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
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-37: Discord Integration des BHT-Bot

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
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
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-49: User-Messenger-Servie: 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
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-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
Technologien	* Javascript * Docker * Rasa * 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-XXX: EPIC_TITLE

EPIC_DESCRIPTION

Initiale Schätzung	TIME
Technologien	* <Programmiersprache 1> * <Containerisierung 1> * <Bot Servie 1>
Abhängigkeiten	* <Ticket ID1> * <Ticket ID2>
Anforderungen	* <Anforderung 1> * <Anforderung 2>
Tasks	* <Task1> * <Task2>

BOT-XXX: EPIC_TITLE

EPIC_DESCRIPTION

Initiale Schätzung TIME

Technologien
* <Programmiersprache 1>
* <Containerisierung 1>
* <Bot Servie 1>

Abhängigkeiten
* <Ticket ID1>
* <Ticket ID2>

Anforderungen
* <Anforderung 1>
* <Anforderung 2>

Tasks
* <Task1>
* <Task2>

BOT-XXX: EPIC_TITLE

EPIC_DESCRIPTION

Initiale Schätzung TIME

Technologien
* <Programmiersprache 1>
* <Containerisierung 1>
* <Bot Servie 1>

Abhängigkeiten
* <Ticket ID1>
* <Ticket ID2>

Anforderungen
* <Anforderung 1>
* <Anforderung 2>

Tasks
* <Task1>
* <Task2>

BOT-XXX: EPIC_TITLE

EPIC_DESCRIPTION

Initiale Schätzung TIME

Technologien
* <Programmiersprache 1>
* <Containerisierung 1>
* <Bot Servie 1>

Abhängigkeiten
* <Ticket ID1>
* <Ticket ID2>

Anforderungen
* <Anforderung 1>
* <Anforderung 2>

Tasks
* <Task1>
* <Task2>