Zwischenbericht SS2020

BeuthBot Project Group

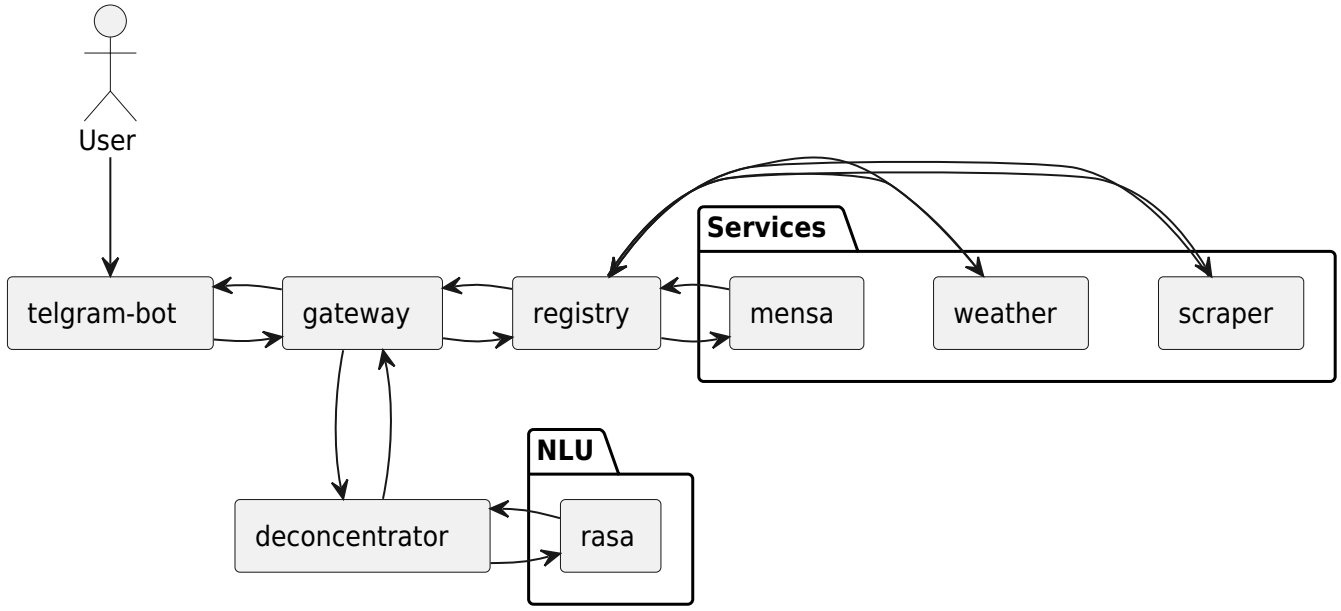
- Tobias Belkner
- Lukas Danckwerth
- Jan Fromme
- Denny Schumann

Inhaltsangabe

Was haben wir vorgefunden?

Es wurden verschieden Github Projekte wie, Telegram Bot, Gateway, Deconcentrator, Rasa, Registry, Weather, Scraper(Öffnungszeiten) und Mensa vorgefunden. Jedes dieser Projekte besaß seine eigene docker-compose.yml file. Die in jedem Projekt vorgefundenen .env Dateien waren wie zu erwarten leer, leider wurde in der Dokumentation nicht beschrieben wie diese zu befüllen sind. Zudem existierte ein Token, welcher für den Telegram Bot benötigt wurde. Ebenso konnte der Bot nicht vollständig ausgeführt werden, da der Service Deconcentrator fehlerhaft war. Auch konnte die virtuelle Maschine, welche zur verfügung gestellt wurde nicht standart gemäß ausgeführt werden. Leider wies die Dokumentation einige Lücken, im bezug auf das starten des Bots auf, wodurch es zu vielen Selbstrechergen und verzögerungen kam. Auch fehlte die Zugangsberechtigung des Servers, auf dem der Bot lief, da es ein “privater” Server eines Studenten gewesen war.

Initial Project State



Was haben wir gemacht?

Es wurde anfangs versucht alle Docker-Container der einzelnen Services des Bots lokal zu starten. Dabei musste erst einmal herausgefunden werden welche URLs bzw. welcher Token zu benutzen war. Da wir keinen Token in der bereits vorhandenen Dokumentation finden konnten bzw. es keinen hinterlegten gab, wurden übergangsweise mehrere Test-Tokens mit BotFather erstellt. Ebenso wurde versucht die Container sowohl auf Windows, also auch auf Linux zu starten. Daraufhin wurde versucht das Image des Beuth Bots auf einer virtuellen Maschine zu starten. Um dies ausführen zu können musste Anfangs erst einmal die Datei mit lz4 dekomprimiert werden. Dafür musste erst einmal das Github lz4(https://github.com/lz4/lz4) gedownloadet und ausgeführt werden, welches einige Zeit in Anspruch nahm. Danach wurde die kvm Datei mit qemu zu einer vdi Datei konvertiert, damit VirtuelBox diese Datei akzeptiert. Nach dem erfolgreichen starten der virtuellen Maschine gab es das Problem, dass das Passwort nicht richtig war, welches uns gegeben wurde. Letztendlich musste dies mit folgenden Codezeilen umgangen werden:

```
mount -o remount,rw /
passwd
# passwort eurer wahl eingeben
# oder, müsste auch gehen:
passwd --delete
mount -o remount,ro /
```

Nun wurde der Beuth Bot gestartet, konnte allerdings nicht in Betrieb genommen werden, da dieser keine Netzwerkverbindung nach draußen hatte. Daraufhin wurde uns ein Server auf der Beuth zur verfügung gestellt, auf welchem der Beuth Bot letztendlich laufen sollte. Nachdem dieser dort versucht wurde eingerichtet zu werden, fehlte es dem Server an zugewiesenem Speicher, was erst behoben werden musste um den Bot vollständig installieren und in Betrieb nehmen zu können. Es konnten nach der Behebung des fehlenden Speichers letztendlich fast alle Docker-Container erfolgreich gestartet werden, bis auf den Service Deconcentrator, welcher sich erst nicht richtig starten ließ und als er lief nicht funktionierte. Nach mehreren Wochen der Versuche diesen Service zum laufen zu bekommen entschieden wir uns in Absprache mit Herrn Ziemer diesen zu verwerfen und einen neuen Deconcentrator zu schreiben.

Resultierende Aufgaben

- Dokumentation komplettieren / umstruktorieren
- Den alten deconcentrator „rausschmeissen“ und neuen (deconcentrator-js) schreiben
- Übergeordnetes Git Projekt erstellen mit Paketen als Submodule

Wo stehen wir gerade?

Aktuell Ist der Beuth Bot vollständig in Betrieb genommen, dank des neu geschriebenen Deconcentrators. Ebenso liegt dieser nun, auf einem für uns zugreifbaren Server, welcher dort erfolgreich in Betrieb genommen wurde. Auch wurde er nun ermöglicht den gesamten Bot, welcher 8 Services beinhaltet mit zwei docker-compose files zu starten. Das Projekt wurde nun in 2 Git Submodule unterteilt, welche in Zukunft leichter bearbeitet werden können. Ebenfalls wurden sowohl in diesem Semester zu bearbeitende, also auch zukünftige Projektideen erschlossen. Die Arbeitsteilung der einzelnen Gruppenmitglieder wurde durchgeführt und ein Zwischenbericht wurde angelegt.

Wo werden wir tun?

Zusammengefasst, werden wir den Scraper Microservice fertigstellen und anpassen, eine Persistenz hinzufügen die es dem System erlaubt Präferenzen vom User zu speichern, den Wetter Microservice anpassen, einen Cache hinzufügen und zwei neue Microservices namens Schedule (Stundenplan) und Finals (Prüfungen) hinzufügen.

Auf diese Punkte wird im weiteren Verlauf dieses Dokuments eingegangen.

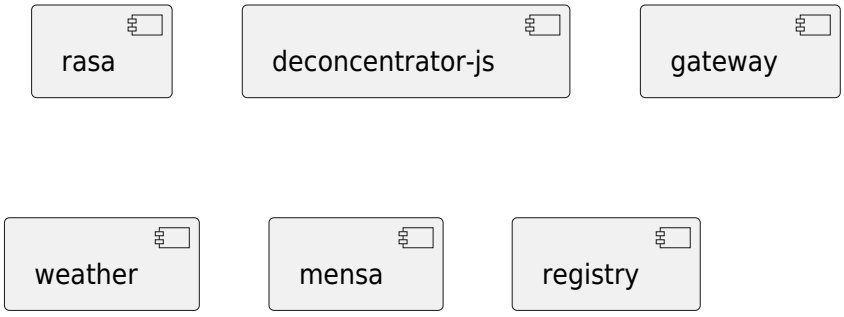
BeuthBot Project (One Git Repository)

https://github.com/beuthbot/beuthbot

Motivation

Fulfilling the following requirements (Link, Section: 'Requirement Analysis BeuthBot') `/NF300/`, `/NF301/`, `/NF302/`, `/NF400/` it's obvious to have the BeuthBot splitted up into many (small) repositories. Especially when having the requirements to have the project as modular as possible and to have the project easy extendable.

BeuthBot's structure before this repository:



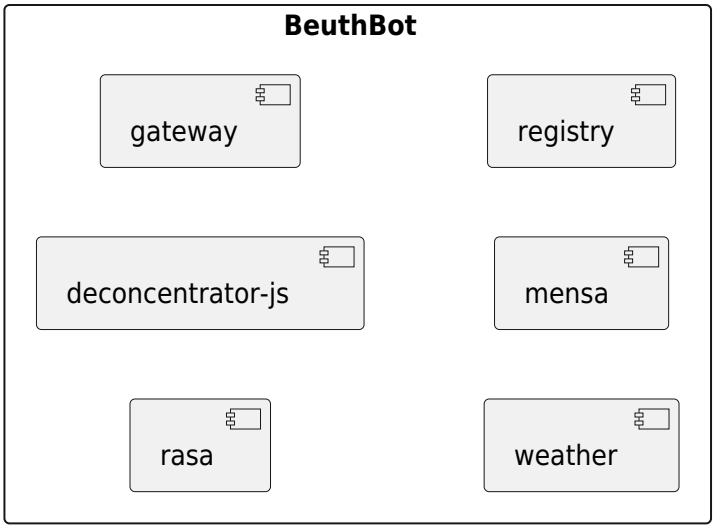
But when deploying on a (productive) machine we faced the problem cloning at least a half-dozen repositories, editing the `.env` files of these projects and invoking each `docker-compose.yml` individually. This means at least typing in the following commands **six** times:

```
# clone repository
$ git clone https://github.com/beuthbot/$PROJECT_NAME.git

# change into directory
$ cd $PROJECT_NAME

# edit environment file
$ cp .env.sample .env && vim .env

# start project
$ docker-compose up -d
```

Not even for us this is / was a huge workload before starting development and / or when deploying the project. Facing this problem we started writing bash scripts updating and editing these projects. But in the end that didn't felt right so we decided having a master project (this repository) containing and combing the packages of the BeuthBot and making it possible running the whole system with one `.env` and one `docker-compose.yml` file.

> We **did not** remove the `docker-compose.yml` files of the components in order to have the option to start all services seperatly.

BeuthBot's structure with this repository:



With this repository it became way easier to manage, develope and deploy the BeuthBot. The following collection of commands which can be used to **fully** deploy the BeuthBot demonstrates that. Note the `–recursive` argument for the git `clone` command which make git fetching the submodules, too. Have a look at the section [Working with Submodules](#Working-with-Submodules) for further information about working with submodules.

```
# clone project
$ git clone --recursive https://github.com/beuthbot/beuthbot.git

# change into directory
$ cd beuthbot

# edit environment file
$ cp .env.sample .env && vim .env

# start BeuthBot
$ docker-compose up -d
```

> Having this project organized with submodules makes it also easier to have and organize multiple **distributions** of this project. It further allows us having a global state / version of the BeuthBot.

Default Ports of Services

Service	External Port	Internal Port
gateway	3000	3000
deconcentrator-js	8338	8338
rasa	5005	5005
registry	9922	3000
mensa	9950	8000
weather	9951	7000

Packages / Submodules

Packagename	About	Language
gateway	Receives messages from bot clients via a API.	JS
deconcentrator-js	Asks multiple NLU processors for the interpretation of a given message	JS
rasa	NLU	Python
registry	The registry of services. It knows all existing services and handles the requests against these services.	JS
mensa	The mensa service of the BeuthBot. It knows whether the Mensa is open or closed.	JS
weather	The weather service.	JS

Other Packages

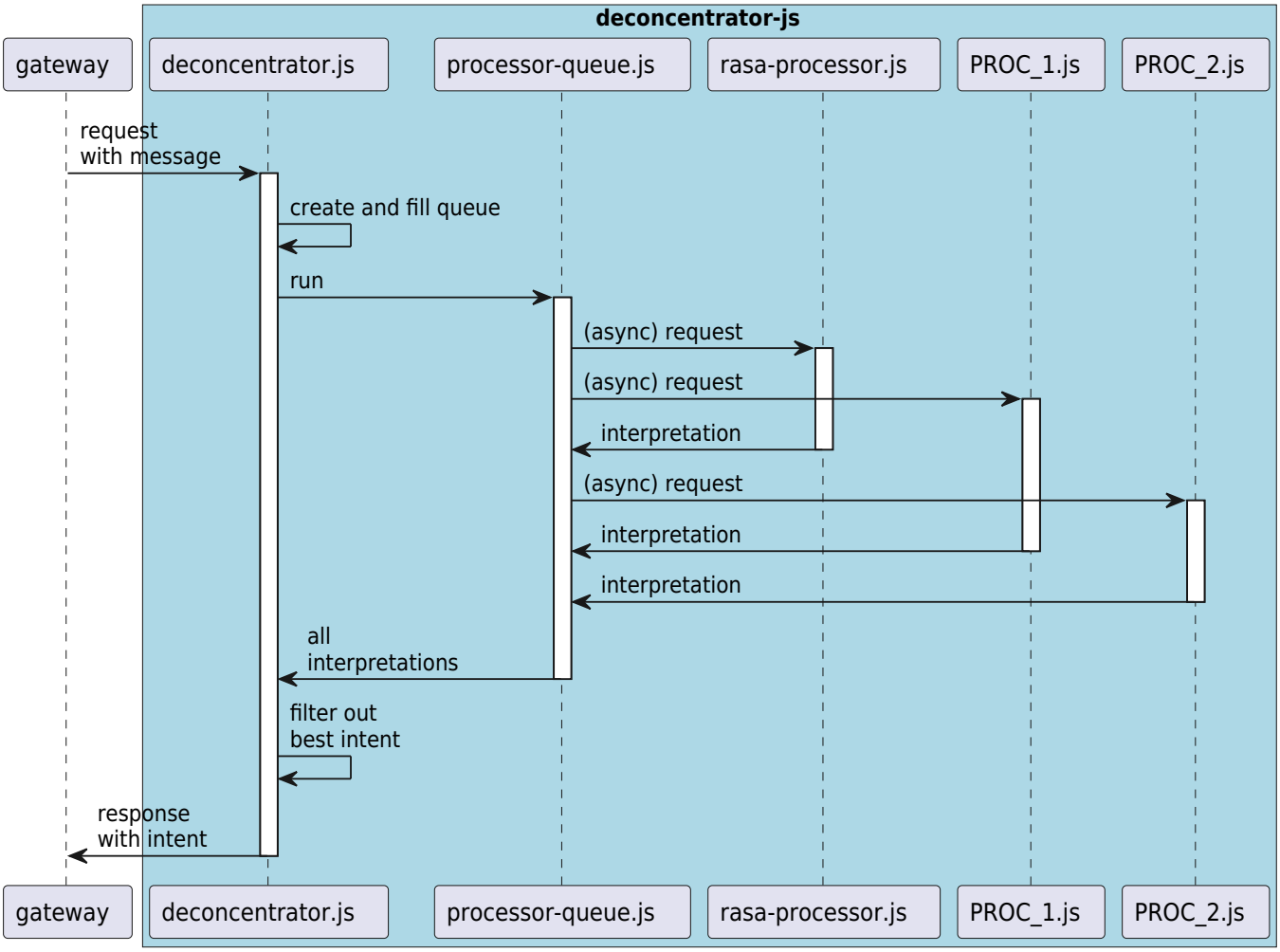
Packagename	About	Language
.documentation	Contains mostly text, image and markdown files with information and documentation about this repository.	-
scripts	Contains scripts to automate tasks.	BASH

deconcentrator-js

https://github.com/beuthbot/deconcentrator-js

The deconcentrator uses different NLU processors to compare their results and tries to choose an best fitting answer. The NLU processors like RASA must know their domain on their own. The deconcentrator simply compares the confidence score of the intents given from the processors and returns the intent with the highest score.

Functionality



processor-queue.js

For every incoming message the deconcentrator creates a new `ProcessorQueue` (defined in `processor-queue.js`) and adds all available processors to it. When calling the `.interpretate(message)` function of the queue it starts requesting the processors for an interpretation. The number of asynchronous requests can be set with the `numOfSynchronProcessors` property of the queue.

processor.js

Defines the interface of a NLU processor.

API

The following lists the resources that can be requested with the deconcentrator API.

Request life sign.

```
GET   http://localhost:8338
```

Answer:

```
Hello from BeuthBot Deconcentrator: 0.1.1
```

Request interpretation of message.

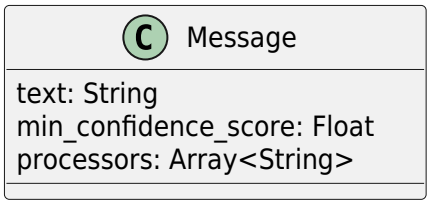
```
POST  http://localhost:8338/messages
```

Request Schema

```
{
  "text": "Wie wird das Wetter morgen?",
  "min_confidence_score": 0.8,
  "processors": ["rasa"]
}
```

Whereas the specification of the min_confidence_score and theprocessors is optional. If not minimum confidence score is given a default one is used (by now this is 0.8). For now there is only the usage of RASA implemented so there is no effect of specifying the processors property.

Model of an incoming message.



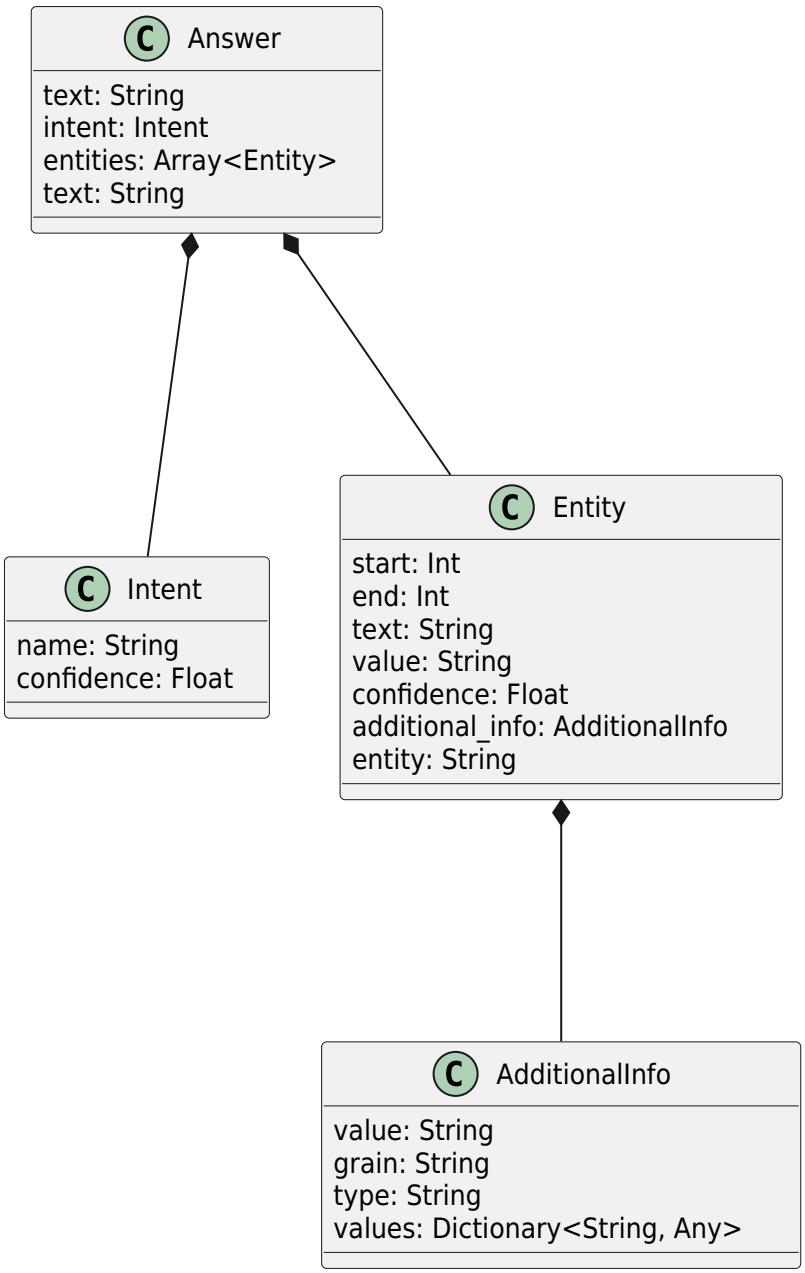
Response Schema

The response for a successfully processed request to the deconcentrator contains the following information.

```
{
  "intent": {
    "name": "wetter",
    "confidence": 0.9518181086
  },
  "entities": [
    {
      "start": 20,
      "end": 26,
      "text": "morgen",
      "value": "2020-01-20T00:00:00.000+01:00",
      "confidence": 1.0,
      "additional_info": {
          "values": [
              {
                  "value": "2020-01-20T00:00:00.000+01:00",
```

```
                "grain": "day",
                "type": "value"
            }
        ],
        "value": "2020-01-20T00:00:00.000+01:00",
        "grain": "day",
        "type": "value"
      },
      "entity": "time"
    }
  ],
  "text": "Wie wird das Wetter morgen?"
}
```

Model of an answer.



The response for a unsuccessfully processed request to the deconcentrator or when an error occures contains the following information.

```
{
  "error": "The given message can't be interpretated.",
  "text": "Wie wird das Wetter morgen?"
}
```

Requirements Analysis deconcentrator.js

- /DCF100/ The deconcentrator responds to incoming POST requests by delegating the message to a collection of NLU processor which try to interpretate the given message
- /DCF101/ The deconcentrator accepts incoming messages as defined via the Request Schema
- /DCF102/ The deconcentrator sends answers as defined via the Response Schema
- /DCF103/ The deconcentrator answers with proper messages for occuring errors
- /DCF104/ New NLU processors muss be easy to integrate
- /DCF105/ The deconcentrator has a default value for the minimum confidence score
- /DCF106/ The deconcentrator has a default value for the list of processors
- /DCF107/ The minimum confidence score can be set globally within the Dockerfile
- /DCF108/ The list of processors to be used can be set globally within the Dockerfile

Deploying on Virtual Machine

Install BeuthBot on a virtual machine:

```
# clone project
$ git clone https://github.com/beuthbot/beuthbot.git

# change into directory
$ cd beuthbot

# initialize submodules
$ git submodule init

# clone all submodules
$ git submodule update

# edit environment file
$ vim .env

# start BeuthBot
$ docker-compose up -d
```

Contents of .env file

Following lists the contents of the .env file of the BeuthBot project. Note that the value for WEATHER_API_KEY has been removed for security reasons.

```
# deconcentrator
RASA_ENDPOINT=http://rasa:5005/model/parse

# gateway
DECONCENTRATOR_ENDPOINT=http://deconcentrator:8338/message
REGISTRY_ENDPOINT=http://registry:3000/get-response

# registry
MENSA_ENDPOINT=http://mensa:8000/meals
WETTER_ENDPOINT=http://weather:7000/weather

WEATHER_API_KEY=     # key removed
```

Contents of docker-compose.yml file

Following lists the contents of the docker-compose.yml file of the BeuthBot project.

```
version: '3.7'

services:

  # === ----------------------------
  gateway:
    build: gateway
    restart: unless-stopped
    links:
      - deconcentrator
      - registry
    ports:
      - 3000:3000
    environment:
      - DECONCENTRATOR_ENDPOINT
      - REGISTRY_ENDPOINT

  # === ----------------------------
  deconcentrator:
    build: deconcentrator-js
    restart: unless-stopped
    links:
      - rasa
    ports:
      - 8338:8338
    environment:
      - RASA_ENDPOINT

  # === ----------------------------
  registry:
    build: registry
    restart: unless-stopped
```

```
    links:
      - mensa
      - weather
    ports:
      - 9922:3000
    environment:
      - MENSA_ENDPOINT
      - WETTER_ENDPOINT

  # === ----------------------------
  rasa:
    image: rasa/rasa:1.6.0-spacy-de
    restart: unless-stopped
    ports:
      - 5005:5005
    volumes:
      - ./rasa/docker/rasa-app-data:/app
    command:
      - run
      - --enable-api
      - --cors
      - "*"
  duckling:
    image: rasa/duckling:0.1.6.2
    restart: unless-stopped
    ports:
      - 8000:8000

  # === ----------------------------
  mensa:
    build: mensa_microservice
    restart: unless-stopped
    ports:
      - 9950:8000

  # === ----------------------------
  weather:
    build: weather_microservice
    restart: unless-stopped
    ports:
      - 9951:7000
    environment:
      - WEATHER_API_KEY
```

Install Telegram Bot on a virtual machine:

```
# clone with HTTPS
$ git clone https://github.com/beuthbot/telegram-bot.git

# change into directory
```

```
$ cd telegram-bot

# edit environment file
$ vim .env

# start Telegram Bot
$ docker-compose up -d
```

Contents of .env file

Following lists the contents of the .env file of the telegram-bot project. Note that the value for TELEGRAM_TOKEN has been removed for security reasons.

```
GATEWAY_ENDPOINT=http://172.17.0.1:3000
TELEGRAM_TOKEN=        # removed
```

Contents of docker-compose.yml file

Following lists the contents of the docker-compose.yml file of the BeuthBot project.

```
version: '3.7'
services:
  telegram-bot:
    build: .
    restart: unless-stopped
    environment:
      - GATEWAY_ENDPOINT
      - TELEGRAM_TOKEN
```

Current output of `docker ps` on virtual machine:

```
CONTAINER ID        IMAGE                          COMMAND
CREATED             STATUS              PORTS
NAMES
881848bfaa3c        beuthbot_gateway               "docker-entrypoint.s…"   55
minutes ago      Up 54 minutes       0.0.0.0:3000->3000/tcp
beuthbot_gateway_1
c5ef1f78da2f        beuthbot_deconcentrator        "docker-entrypoint.s…"   55
minutes ago      Up 55 minutes       0.0.0.0:8338->8338/tcp
beuthbot_deconcentrator_1
992d5ed6d94b        beuthbot_registry              "docker-entrypoint.s…"   55
minutes ago      Up 55 minutes       0.0.0.0:9922->3000/tcp
beuthbot_registry_1
62e0bf7e2c8d        rasa/rasa:1.6.0-spacy-de       "rasa run --enable-a…"   55
minutes ago      Up 55 minutes       0.0.0.0:5005->5005/tcp
beuthbot_rasa_1
b33f342d24fd        beuthbot_weather               "docker-entrypoint.s…"   55
```

```
minutes ago      Up 55 minutes       8000/tcp, 0.0.0.0:9951->7000/tcp
beuthbot_weather_1
8beba8a324e8        beuthbot_mensa              "docker-entrypoint.s…"   55
minutes ago      Up 55 minutes       0.0.0.0:9950->8000/tcp
beuthbot_mensa_1
75b7b5653577        telegram-bot_telegram-bot   "docker-entrypoint.s…"   2
hours ago         Up 2 hours
telegram-bot_telegram-bot_1
7d9e26988632        rasa/duckling:0.1.6.2       "duckling-example-ex…"   24
hours ago        Up 24 hours         0.0.0.0:8000->8000/tcp
beuthbot_duckling_1
```

Both projects contains a update.sh file which can be used to fast update the projects.

Requirements

Funktionale Anforderungen:

/F100/ Das System muss den User fragen, ob er möchte dass seine Präferenzen gespeichert werden.

/F101/ Das System muss die User-Präferenzen in einer Datenbank speichern können.

/F102/ Das System muss die Responses der Microservices in einem Cache zwischenspeichern.

/F103/ Das System muss den User an Termine und Prüfungen erinnern.

/F200/ Das System muss die Prüfungen von der Beuth Prüfungs-Website scrapen.

/F201/ Das System muss die PDF die das System vom User bekommt verarbeiten können.

/F202/ Das System muss dem User die Möglichkeit bieten einen Stundenplan manuell anzulegen.

/F203/ Das System muss dem User die Möglichkeit bieten einen Stundenplan per PDF anzulegen.

Nicht-Funktionale Anforderungen:

/NF100/ Das System sollte Nachrichten innerhalb von 3 Sekunden beantworten.

/NF101/ Das System sollte so modular wie möglich aufgebaut sein.

/NF102/ Das System sollte eine Downtime von maximal 1% haben.

/NF103/ Die Datenbank sollte eine Downtime von maximal 1% haben.

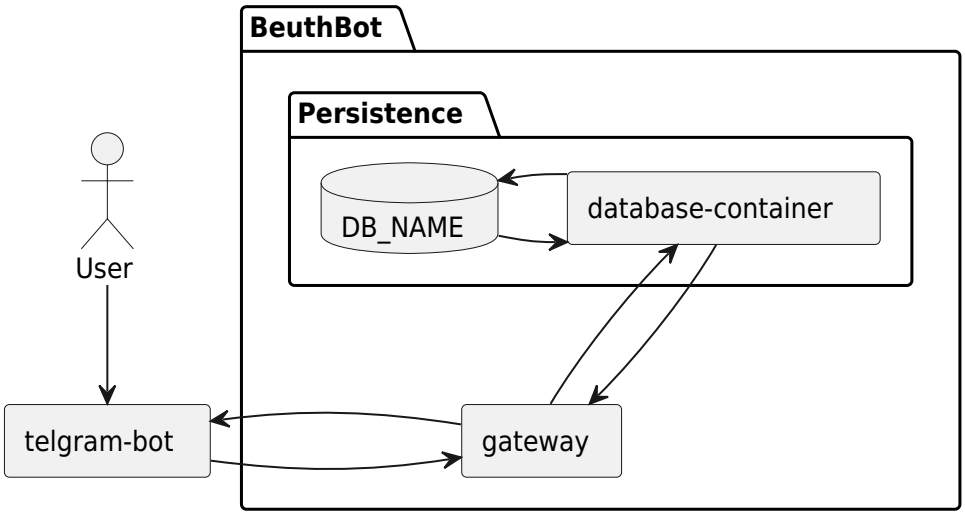
/NF200/ Das System sollte DSGVO konform sein.

/NF201/ Das System sollte standard Sicherheitsvorkehrungen besitzen.

Persistence

Damit der Benutzer sich selbst nicht ständig wiederholen muss, wird ihm die Möglichkeit geboten,

seine Vorlieben zu speichern. Als Datenbank haben wir uns für die MongoDB entschieden.

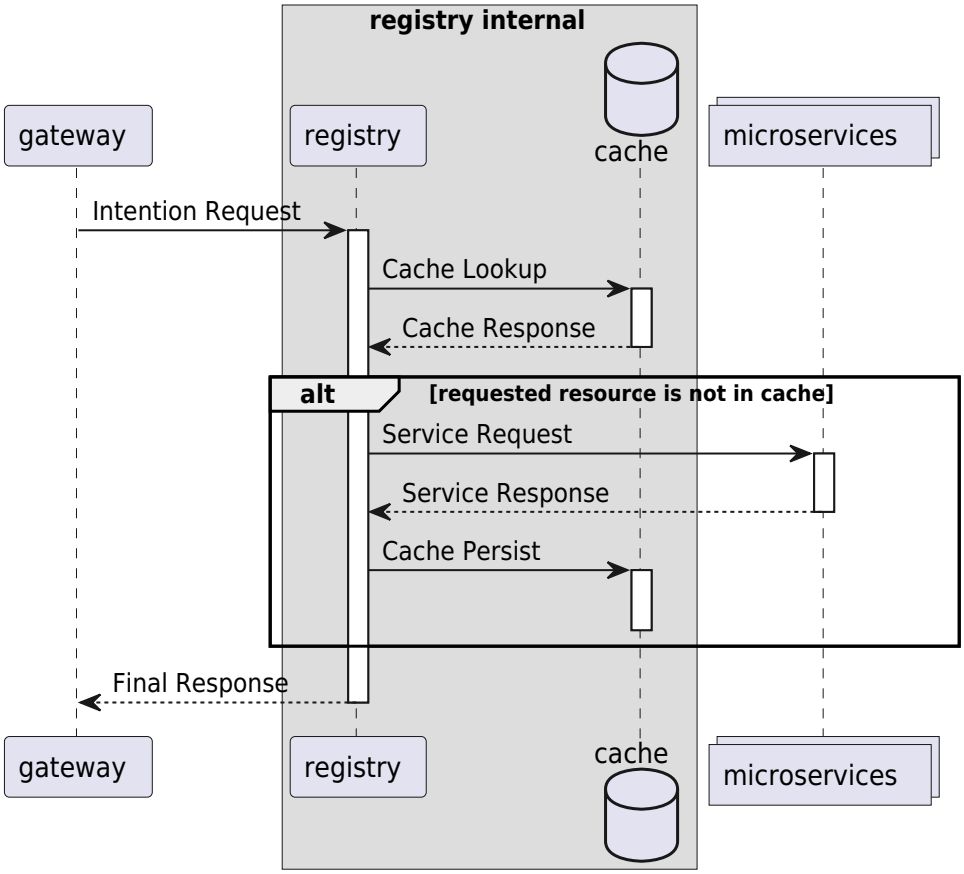


Cache

Die Motivation hinter dem Cache besteht darin, dass externe API's, wie z.B. die WetterAPI, welche in dem Beuthbot Projekt benutzt wird, nur begrenzt Zugriffe zulassen. Daher müssen die Anfragen durch ein Cache begrenzt werden, um wiederholte Anfragen an die API zu limitieren und im Cache abzulegen. Ein positiver Effekt des Caches wird auch sein, dass Zeit gespart werden kann, weil unnötige Anfragen an die Services erspart bleiben.

Damit die Anfrage gecached werden kann, muss diese zuvor von dem Deconcentrator interpretiert werden, um die korrekte Intention hinter dieser Anfrage zu verstehen, da die Anfrage des Benutzers jedesmal anders formuliert sein könnte. Diese Intention gibt der Registry an, an welchen Microservice diese Anfrage geschickt werden soll und erhält auch die Antwort des jeweiligen Microservices. Um doppelte Anfragen in zu kurzer Zeit zu verhindern, ist die optimale Stelle für den Cache bei der Registry.

subject to change

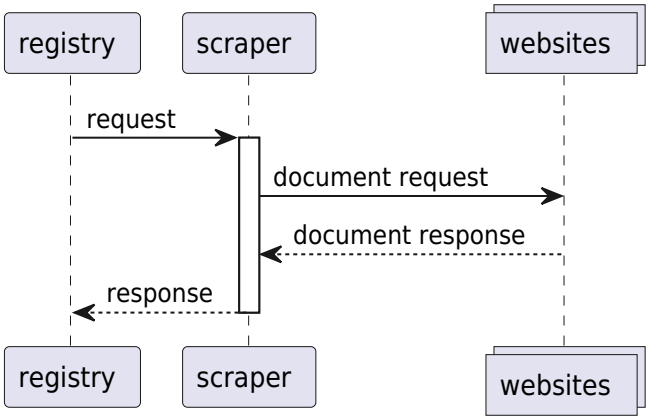


Additional Services

Ebenfalls wurden weitere Mirko Service Ideen für denn zukünftigen Beuth Bot entwickelt welche da wären:

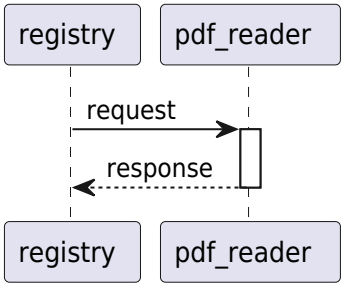
Microservice: Scraper

Dieser Microservice soll ausschließlich von anderen Services, wie beispielsweise Schedule benutzt werden. Er soll Informationen von Webseiten, wie von der Beuth Website scapen und in eine Datenbank speichern. Auf diese Daten können letztendlich andere Microservices zugreifen und diese für ihre Dienste benutzen.



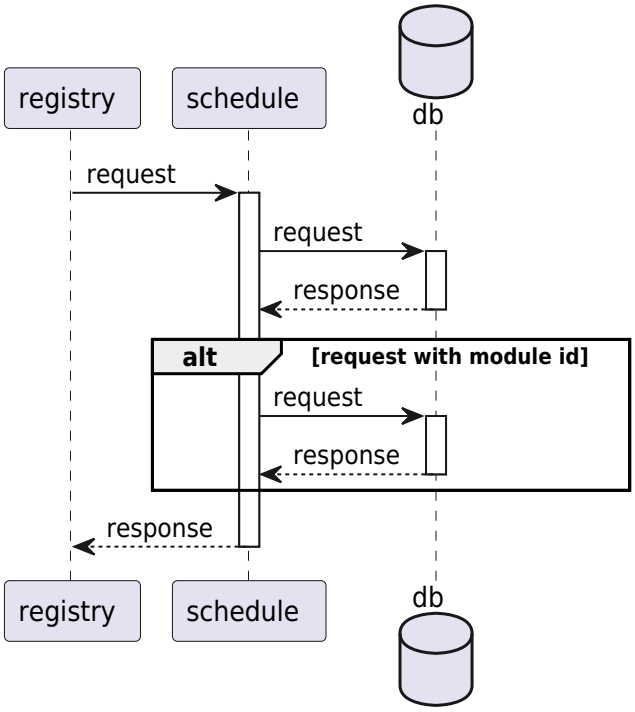
Microservice: PDF Reader

Dieser Service erwertet eine spezielle PDF des jeweiligen Stundenplans des Studenten und gibt den Inhalt der PDF in einem JSON zurück. Der Service soll später von den Microservices Schedule und finals benutz werden, welche die Daten weiterverarbeiten.



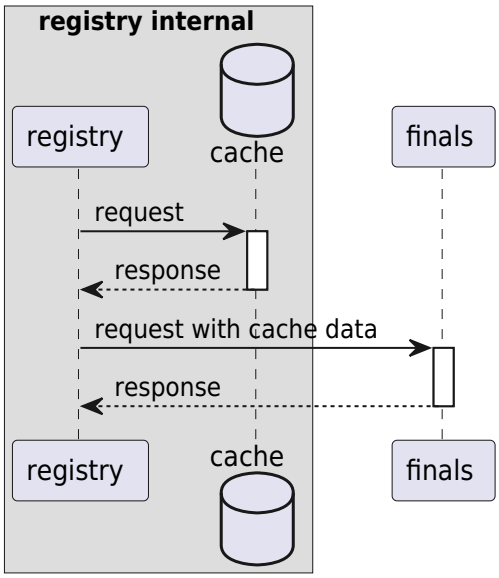
Microservice: Schedule

Dieser soll dem User die Funktionalität zur Verfügung stellen seinen eigenen Stundenplan erstellen und diesen auch ändern zu können. Ebenfalls soll der Service den User Notifications senden können, wie in der folgenden Beispiel Notification: „In 3 Wochen ist in dem Fach xy folgende Abgabe: Abgabe 3“. Dabei werden mehrere Möglichkeiten der Eingabe der Module in Betracht gezogen, wie Grundlegend das ganz normale manuelle einfügen per Text(Chat Eingabe), aber auch das Hinzufügen von Modulen über die Module ID. Für das hinzufügen eines Modules über die Modul ID wurde überlegt in eine Datenbank alle sich in diesem Semester existierenden Fächer abzuspeichern und aus diesen Einträgen die Daten der Modul ID abzugleichen. Auch wurde überlegt die PDF, welche sich jeder Student frei von der Beuth herunterladen kann abzuschicken und den Stundenplan automatisch anhand der Daten auf der PDF zu generieren. Dabei wird darauf geachtet, dass die PDF nicht zwischengespeichert wird, sondern nur eingelesen, Übersetzt und aus den Zwischenspeicher wieder gelöscht wird.



Microservice: finals

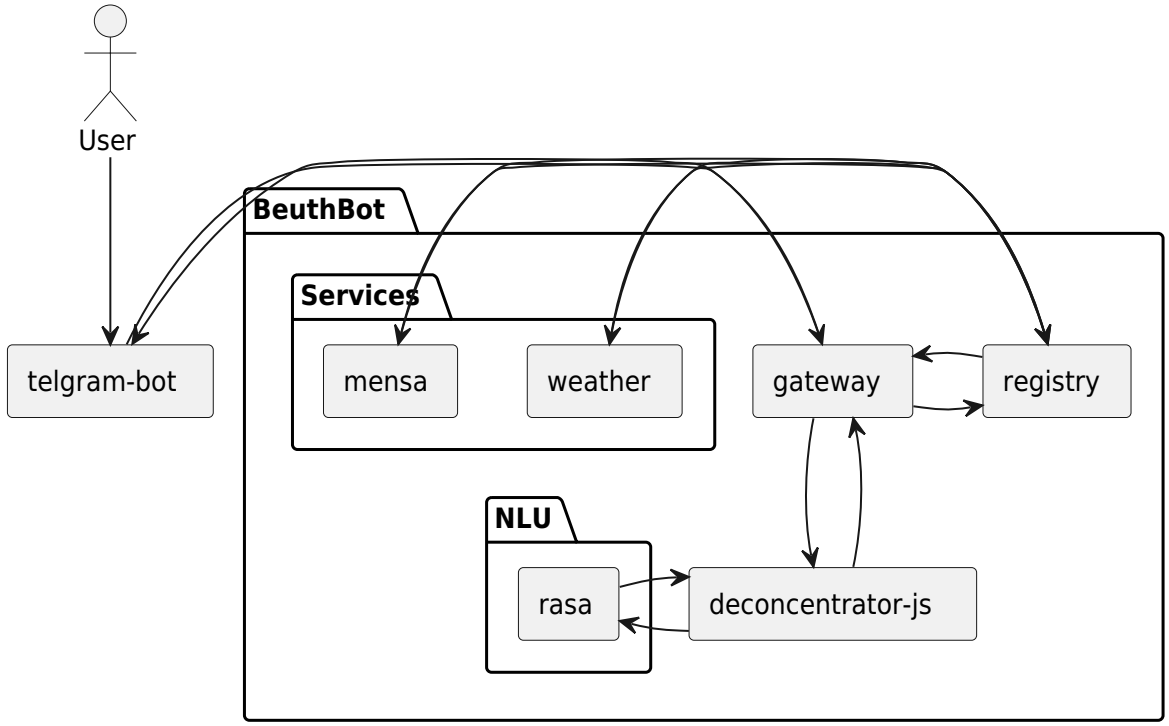
Dieser Dienst bietet dem User seine noch ausstehenden Prüfungen einsehen zu können. Dabei soll der Service das Datum der noch ausstehenden Abgaben bzw. Prüfungen, sowie dessen Fach und Inhalt wiedergeben. Auch soll dem User ermöglicht werden eigene Abgaben über eine Chat-Funktion hinzuzufügen, bzw. zu ändern. Die Daten werden einerseits aus dem vom User selbst eingetragenen Daten ermittelt, aber auch durch einen Scraper, welcher die aktuell eingetragenen Prüfungen des jeweiligen Faches über die Beuth Website ermittelt. Für das automatische ermitteln der Prüfungsdaten muss entweder der Studiengang, sowie das Semester und der Zug eingegeben werden, oder aber der Stundenplan als PDF gesendet werden.



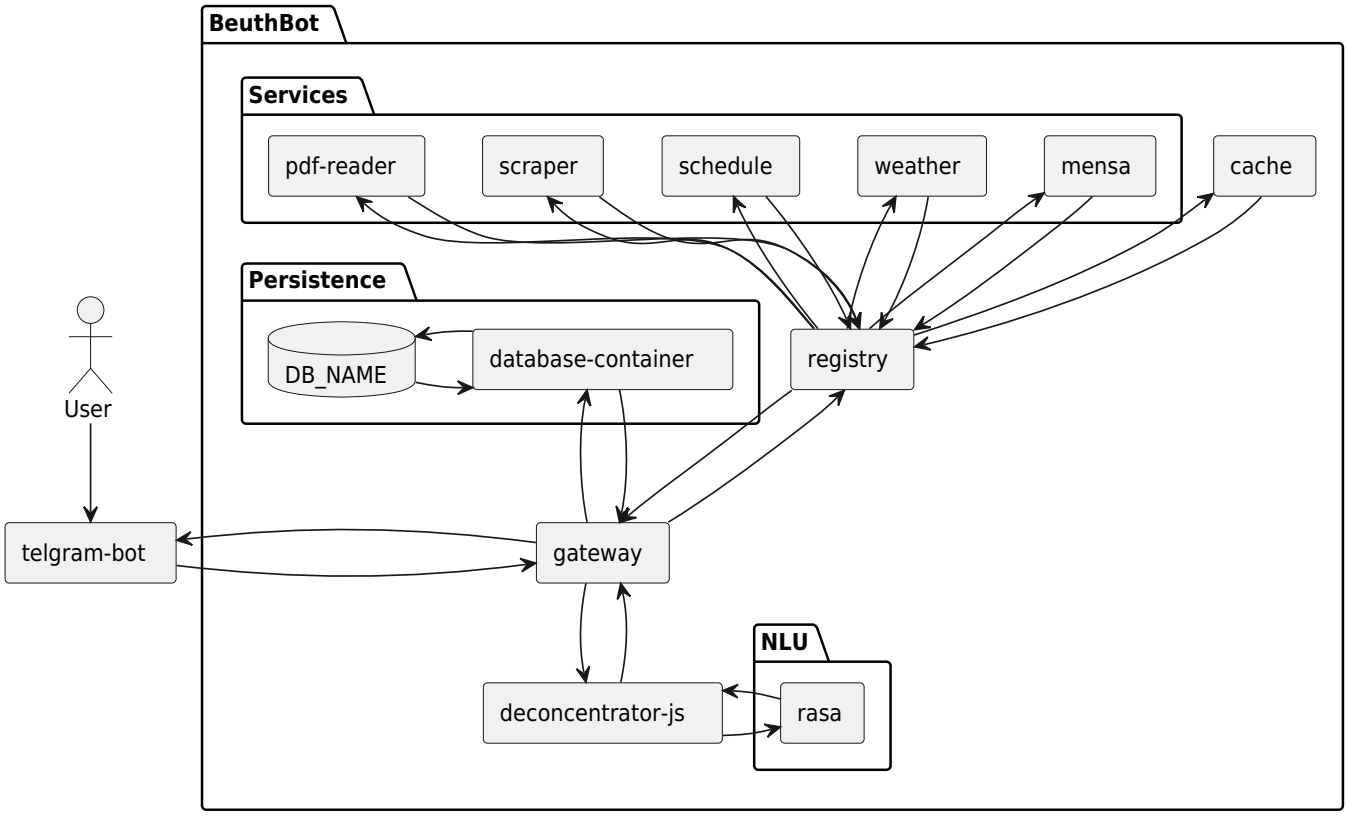
Goal for end of semester

Following diagram demonstrates the current state of the working component of the BeuthBot.

Current State:



Following diagram demonstrates the target state of the working component of the BeuthBot.



Timeplan

subject to change

@startuml

project starts the 2020/04/23
[Introductory training & building comprehension] as [IT] lasts 25 days
-- Transitional tasks --
then [Replace Deconcentrator] as [D] lasts 7 days
then [Conflate project] as [C] lasts 4 day

-- Implementation --
[Persist user preferences (Lukas & Tobias)] as [I1] lasts 30 days and starts 5 days after [C]'s end
[Cache microservices responses (Jan)] as [I2] lasts 30 days and starts 5 days after [C]'s end
[Transform scraper microservice (Denny & Jan)] as [I3] lasts 30 days and starts 5 days after [C]'s end
[Adjust weather microservice (Denny)] as [I4] lasts 30 days and starts 5 days after [C]'s end
[New course schedule microservice (?) (Denny & Jan)] as [I5] lasts 30 days and starts 5 days after [C]'s end
[C] -> [I1]
[C] -> [I2]
[C] -> [I3]
[C] -> [I4]
[C] -> [I5]
[D] is 80% completed
[I1] is 0% completed
[I2] is 0% completed
[I3] is 0% completed
[I4] is 0% completed
[I5] is 0% completed
@enduml