

Software Architecture

Table of Contents

<!-- TOC depthFrom:2 depthTo:6 withLinks:1 updateOnSave:1 orderedList:0 -->

- [Table of Contents](#table-of-contents) - [Overview](#overview) - [Basic Structure](#basic-structure)

1. [Bot](#bot)
2. [Gateway](#gateway)
3. [Registry](#registry)
4. [Service](#service)

- [API](#api)

<!-- /TOC -->

Overview

BeuthBot consists of many interwoven Microservices. Every 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.

![structure](../assets/structure-without-notes.png)

A user can write the Bot to request informations, the meaning of the message is extracted and a fitting Microservice is chosen 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)]

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.