabe-Stimme im Einstellungsmenü wählbar (Weiblich/Männlich)
- Mehrere Designs im Einstellungsmenü nach belieben wählbar

5.0.1.4 Allgemeines

- Pädagogenschnittstelle: Eigenständige Software, die die AIML-Syntax parst und Einträge bearbeiten kann