Aktueller Stand

Der Bot war ständig nicht erreichbar

Der Bot läuft via docker-compose in einer VM. Immer wenn der Bot nicht erreichbar war, startete er neu sobald sich jemand in die VM einloggte und war dann auch wieder erreichbar. Der Grund dafür war, dass docker-compose so konfiguriert war, dass die Container zwar neu starten sollten, aber nicht sofort wenn sie abstürzten (sondern in diesem fall dann eben beim Login durch einen docker-user).

Der Lösung bestand entsprechend in der Änderung der Configuration nach „restart-always“. https://github.com/beuthbot/beuthbot/pull/3

Gateway funktionierte nicht ohne Telegram-ID

Bei ersten Experimenten ist aufgefallen, dass wenn eine Nachricht an das Gateway geschickt wird und diese keine valide Telegram-ID enthielt, wurde die Nachricht ignoriert. Dieses war entgegen der Dokumentation, welche die Telegram-ID als optional definierte. Da dies für Testzwecke sehr hinderlich ist und im Projektverlauf zwei weitere Messenger (Discord und eigene Webseite) hinzugefügt werden sollen, galt dieses als eines der ersten Probleme die behoben werden sollten.

Durch eine Anpassung des Gateways bei der User-Abfrage wird eine valide Telegram-ID nicht vorausgesetzt. https://github.com/beuthbot/gateway/pull/2

Continous Deployment

Continous Integration & Deployment ist ein wichtiger Pfeiler für ein stabiles Production Environment. Durch eine CI/CD Pipeline kann sichergestellt werden, dass das Deployment nachvollziehbar, zuverlässig ausgeführt wird und bietet zugleich die Möglichkeit Qualitätssicherungs-Mechanismen in der Pipeline zu manifestieren.

Zu Beginn des Semesters wurde das Deployment manuell ausgelöst. Es gab mehrere Scripte, die diesen Vorgang unterschiedlich angingen. Die Updatestrategie ist grundsätzlich „alle Repositories updaten via git pull“ - leider gab es hier allerdings flaws, die dazu führten dass das lokale Repository „unrein“ wurde und nicht automatisch aktualisiert werden konnte.

Hinzu kam das Problem, dass die Ordnerstruktur unterschiedlichen Usern gehörte, immer denjenigen, die das Update ausgeführt haben, bei dem die Dateien erstmals im Repository auftauchten. Dadurch scheiterten Updates zusätzlich, wenn „der falsche user“ das update versuchte bzw. „die falschen dateien“ im update aktualisiert würden.

Wir haben via Github-Actions eine CI/CD Pipeline erstellt, die den Deployment Prozess in 3 Stages ausführt: Build, Test, Deploy. Die Test-Stage ist via Makefile angebunden, so dass EntwicklerInnen neue Tests simpel in eine zentrale Stelle eintragen können. Das Makefile ist nun Single Point of Truth. Die teilweise widersprüchlichen Scripte von vorher wurden aufgeräumt

Code Änderungen im Repo (Pull Request): https://github.com/beuthbot/beuthbot/pull/4

Mithilfe eines Selfhosted-Runners welcher auf dem BeuthBot-Server installiert wurde, ist es jetzt möglich diesen Prozess zu automatisieren. Sobald ein Git-Commit mit einem Versions-Tag gepusht wird, wird dies vom Runner erkannt und der Deploy-Prozess wird angestoßen. Der Runner führt die Aktualisierung auf der VM des Bot durch.

Doku zur Runner Config:
https://github.com/beuthbot/beuthbot/blob/master/.documentation/github-runner.md

Public Domain

Es gab keine Public Domain zum Telegram Gateway. Diese brauchen wir aber um a) eine Landing Page für den Bot zu hosten und b) das Gateway von Chatbots ansprechen zu können, die nicht in der VM gehosted sind. Damit dies funktioniert wurde ein Proxy-Pass für die Default-Domain von **https://beuthbot.ziemers.de/** angelegt. Damit wird jetzt folgender Curl Möglich: *$ curl https://beuthbot.ziemers.de/message -X POST -H „Content-Type: application/json“ –data „{\“text\„:\“Wie wird das Wetter morgen?\„}“*

Text To Speech Recherche

- Say.js
  - https://www.npmjs.com/package/say
  - https://github.com/marak/say.js/
2. Web Speech API
   - https://wiki.selfhtml.org/wiki/JavaScript/Web_Speech
3. Text2Speech
   - https://www.npmjs.com/package/text-to-speech-file

Speech To Text Recherche

In diesem Projekt soll es ermöglicht werden, dass Nutzer ebenfalls Sprachnachrichten an den BeuthBot schicken können um mit diesem zu interagieren. Um dieses umzusetzen ist eine sogenanntes Speech-To-Text-Programm erforderlich, welche Sprachnachrichten in Text umwandelt. Diese umgewandelten Nachrichten können dann wie normale Textnachrichten vom BeuthBot weiterverabeitet werden. Da es sich hierbei um ein äußerst kompliziertes technisches Problem handelt, bei dem Ansätze mit statischen Algorithmen nicht anwendbar sind, werden ausschließlich Ansätze des DeepLearning angewendet. Neben vielen Cloud-Lösungen von namenhaften Anbietern wie Amazon und Google gibt es ebenfalls eine Reihe von OpenSource-Lösungen, welche privat gehostet werden. Dieses bietet mehrere Vorteile. Zum einen, fallen keine Gebühren für die Nutzung an, da alle Berechnungen lokal auf dem BeuthBot-Server ausgeführt werden. Zum anderen ist Datenschutz leichter umzusetzen, da alles lokal verarbeitet wird und keine Daten an externe Services weitergegeben werden. Der Recherche ergab eine Vielzahl an Lösungen, jedoch sind nur drei für das Projekt geeignet, da nur für diese ein vortrainiertes Modell für die deutschen Sprache verfügbar ist. Diese sind:

Mozilla Voice STT (DeepSpeech)

- https://github.com/mozilla/DeepSpeech
- https://github.com/AASHISHAG/deepspeech-german

- Entwickler: Mozilla
- Opensource
- Offline nutzbar
- Viel Dokumentation
- Deutsches Modell
- WER: 15%
- Zukunft ungewiss (https://www.ghacks.net/2020/08/24/the-future-of-mozillas-speech-to-text-project-deepspeech-is-uncertain/)

Kaldi

- https://github.com/kaldi-asr/kaldi
- http://kaldi-asr.org/doc/about.html
- http://zamia-speech.org/asr/
- Entwickler: Kaldi
- Opensource
- Offline nutzbar
- Deutsche Modelle
- WER: 8,44%

Wav2Letter

- https://github.com/facebookresearch/wav2letter
- http://zamia-speech.org/asr/
- Entwickler: Facebook Research
- Opensource
- Offline nutzbar
- Deutsche Modelle
- WER: 3,97%

Während des Projekts gilt es, diese drei Lösungen zu testen, miteinander zu vergleichen und darauf basierend die beste Lösung auszuwählen und im BeuthBot zu implementieren.

Die STT-Programme ohne verfügbares deutsches Modell sind folgende:

Espresso

- https://github.com/freewym/espresso
- Entwickler: Freewym
- Opensource
- Offline nutzbar
- Kein deutsches Modell

OpenSeq2Seq

- https://github.com/NVIDIA/OpenSeq2Seq
- Entwickler: NVIDIA
- Opensource
- Offline nutzbar
- Kein Deutsches Modell

Word Error Rate (WER)

Um die Qualität eines STT-Modells zu messen, wird der sogenannte Word Error Rate (WER) Wert verwendet. Dieser Wert gibt an, basierend auf dem Testdatensatz, wie viele Wörter prozentual falsch erkannt werden. Zum Beispiel, wenn bei einem Satz, welcher 100 Wörter enthält, 10 Wörter falsch erkannt werden, dann beträgt der WER-Wert 10%.

Unten befindet sich eine Auflistung von WER-Werten von kommerziellen STT-Diensten für die englische Sprache aus dem Jahre 2017. Darunter befindet sich ebenfalls die WER-Werte der recherchierten OpenSoruce-Lösungen. Da alle Dienste unterschiedliche Datensätze zum Training und Test verwenden, sind diese Ergebnisse nicht komplett vergleichbar, aber sie bieten eine grundsätzliche Übersicht über die Performance der OpenSource-Lösungen.

- Google (8%)
- Microsoft (5.9%)
- IBM (5.5%)
- Apple (5%)
- Baidu (16%)
- Hound (5%)
- Mozilla Voice STT (15%)
- Kaldi (8,44%)
- Wav2Letter (3,97%)

Quelle:
https://askwonder.com/research/current-voice-recognition-word-error-rates-google-amazon-microsoft-ibm-apple-5b88trj0t

DokuWiki Plugins

edittable

Standartmäßig werden im Ziemer's-Wiki alle Tabellen mittels MarkDown angelegt. Da dieses jedoch besonders bei großen Tabellen sehr fehleranfällig ist, wurde das edittable-Plugin installiert. Dieses erlaubt es mittels einer grafischen Benutzeroberfläche Tabellen anzulegen und zu bearbeiten. So entstandene Tabellen werden dann als normale MarkDown-Tabellen im Wiki abgelegt. Dieses erleichterte das Arbeiten mit Tabellen ungemein.
https://www.dokuwiki.org/plugin:edittable

PageBreak

Die finale Abgabe des Zwischenberichtes sollte in Form eines PDFs abgeben werden. Das Ziemer's-Wiki hatte bereits das DW2PDF-Plugin installiert, welches es auf einfache Weise ermöglicht jede beliebige Wiki-Seite als PDF zu exportieren. Hierbei ergab sich jedoch das Problem, dass alle Kapital ohne große Abstände hintereinander in das PDF geschrieben wurden, welches die Übersichtlichkeit stark beeinträchtigt hat. Um dieses Problem zu lösen, wurde das PageBreak-Plugin im Ziemers-Wiki installiert. Dieses erlaubt es, mittels des pagebreak-Tags, dem DW2PDF-Plugin mitzuteilen wann ein Seitenumbruch passieren. Damit konnten wir nach

jedem Kapitel und Feature-Tabelle einen Seitenumbruch hinzufügen. Dies hat die Übersichtlichkeit des Zwischenberichtes deutlich erhöht.
https://www.dokuwiki.org/plugin:pagebreak

Nutzungshinweis: Auf dieses vorliegende Schulungs- oder Beratungsdokument (ggf.) erlangt der Mandant vertragsgemäß ein nicht ausschließliches, dauerhaftes, unbeschränktes, unwiderrufliches und nicht übertragbares Nutzungsrecht. Eine hierüber hinausgehende, nicht zuvor durch *datenschutz-maximum* bewilligte Nutzung ist verboten und wird urheberrechtlich verfolgt.