Geplanter Stand

Präambel

Neben einigen wünschen der Projektleitung konnte die Projektgruppe eigene Schwerpunkte in die Feature Planung einbringen. Die nachfolgende Feature-Planung ist ein resultat der folgenden Überlegungen

Content First

Der BHT-Bot besteht bisher aus lediglich zwei Services, die Content zur Verfügung stellen: Mensa- und Wetter-Service. Aufgrund der Covid19-Pandemie findet keine Präsenzveranstaltungen an der Hochschule statt und in Folge dessen hat die Mensa geschlossen, der Bot-Service ist entsprechend auch eingestellt. Defakto kann der BHT-Bot damit derzeit ausschließlich das Wetter ansagen. Für uns ist es daher umso wichtiger dieses Semester neuen Content zu erzeugen bzw. nutzbare Features in den BHT-Bot zu integrieren, damit dieses Projekt überhaupt eine Daseinsberechtigung bekommt.

Hands-On & Dokumentation

Wenn man neu in ein Projekt kommt gibt es viel Dokumentation aufzuarbeiten, als auch undokumentierte Zustände zu entdecken. Wir hatten Gelegenheit den BHT-Bot in einem Hands-On Workshop von Lukas Dankwerth aus SoSe2020 zu bekommen. Im Workshop haben wir uns beispielsweise angeschaut a) Warum der Bot eine Telegram-ID braucht in allen Requests b) Dass der Bot derzeit zwei fast identische Datenbankservices hat, die zusammengeführt werden könnten c) Dass die Services derzeit inkonsistenzen aufweisen, die bis zu essentiellen Debugging-Strategien wie der Request-History reichen, die nicht gepflegt wurde d) Dass Dokumentation auf verschiedene Projekte und Plattformen verteilt wurde.

Ohne einen persönliche Hands-On-Workshop ist eins der größten Probleme um ins Projekt zu finden, dass es verschiedene Stellen gibt, an denen Dokumentation, Code, Infrastruktur, etc. verteilt ist, aber keine zentrale Stelle, die als „Single Point of truth“ registerartig auf die weiteren Quellen verweist.

Wir wollen diese Situation verbessern, indem wir falsche und für uns nicht-nachvollziehbare Dokumentation verbessern, Wikistrukturen bereinigen, eine Landing-Page für den Bot unter einer zentralen Bot-Domain verfügbar machen und Abläufe und Strukturen in Code gießen.

Redundanter Code / Wartbarkeit der Microservices

Beim Studium der einzelnen Services wurde sichtbar, dass fast ausnahmslos jeder Service des BHT-Bot die gleiche Grundstruktur hat: Alle Services stellen eine JSON-REST-Schnittstelle zur Verfügung, die via NodeJS + Express Framework implementiert ist. Vergleicht man die Services, sieht man, dass insbesondere Services, die Inhalte ausspielen sollen einer identischen Struktur folgen (müssen), dies aber individuell handhaben. Dadurch ist der Boilerplate Code für jeden Service unnötig hoch und

zugleich ist das Warten der Services bei Änderung von globalen Schnittstellen mit hohem individuellen Aufwand verbunden. Wir wollen diese Common-Funktionalitäten identifizieren und in zentralen Bibliotheken bzw. Frameworks bündeln.

Qualitätsmanagement

Bisher sind alle Services lose miteinander verbunden, jeder Service implementiert die Kommunikation für sich selbst, die Integration erfolgt durch manuelles Deployment, es gibt keine Typisierung, keine (automatischen) Regeln zur Collaboration, keine Versionierung, kein Unit-Testing, kein Monitoring, .. Kurz gesagt: Der BHT-Bot hat bislang keine Form von Qualitätssicherung. Dadurch ergeben sich natürlich viele Baustellen, die wir, insbesondere auch angesichts unserers Anspruchs „Inhalte in den Bot zu bringen“ kaum befriedigend erfüllen können. Wir wollen dennoch einen Beitrag zur Verbesserung der aktuellen Situation erbringen:

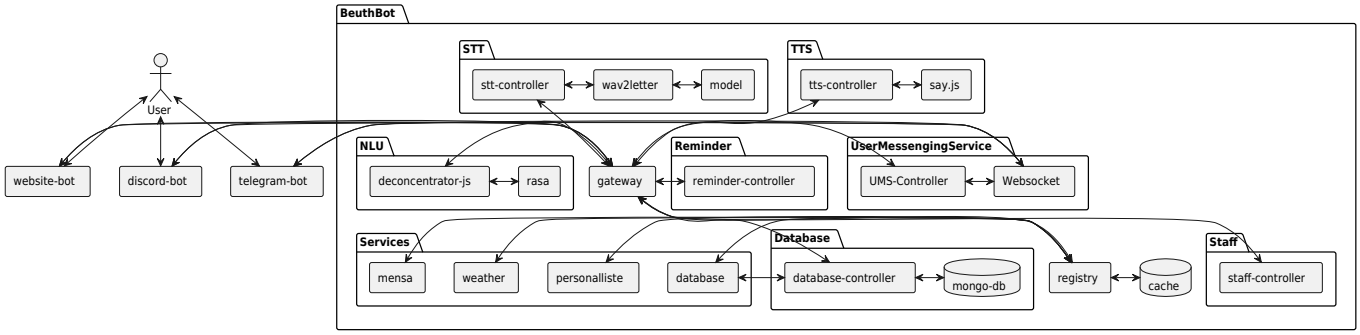
1. Wir schreiben typisierte Libraries, die eine zentrale Dokumentation der Kommunikation mit dem Bot darstellt.
2. Alle Libraries und Frameworks werden durch Linting-Regeln in ein, für alle Entwickler einheitliches, Format gebracht
3. Alle Libraries und Frameworks haben automatische Unit-Tests, als auch Prüfung der Testabdeckung, wir verlangen mindestens 80% Testabdeckung, Ziel ist 100% zu erreichen
4. Wir wechseln vom bisherigen manuellen Deployment zu einer automatischen CI/CD Pipeline. Diese Pipeline beinhaltet auch eine Testing-Stage, in die wir dieses Semester mindestens einen Service anbinden werden
5. Releases werden durch Versionierte Git Tags erzeugt (und dann automatisch deployed) - Die Tags werden semantisch versioniert
6. Neue Features werden via Pull Request in das Repository übernommen. Nach Möglichkeit werden die Requests durch ein Projektmitglied reviewd und freigegeben

Übersicht geplante Tasks und Abhängigkeiten

ID	Name	Priorität	Initiale Aufwandsschätzung (in Tage)	Abhängigkeiten	Wird verticketet von
BOT-16	Support mehrerer Messenger-Typen durch Umbau der Benutzererkennung	1	1		Lukas Danke
BOT-12	Rasa Update auf 2.0	1	1		Robert Halwaß
BOT-30	Chatbot Library: Vereinheitlichung der Kommunikation von Javascript Chatbots mit dem Gateway	2	1	BOT-16	Dennis Walz
BOT-74	Webseite	2	3	BOT-16, BOT-30	Rim Khreis

ID	Name	Priorität	Initiale Aufwandsschätzung (in Tage)	Abhängigkeiten	Wird verticketet von
BOT-49	User-Messenger-Service: Nachricht proaktiv, requestunabhängig an Clients senden	2	2.5		Dennis Walz
BOT-37	Discord Integration	2	2	BOT-16, BOT-30	Dennis Walz
BOT-43	Erstellung eines Common-Frameworks für (Content-)Services	2	1.5		Dennis Walz
BOT-13	Komponente zur Umwandlung von Sprache zu Text (STT)	3	3	BOT-43	Robert Halwaß
BOT-23	Komponente zur Umwandlung von Text zu Sprache (TTS)	3	3	BOT-43	Alexis Popovski
BOT-55	Erinnerungs-Service: Behandelt „erinnere mich“ Befehle und erinnert bei Fälligkeit autonom	3	3	BOT-12	Dennis Walz
BOT-82	Termin-Scraper, der automatisch Erinnerungen aus öffentlichen Quellen bezieht	3	1	BOT-12, BOT-55	Dennis Walz
BOT-89	Moodle iCal import als Erinnerungen	3	1	BOT-12, BOT-55	Dennis Walz
BOT-75	Begrüßungsnachricht	4	1		Rim Khreis
BOT-11	Universelles Scraper & Download	4	3	BOT-43	Alexis Popovski
BOT-15	Personalliste der Beuht-Hochschule im Beuthbot abrufbar machen	4	2	BOT-12, BOT-43	Lukas Danke

Architektur



Aufgaben Spezifikation

BOT-16: Support mehrerer Messenger-Typen durch Umbau der Benutzererkennung

Aktueller Stand:

Der BeuthBot unterstützt aktuell nur eine Verwendung über Telegram. Die Telegram-ID des Benutzers wird in der Datenbank gespeichert.
Diese wird dann verwendet, um individuelle Informationen zu dem Benutzer abrufen zu können.

Geplanter Umbau:
In Zukunft sollen für den BeuthBot mehrere Kontaktmöglichkeit zur Verfügung gestellt werden. Damit ein Benutzer aber auch unabhängig vom Messenger erkannt wird muss der BeuthBot angepasst werden:
1. Umbau der Datenbank damit mehrere Messenger-IDs gespeichert werden können
2. Umbau der Erkennung des Messengers
3. Vereinfachen der Erkennung, damit ein zukünfiger Support von neuen Messengern einfach und schnell erfolgen kann.

Initiale Schätzung	3 Tage
Technologien	* Javascript * mongodb
Abhängigkeiten	keine
Anforderungen	* einzelne Speicherung der Telegram-ID aus der Datenbank entfernen * Neue Spalte zur Verwaltung aller Benutzer-IDs * vereinfachter Einbau von neuen Messengern gewährleisten * Abhängigkeit von der Telegram-ID entfernen * Mehrere Messenger verfügbar machen * Anmeldung eines neuen Benutzers * Löschen eines Benutzers/Messengers * Anmeldung eines neuen Messengers für ein bestehenden Benutzer

Tasks	* BOT-20 - Umbau der Datenbank - Speicherung von mehreren Accounts für einen Benutzer * BOT-21 - Anmeldung eines neuen Accounts für einen Benutzer * BOT-22 - Löschen eines bestehenden Accounts für einen Benutzer * BOT-97 - Methodik zur einfachen Erweiterung der unterstützten Messenger

BOT-12: Rasa 2.0 Update

Um die neusten Funktionen und Fixes von Rasa zu benutzen, ist ein Update von Version 1.6 auf 2.0 notwendig.

Initiale Schätzung	3 Tage
Technologien	* Python * Rasa
Abhängigkeiten	keine
Anforderungen	* Kompatibilität mit bestehenden NLU-Trainingsdaten erhalten * Mögliche JSON- und MarkDown-Dateien in YAML-Datein umwandeln
Tasks	* BOT-62: Config Anpassen * BOT-63: Duckling updaten * BOT-64: Modell neu trainieren * BOT-116: Performance zwischen Version 1.6 und 2.0 vergleichen

BOT-30: Chatbot Library: Vereinheitlichung der Kommunikation von Javascript Chatbots mit dem Gateway

Problem: Derzeit muss jede Anwendung, die den BHT-Bot als Chatbot implementieren möchte selbst implementieren, wie die Kommunikation zwischen Anwendung und Gateway aussieht, als auch die Schnittstellen Parameter in Anfrage und Antwort.
Um diese Implementationsredundanz zu verhindern, wird die Kommunikation und Typedefinitionen in einer zentralen Javascript Bibliothek zusammengefasst.
Dies ermöglicht auch weitere geplante Funktionalitäten (wie der asymmetrische Kommnuikationskanal zur Requestunabhängigen Server → Client Kommunikation) zentral entwickelt und mittels Dependency Management schnell in die ChatClients überführt werden.

Initiale Schätzung	1 Tag
Technologien	* Javascript * Typescript
Abhängigkeiten	keine
Anforderungen	* Die Library lässt sich in Node und Browser Javascript einbinden * Die Library nutzt semantische Versionierung zur Ermöglichung von Non-Breaking-Updates * Die fertige Library lässt sich via Dependency-Management (npm/yarn/webpack) userseitig einbinden und updaten * Die Library enthält typisierte (typescript) Entitäten für Common Request und Response Format(e) * Die Library enthält Unit-Tests für essentielle Funktionen und Typen * Die Library ist dokumentiert, sowohl was Nutzung, als auch Contribution angeht * Die Library verbessert die Collaboration mittels Linting-Regeln und Workflow-Scripten
Tasks	* BOT-33 Library Usage Dokumentieren * BOT-34 Library in Discord Bot integrieren * BOT-35 Library in Telegram Bot integrieren * BOT-36 Library in Website integrieren * BOT-31 Common Funktionalität / Use Cases identifizieren * BOT-32 Typescript Library für Bot erstellen

BOT-74: Webseite

Um eine komplette Übersicht für alle genutzten BeuthBot-Resourcen zu haben, soll eine Webseite zur Präsentation dieser Ressourcen erstellt werden. Zu den Ressourcen zählen das Ziemer's Wiki, Telegram, Discord, Github und die Implementation des Chatbots.

Initiale Schätzung	3 Tage
Technologien	* TypeScript * JavaScript
Frameworks/ Libraries	* Angular * Bootstrap
Abhängigkeiten	* BOT-10 * BOT-30
Anforderungen	* Jeder Ressource wird ein Abschnitt gewidmet, welcher Infos & einen Link zu der jeweiligen Ressource enthält * Anschauliches, Einheitliches und responsive Design der Webseite * leicht austauschbare Komponenten * Das System sollte eine Ansicht innerhalb von 3 Sekunden laden * Das System sollte gut dokumentiert sein * Das System sollte leicht zu verstehen sein
Tasks	* BOT-76 Webseite Einrichten * BOT-77 Infos zum Wiki * BOT-78 Infos zu Telegram * BOT-79 Infos zu Discord * BOT-80 Infos zu GitHub * BOT-81 Implementation Chatbot * BOT-130 Notifications wenn Nachricht vom Chatbot * BOT-131 Responsive

BOT-49: User-Messenger-Service: Nachricht proaktiv, requestunabhängig an Clients senden

Der BHT-Bot kann bisher nur passiv auf Anfragen warten und diese dann beantworten. Zur Implementierung von asymmetrischer bzw. request-unabhängiger Kommunikation benötigt der Bot einen neuen Service, der als Schnittstelle für diese Art von Kommunikation dient.

Initiale Schätzung	2.5 Tage
Technologien	* Javascript * Websockets * Docker
Abhängigkeiten	* BOT-30
Anforderungen	* Der User-Messenger-Service kann eine Nachricht an einen User (via Client-Unabhängiger User-ID) senden * Wenn ein User mehrere Clients benutzt, wird die Nachricht an alle Clients gesendet * Wenn ein User (über längere Zeit) nicht erreichbar ist wird die Kennung entfernt * Wenn eine Nachricht nicht an einen User gesendet werden kann bekommt der auslösende Service diese Information unterscheidbar zwischen a) Der Nutzer ist gerade nicht erreichbar b) Der Nutzer ist dauerhaft nicht erreichbar (gelöscht) c) Der Dienst ist generell unhealthy * Der Service wird als Docker Image via docker-compose in die BHT-Bot Infrastruktur integirert. Er ist Teil des BHT-Bot Repositories * Die Funktionalität des Services wird auf Clientseite in die Common-Chatbot-Library (BOT-30) * Alle bestehenden ChatBot-Services werden an den Messenger Service angebunden (Telegram, Discord)
Tasks	* BOT-50 Websocket Registry für ChatBotClients * BOT-51 REST-Service für Nachrichtenversand * BOT-52 Implementation der Websocket-Registrierung in Common-Library für Chatbots * BOT-53 Implementierung der Common-Library-Websocket-Registrierung in TelegramBot * BOT-54 Implementierung der Common-Library-Websocket-Registrierung in DiscordBot * BOT-56 Dokumentation Usage Service * BOT-110 Deployment / Release

BOT-37: Discord Integration

Discord ist eine weit verbreitete Kommunikatinsplattform, auf der Nutzer sich in „Servern“ vernetzen und dort meist thematisch organisiert kommunizieren können. Die Projektgruppe des WS2020 ist selbst Teil der Zielgruppe des Discord-Messengers, wodurch sich diesese Plattform besonders eignet um einen weiteren Chatservice (neben Telegram) an den BHT-Bot anzubinden. Eine Implementation des Chatbots innerhalb der Discord-Struktur steigert somit zum Einen die Verbreitung(smöglichkeit) des BHT-Bot und bietet gleichzeitig eine gute Möglichkeit Debug-Bot-Instanzen im präferierten Messenger zu betreiben.

Initiale Schätzung	2 Tage
Technologien	* Javascript * Docker
Abhängigkeiten	* BOT-30
Anforderungen	* Der Discourse Bot benutzt die zu entwickelnde zentralisierte Library zur Gateway Kommunikation um Coderedundanz mit dem Telegram Bot zu verhindern * Der Discourse Bot kann (direkte) Nutzer-Nachrichten mittels Gateway verarbeiten und antwortet dem User entsprechend * Wenn keine Verbindung mit dem Gateway besteht oder Fehler bei Anfragen auftreten reagiert der Chatbot durch Präsentation einer hilfreichen Fehlermeldung * Der Discourse Bot wird äquivalent zum Telgram Bot in das BHT-Bot Universum mittels Docker + compose integriert * Der Discourse Bot ist nicht Teil des BHT Bot, er wird als paralleler, unabhängiger Service betrieben * Alle Credentials und Urls/Ports werden aus dem Environment bezogen, es gibt keine hard-coded Referenzen zu Strukturen des BHT-Bot Gateways
Tasks	* BOT-38 NodeJS Chatbot erstellen * BOT-39 Docker Container + Compose für Container erstellen * BOT-40 Bot Usage dokumentieren * BOT-41 Bot Account anlegen für release (https://discord.com/developers/applications) * BOT-42 Bot Container in Beuth-Docker-Netzwerk einbinden (release)

BOT-43: Erstellung eines Common-Frameworks für (Content-)Services

Services im BHT-Bot kommunizieren alle über REST-Schnittstellen. Diese sind alle als Express-Anwendung mit JSON-Kommunikation implementiert, was für viel Code-Redundanz führt. Gleichzeitig benutzt kein Service strukturierte Darstellungen der Schnittstellen, wie Anfragen und Antworten zum Gateway, User Daten oder Rasa-Intents.
Ein spezieller, repititiver, Unterfall der Microservices sind solche, die Content bereitstellen. Diese erhalten alle Nachrichten vom Gateway und senden ihre antworten auch dort wieder zurück. Request- und Response sind durch das Gateway definiert, die resultierende Struktur ist entsprechend für alle Contentservices prinzipiell Identisch.
Zur Vermeidung von Code-Redundanzen und Erleichterung des „Kick-Off“ eines neuen Content-Services sollen die Common Funktionen und Entitäten in ein Framework gegossen werden

Initiale Schätzung	1 Tag
Technologien	* Javascript * Typescript * Dockerfile
Abhängigkeiten	* BOT-37
Anforderungen	* Das Framework implementiert eine NodeJS Express REST-API, äquivalent zu den existierenden Content-Services * Das Framework lässt sich in NodeJS Anwendungen via Dependency-Management einbinden (npm/yarn) * Das Framework abstrahiert den (Express) Server und dessen Routing, so dass ein Contentservices nur noch die Response implementieren muss * Das Framework implementiert die Schnittstelle zum Datenbankservice um a) Userdaten zu speichern/abzurufen und b) eigene Daten zu speicher und abzurufen * Das Framework füllt die “debug-history” der Requests so, dass ein Service dieses Feature zwangsweise implementiert / nutzt * Requests, Responses, User, Rasa-Intents und ggf. weitere Entitäten werden durch das Framework als typisierte (typescript) Objekte definiert * Das Framework wird in allen bestehenden und geplanten Content-Services implementiert: Wetter, Mensa, Reminder * Die Nutzung des Frameworks ist verständlich dokumentiert * Die Contribution wird mittels Linting, Dokumentation und Build-Scripts erleichtert * Das Framework nutzt types aus der (noch zu entwickelnde) BHT-Bot Library um Redundanzen zwischen Client-Bibliothem und Service-Framwork zu vermeiden
Tasks	* BOT-44 Common Code, Features, Entitäten identifizieren → Use Case ableiten * BOT-45 Typisiertes Javascript Framework erstellen * BOT-46 Framework einbinden in Weather Service * BOT-47 Framework einbinden in Mensa Service * BOT-48 Framework einbinden in Reminder Service * BOT-108 Framework dokumentieren * BOT-117 Request History implementieren - Nutzung enforcen

BOT-13: Komponente zur Umwandlung von Sprache zu Text (STT)

Es soll ermöglicht werden, dass Benutzern neben Textnachrichten auch mittels Sprachnachrichten mit dem BeuthBot kommunizieren können. Dabei sollen die Sprachnachrichten mittels eines neuen Services in Text übersetzt werden und dann wie andere Textnachrichten verarbeitet werden. Hierzui sollen 3 bekannte STT-Frameworks (Kaldi, Mozilla Voice STT und Wav2Letter) getestet und vergleichen werden. Basierend darauf soll eine Entscheidung getroffen werden, welches Framework schlussendlich in der Production-Environment verwendet werden soll. Das Framework wird dann in Form eines neuen Micro-Services in den BeuthBot integriert.

Initiale Schätzung	3 Tage
Programmiersprachen	* Python (Mozilla Voice STT) * C++ (Kaldi, WAV2Letter) * Kaldi * Mozilla Voice STT * WAV2Letter
Abhängigkeiten	* BOT-43: Erstellung eines Common-Frameworks für (Content-)Services
Anforderungen	* Die Übersetzung soll mittels neuronaler Netzte geschehen * Nur Sprachnachrichten auf Deutsch sollen übersetzt werden * Das verwendete Framework muss OpenSource sein und Lokal auf dem BeuthBot-Server ausführbar sein * Es soll keine Model-Adapation durchgeführt werden
Tasks	* BOT-69 WAV2Letter testen * BOT-70 Mozilla Voice testen * BOT-73 Kaldi testen * BOT-71 Framework aussuchen * BOT-122 Micro-Service erstellen * BOT-123 Sprachnachricht in WAV umwandeln

BOT-23: Komponente zur Umwandlung von Text zu Sprache (TTS)

Neben der bereits vorhandenen Funktion Textnachrichten vom Beuthbot zu erhalten, sollen Nutzer die Möglichkeit bekommen ebenfalls Sprachnachrichten zu empfangen. Hierfür wird eine Komponente zur Konvertierung von Text in Sprache (Eng: „Text-To-Speech (TTS)) benötigt. Dieses Feature soll dem Nutzer in künftigen, dem Beuthbot hinzugefügten, Messenger-Diensten zur Verfügung stehen. Zur Umsetzung soll optimalerweise von einer Library Gebrauch gemacht werden, welche den Anforderungen gerecht wird.

Initiale Schätzung	1 Tag
Technologien	Javascript
Abhängigkeiten	* BOT-43: Erstellung eines Common-Frameworks für (Content-)Services
Anforderungen	* Support für die deutsche Sprache * Sprachnachrichten lassen sich als MP3- und WAV-Dateien exportieren * Das Feature ist bzgl. Opt-in oder Opt-out in den Hilfe-Texten/Begrüßungsnachrichten des Beuthbots dokumentiert
Nice to Haves	Sprachgeschwindigkeit und Stimme des Sprechers sind konfigurierbar
Tasks	* BOT-24 Recherche nach geeignetem Tool (TTS) * BOT-25 Eigene Implementierung (TTS) * BOT-98 Integration in Beuthbot

BOT-55: Erinnerungs-Service: Behandelt „erinnere mich“ Befehle und erinnert bei Fälligkeit autonom

Erinnerungen schedulen zu können ist ein typischer, weil praktischer, Anwendungsfall in beliebten (Business) Kommunikations-Services wie Slack oder Mattermost. In beiden Fällen wird diese Funktionalität durch die hauseigenen Reminder-Bots zur Verfügung gestellt.
Um die Featuredichte des BHT-Bot zu erhöhen wird ein Reminder-Service erstellt, durch den identische Funktionalität wie bei genannten Diensten zur Verfügung stellt. Durch die Multi-Messenger-Fähigkeit des BHT-Bot wird dieses Feature somit auch für User angeboten, deren Kommunikations-Plattform keinen eigenen Reminder-Bot anbietet.
BeispielAnfragen:
* Erinnere mich am 22.10. an die Klausur in Mathe
* Erinnere mich jeden Donnerstag um 18 Uhr an den Ballettkurs
* Erinnere mich jedes Jahr am 01.01 an den Geburtstag meiner Mutter.
* Erinnere mich in 10 Tagen das Probeabonnement zu kündigen
* <Erinnere> <Zeitpunkt/Zeitspanne/Interval> <Thema>

Initiale Schätzung	3 Tage
Technologien	* Javascript * Docker * Rasa * MongoDB * Cronjob
Abhängigkeiten	* BOT-43 * BOT-30 * BOT-12 * BOT-49
Anforderungen	* Erfolgreiche „erinnere“-Anfragen werden vom Dienst durch Bestätigung der erkannten und persistierten Daten beantwortetoder * Fehlerhafte “erinnere”-Anfragen werden durch ein Mini-Usage-Tutorial beantwortet, damit der User seine Anfrage korrigieren kann * Erinnerungen werden auf user-ebene (clientunabhängig) gespeichert, so dass ein User die gleichen Erinnerungen in allen genutzten Clients zur Verfügung hat * Erinnerungen werden bei Fälligkeit einmalig (an alle clients des users) ausgespielt * Wiederkehrende Erinnerungen werden ausgespielt und anschließend an Hand des Intervals neu terminiert * Der Nutzer kann Erinnerungen löschen * Der Nutzer kann seine Erinnerungen anzeigen lassen * Der Service wird als Docker Container via docker-compose verwaltet und in das BHT-Repository integriert * Der Service nutzt das (noch zu schaffende) Content-Service-Framework * Wenn der Reminder-Service nach einem Ausfall wieder aktiv wird erinnert er nicht (einzeln) an alle Termine, die zwischenzeitig fällig waren * Der Serive ist in Hilfe-Texten des (Chat) BHT-Bots integriert

Tasks	* BOT-57 Rasa Anbindung „Erinnere“-Direktive * BOT-58 Service Endpoint speichert Reminder und Antwortet auf Probleme * BOT-59 Scheduler/Cronjob prüft und sendet regelmäßig fällige Erinnerungen * BOT-60 Dokumentation Service Usage * BOT-109 Deployment / Release * BOT-112 Hilfe-Texte in (Chat)BHT-Bot help-command / willkommensnachricht

BOT-82: Termin-Scraper, der automatisch Erinnerungen aus öffentlichen Quellen bezieht

Es gibt Termine, an die wollen alle Studierenden typischerweise erinnert werden, als auch Termine, von denen die Universität möchte, dass die Studierende sich daran erinnern. Typische Beispiele hierfür sind Rückmeldefristen und (Beuth-eigene) Feiertage.
Durch einen Webscraper sollen solche Termine aus (öffentlichen) Quellen automatisch bezogen und dann an alle Studierenden ausgespielt werden.
Auch wenn dieses Feature für die meisten Studierenden interesssant sein dürfte, muss es eine Möglichkeit zum Opt-Out geben. Ggf. ist sogar angezeigt, dass ein Opt-In erfolgt, was allerdings den Nutzen des Features, vor allem aus Universitätssicht, mindern würde.

Initiale Schätzung	1 Tag
Technologien	* Javascript * Docker * HTML-DOM
Abhängigkeiten	* BOT-55
Anforderungen	* Relevante Termine werden regelmäßig, automatisch bezogen und als Erinnerung gespeichert * User können “globale Erinnerungen” aktivieren oder deaktivieren * Das Feature ist resistent gegen Änderungen an den Domains oder deren Struktur → Fallen gescrapete Dienste länger aus wird dies reported * Das Feature wird als nicht-eigenständig in den Reminder Service integriert * Das Feature ist bzgl. Opt-in oder Opt-out in den Hilfe-Texten/Begrüßungsnachrichten des BHT-Bot dokumentiert
Tasks	* BOT-83 Prüfen ob Opt-In (nötig ist) oder Opt-Out (möglich ist) * BOT-84 Quellen mit relevanten Terminen zusammentragen - auf Scrapebarkeit achten * BOT-85 Scraping (mittels Scapring-Service) implementieren * BOT-86 Termine aus Scraping in Erinnerungs Datenbank überführen * BOT-87 Rasa Direktive Opt-In oder Opt-Out implementieren * BOT-88 Error-Reporting implementieren, bei Misslungenen Scraping (über gewisse Zeit hinweg) * BOT-114 Opt-In oder Opt-Out in Hilfe-Texten / Willkommensnachricht dokumentieren

BOT-89: Moodle iCal import als Erinnerungen

Moodle ist die zentrale Online-Lernplattform der Beuth Hochschule. Kurse, die in Moodle verwaltet werden erhalten in der Regel Abgabetermine für Aufgaben, die während des Semesters fällig werden. Oftmals stehen diese Abgabetermine bereits zu Beginn des Semesters in Moodle fest und können dort in einer Kalenderansicht betrachtet werden.
Moodle bietet außerdem eine Funktion zum Export der Eintragungen im eigenen Kalender: https://lms.beuth-hochschule.de/calendar/export.php
Der User kann hier einen Link erzeugen, über den eine iCal Datei bezogen werden kann. Diese Datei enthält die Semestertermine und kann entsprechend in Erinnerungen des Bots umgewandelt werden

Initiale Schätzung	1 Tag
Technologien	* Javascript * iCal * Rasa
Abhängigkeiten	* BOT-55
Anforderungen	* Der User kann dem Bot seinen Moodle-Kalender-Export-Link senden um einen Import auszulösen * Die iCal Datei hinter dem Export-Link wird in Erinnerungs-Einträge des Erinnerungs-Service umgewandelt * Erinnerungen an Abgabetermine erfolgen zu Beginn des Tages / zeitlich versetzt vor der Fälligkeit der Abgabe, nicht zum im Kalender angegebenen Zeitpunkt * Das Moodle-Import Feature wird nicht-eigenständig in den Reminder-Service integriert * Das Feature (und seine Nutzung) ist in der BHT-Bot Hilfe gelistet
Tasks	* BOT-90 Import Moodle Rasa Direktive * BOT-91 Moodle iCal Download via zentralem Scaper Service * BOT-92 iCal Parsing und Persistierung als (sinnvolle) Erinnerung * BOT-111 Feature Release * BOT-113 Hilfe-Texte in (Chat)BHT-Bot help-command / willkommensnachricht

BOT-75: Begrüßungsnachricht

Neue Benutzer des BeuthBots sollen mit einer Begrüßungsnachricht empfangen werden. Diese Nachricht soll die Features des BeuthBots vorstellen und einen Shortcut nennen, welchen die Benutzer verwenden können, wenn sie Hilfe benötigen. Mit dem Shortcut listet der BeuthBot nochmals all seine Features auf.

Initiale Schätzung	1 Tag
Technologien	* JavaScript
Abhängigkeiten	Keine
Anforderungen	* Die Begrüßungsnachricht erscheint nur für neue Benutzer * Das System sollte einen Shortcut zur Wiedervorstellung der Features bereitstellen, falls Benutzer Hilfe brauchen * Das System muss in der Lage sein, auf die Hilfeanfrage. des Benutzers mit Hilfe des Shortcuts innerhalb von 1,5 Sekunden zu antworten * Das System sollte gut dokumentiert sein * Das System sollte leicht zu verstehen sein
Tasks	* BOT-93 Client * BOT-94 Server * BOT-132 Server fragt alle anderen Server was sie können/ machen und gibt das dann aus

BOT-11: Universeller Scraper & Download

Der Beuthbot soll einen „universellen“ Web-Scraper beinhalten, der als Grundlage für künftige Features dienen soll, die für konkrete Scraping-Funktionalitäten vorgesehen sind. Aufgrund der hohen Diversität an Datenstrukturen unterschiedlicher Webseiten, soll dieser möglichst abstrakte Funktionalitäten zur Extrahierung von Datensätzen bieten.

Initiale Schätzung	1 Tag
Technologien	Javascript
Abhängigkeiten	* BOT-43: Erstellung eines Common-Frameworks für (Content-)Services
Anforderungen	*Import von HTML- und XML-Dateien *Daten lassen sich im JSON-Format exportieren *Datensätze sind per HTML-Tags und CSS-Selektoren extrahierbar *Dateien einer Webseite lassen sich downloaden
Tasks	* BOT-26 Recherche nach geeignetster Methode (HTML-JSON)

BOT-15: Personalliste der Beuth-Hochschule im BeuthBot abrufbar machen

Dem Bot wird eines neues Feature hinzugefügt. Dieses Feature soll dem Benutzer des BeuthBots ein Abfrage von Informationen über das Personal der Beuth Hochschule(BHT) ermöglichen
Ablauf:
Der Benutzer teilt dem Bot über einen Befehl mit (“Wer ist Max Mustermann?”, “Welche Person hat die E-Mail mail@mail.com?”, “Welche Personen sitzen in Raum B001?”) , dass er Informationen über eine oder mehrere Personen erhalten möchte. Der Bot prüft dann die mitgegebenen Informationen und gibt dann aus:
* Wenn er passende Daten in der Datenbank findet:
- Auflistung der angefragen Daten
* Wenn er keine passenden Daten finden kann:
- Meldung, das die Suche erfolglos war
Weitere Informationen:
Die Informationen über das Personal werden in einer Tabelle in der Datenbank gespeichert und von dort abgerufen. Diese Informationen können jederzeit aktuallisiert werden.
Spätere Erweiterungsmöglichkeiten:
Zunächst werden nur Entwickler Zugriff auf das Bearbeiten der Daten besitzen, für eine spätere Ausbaustufe ist aber eine Verwaltung der Daten mittels dafür berechtigter User vorstellbar.
Die Einbindung der Personaldaten kann über einen Scrapper auf der Seite der Personalliste auch persepektivisch komplett automatisiert werden.
Die Daten in der Datenbank können mit neuen Informationen erweitert werden. (Persönliches, Sprechzeiten, etc.)

Initiale Schätzung	3 Tage
Technologien	* Javascript * mongodb
Abhängigkeiten	keine
Anforderungen	* Auslesen der Personalliste der BHT * Einbau einer Speichermöglichkeit in der Datenbank * Speichern der ausgelesenen Informationen in der Datenbank * Schnittstelle zur Bearbeitung der Daten erstellen * Anlegen eines neuen Service „Personalliste“ im Bot * Service zur Verwendung für die Benutzer abrufbar machen
Tasks	* BOT-17 - initiales Auslesen der Personalliste * BOT-18 - Abrufen der Information aus der Personalliste * BOT-19 - Erkennung der Benutzeranfrage zur Personensuche * BOT-96 - Bearbeiten der Daten der Personalliste * BOT-125 - Erstellen einer neuen Tabelle „Personalliste“ in der Datenbank

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.