Software Architecture

## Table of content

Overview

BeuthBot consists of many interwoven *Microservices*. Evey Microservice uses our basic API to communicate with other Microservices. This approach enables us to change parts of the system easily at any time or to introduce new Microservices, all they need to do is to implement our API.

Basic Structure

Our application is basically composed of the following four components.

> Bot ⇔ Gateway ⇔ Registry ⇔ Service

Following diagram shows that in more detail.
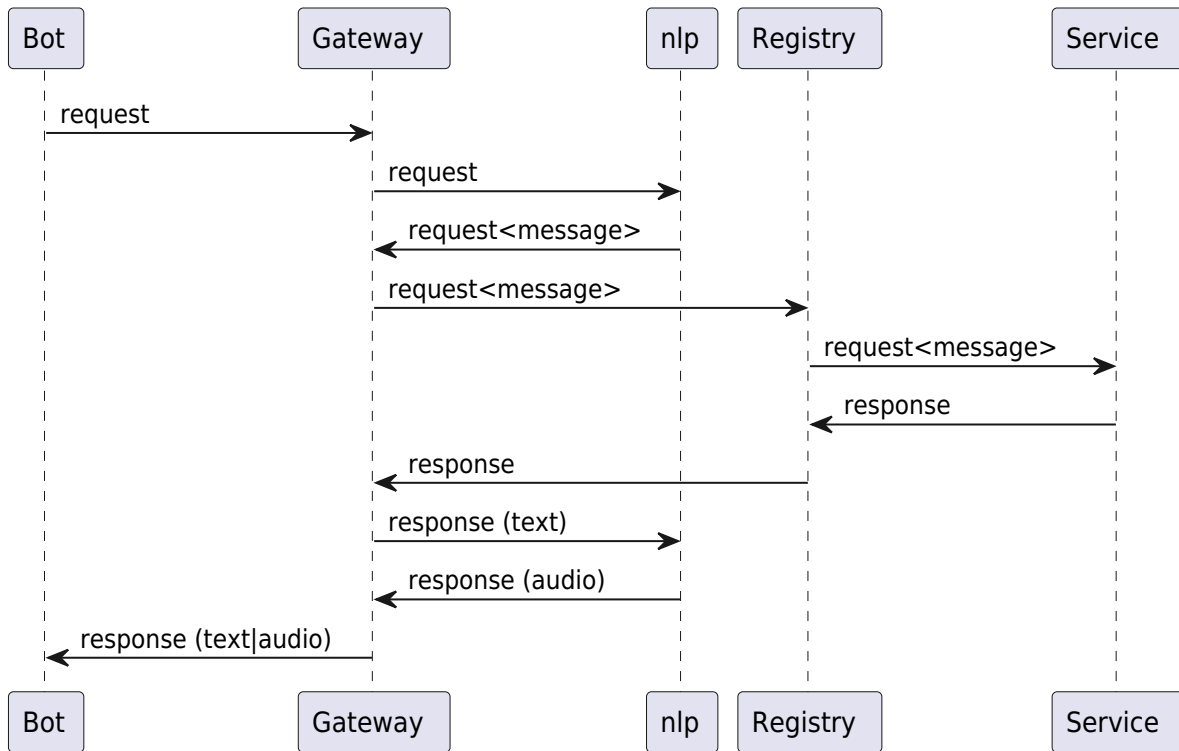
A user can write the *Bot* to request informations, the meaning of the message is extracted and a fitting *Microservice* is choosen to retrieve the necessary data. A response is build from that data and distributed back up to the bot which answers the users request.

following sequence diagram further illustrates that.

![flow](../assets/flow.png)

Bot

This is an abstraction for the available chatbots, e.g. a *Bot* for *Telegram* and another *Bot* for *WhatsApp*.

The user interacts with this *Microservice*, here she can request information and gets answers from *BeuthBot*.

Gateway

The *Gateway* is the centerpiece of *BeuthBot* one could say.

The *Bot* notifies the *Gateway* with the message it got from the user.

The *Gateway* then uses NLP (Natural Language Processing) *Microservices* to get the meaning and intention of the user. Here we try to extract what the user wants from *BeuthBot*, to notify the right service and present a fitting answer to our user.

Registry

After obtaining the intention of our user, the *Gateway* notifies the *Registry*, to get the information the user requested.

The Registry distributes the request to the correct *Service*, that takes care of retrieving the right informations.

Service

Service is an abstraction for the implemented *Microservices* that retrieve the necessary data we need to answer users requests. E.g. the *MensaService* is a *Microservice* that can give informations about the current menu, filtered by a number of parameters, e.g. a vegan user.

API

Because of the complexity of the single *Microservices*, every single *Microservice* implements its own, distinct, API.

But to answer a users request we use a unified, comprehensive API. Its basic idea is to pass a *Response*-Object trough the individual *Microservices*, which consists of the initial request, an answer as a response to the users request and informations about the user.

Following class diagram further illustrates that:

![flow](../assets/response-request-api.png)