

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

Aktueller Stand:

Der BeuthBot

Geplanter Umbau:

Initiale Schätzung 3

Technologien	* Javascript * mongodb
--------------	---------------------------

Abhängigkeiten	keine
----------------	-------

Anforderungen	* <Anforderung 1> * <Anforderung 2>
---------------	--

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.

Zuständigkeit	Robert Halwaß
---------------	---------------

Initiale Schätzung	3 Tage
--------------------	--------

Programmiersprachen	* Python
---------------------	----------

Frameworks/Libraries	* Rasa
----------------------	--------

Services	* NLU
----------	-------

Abhängigkeiten	keine
----------------	-------

Anforderungen	* Kompatibilität mit bestehenden NLU-Trainingsdaten erhalten * Mögliche JSON- und MarkDown-Dateien in YAML-Dateien umwandeln
---------------	---

Tasks	* BOT-124: docker-compose.yml anpassen * BOT-61: Chatito Kompatibilität testen * 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 Kommunikationskanal 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

Discord ist eine weit verbreitete Kommunikationsplattform, 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 diese 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 Verbreitungsmöglichkeit 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	<ul style="list-style-type: none"> * 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 Telegram 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	<ul style="list-style-type: none"> * 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, repetitiver, 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	<ul style="list-style-type: none"> * Javascript * Typescript * Dockerfile
Abhängigkeiten	* BOT-37

Anforderungen	<ul style="list-style-type: none">* 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	<ul style="list-style-type: none">* 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 enforecen

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

Technologien

- * Javascript
- * Websockets
- * Docker

Abhängigkeiten * BOT-30

Anforderungen	<ul style="list-style-type: none"> * 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 integriert. 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	<ul style="list-style-type: none"> * 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	<ul style="list-style-type: none"> * Javascript * Docker * Rasa * MongoDB * Cronjob
--------------	--

Abhängigkeiten	<ul style="list-style-type: none"> * BOT-43 * BOT-30 * BOT-12 * BOT-49
----------------	--

Anforderungen	<ul style="list-style-type: none">* Erfolgreiche „erinnere“-Anfragen werden vom Dienst durch Bestätigung der erkannten und persistierten Daten beantwortet oder* 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 (einzelne) an alle Termine, die zwischenzeitig fällig waren* Der Service ist in Hilfe-Texten des (Chat) BHT-Bots integriert
Tasks	<ul style="list-style-type: none">* 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 interessant 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
Technologien	<ul style="list-style-type: none">* Javascript* Docker* HTML-DOM
Abhängigkeiten	* BOT-55

Anforderungen	<ul style="list-style-type: none"> * 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	<ul style="list-style-type: none"> * 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
Technologien	<ul style="list-style-type: none"> * Javascript * iCal * Rasa
Abhängigkeiten	<ul style="list-style-type: none"> * BOT-55
Anforderungen	<ul style="list-style-type: none"> * 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-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. Hierzu sollen 3 bekannte STT-Frameworks (Kaldi, Mozilla Voice STT und Wav2Letter) getestet und verglichen 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.

Zuständigkeit	Robert Halwaß
Initiale Schätzung	3 Tage
Programmiersprachen	<ul style="list-style-type: none">* Python (Mozilla Voice STT)* C++ (Kaldi, WAV2Letter)
Frameworks/Libraries	<ul style="list-style-type: none">* Kaldi* Mozilla Voice STT* WAV2Letter
Services	<ul style="list-style-type: none">* Speech
Abhängigkeiten	<ul style="list-style-type: none">* BOT-43: Erstellung eines Common-Frameworks für (Content-)Services
Anforderungen	<ul style="list-style-type: none">* Die Übersetzung soll mittels neuronaler Netzze 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-Adaption durchgeführt werden

Tasks**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
Technologien	Javascript

Abhängigkeiten	keine
Anforderungen	<ul style="list-style-type: none"> *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	<ul style="list-style-type: none"> * BOT-24 Recherche nach geeignetem Tool (TTS) * BOT-25 Eigene Implementierung (TTS) * BOT-98 Integration in Beuthbot

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
Technologien	* JavaScript
Abhängigkeiten	Keine
Anforderungen	<ul style="list-style-type: none"> * 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	<ul style="list-style-type: none"> * BOT-93 Client * BOT-94 Server * BOT-132 Server fragt alle anderen Server was sie können/machen und gibt das dann aus

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
Technologien	<ul style="list-style-type: none"> * TypeScript * JavaScript
Frameworks/ Libraries	<ul style="list-style-type: none"> * Angular * Bootstrap

Abhängigkeiten	<ul style="list-style-type: none">* BOT-10* BOT-30
Anforderungen	<ul style="list-style-type: none">* 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	<ul style="list-style-type: none">* 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-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
Technologien	Javascript
Abhängigkeiten	keine
Anforderungen	<ul style="list-style-type: none">* 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	<ul style="list-style-type: none">* 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 angefragten 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 aktualisiert 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 perspektivisch komplett automatisiert werden.

Die Daten in der Datenbank können mit neuen Informationen erweitert werden. (Persönliches, Sprechzeiten, etc.)

Initiale Schätzung	3
Technologien	* Javascript
Abhängigkeiten	keine
Anforderungen	<ul style="list-style-type: none"> * <Anforderung 1> * <Anforderung 2>
Tasks	<ul style="list-style-type: none"> * 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.