

database

Motivation

Die Motivation hinter einer Datenbank im BeuthBot Projekt kommt durch das Problem, dass Benutzer ihre Wünsche immer wieder komplett ausführen müssen.

Als Beispiel: Wenn der Benutzer die Mensa nach veganen Gerichten anfragt, dann muss er das bei der nächsten Anfrage wiederholen.

Die Datenbank soll das Problem beheben und den Benutzern die Möglichkeit bieten ihre Vorlieben zu persistieren, ohne dass diese sich einen extra Account anlegen müssen. Dabei muss darauf geachtet werden, dass in der Zukunft noch neue Services dazu kommen können. Die Architektur und die Datenbank sollten so konzipiert werden, dass neue Details die zu neuen Services gehören gespeichert werden können, ohne dass die Datenbank dazu angepasst werden muss.

Requirements

Was soll die DB können?

Functional

- /DBF100/ The system must be able to store details about a user.
- /DBF101/ The system must be able to add a detail related to a user.
- /DBF102/ The system must be able to change a detail related to a user.
- /DBF103/ The system must be able to delete a detail related to a user.
- /DBF104/ The system must be able to load all details about a user.
- /DBF105/ The system must be able to delete all entries related to a user.
- /DBF106/ It must be able to add new services and store related user details without modifying the database.

- /DBF200/ The database must store data related to a user.
- /DBF201/ The database must be able to store a nickname related to a user.
- /DBF202/ The database must be able to store a new detail about a user without scaling the schema.
- /DBF203/ The database must have a capacity of N.

Non Functional

- /DBNF200/ The database must be easily scalable.
- /DBNF201/ The database must be easy replaceable.

Referenz WS2019:

- /NF500/ The system should comply with DSGVO guidelines
- /NF501/ The system should be based on security standards
- /NF502/ Databases should be protected from unwanted access

- /NF503/ The databases should be password protected
- /NF504/ The databases should be based on security standards
- /NF600/ The system should restart the service independently in the event of a service failure
- /NF700/ The system should be well documented
- /NF701/ The system should be easy to understand

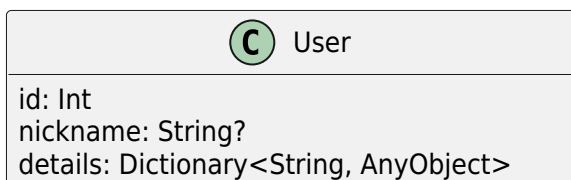
User Stories

"Als <Rolle> möchte ich <Ziel/Wunsch>, um <Nutzen>"

- /DBUS100/ Als Student möchte ich meine Vorlieben speichern, damit ich sie nicht immer wieder ausschreiben muss.
- /DBUS101/ Als Student möchte ich nach einigen Anfragen, dass ich gefragt werde ob ich meine Vorlieben speichern möchte, damit ich sie nicht immer wieder ausschreiben muss.
- /DBUS102/ Als Student möchte ich, dass der Bot mich wiedererkennt ohne einen extra Account anlegen zu müssen, damit ich keine zusätzlichen persönlichen Informationen preisgeben muss.
- /DBUS103/ Als Student möchte ich dem Bot sagen können, dass er meine ALLE meine Daten löschen soll.
- /DBUS103/ Als Student möchte ich dem Bot sagen können, dass er ein Detail über mich löschen soll.
- /DBUS104/ Als ... möchte ich ..., damit ich ...

Use Cases (?)

Klassendiagramm User



Technologies

Durch die Anforderung, dass die Details, die zu einem User gespeichert werden sehr variabel sein können, ist von einer relationalen Datenbank wie MySQL o.ä. abzuraten. MongoDB ist eine dokumentenorientierte NoSQL-Datenbank. Mit ihr können Sammlungen von JSON-ähnlichen Dokumenten erstellt und verwaltet werden. So können wir die Daten zu einem User in komplexen Hierarchien verschachteln und erweitern ohne uns Gedanken zu einem Tabellen-Schema machen zu müssen.

- MongoDB [Link](#)
- MongoDB Docker Image [Link](#)

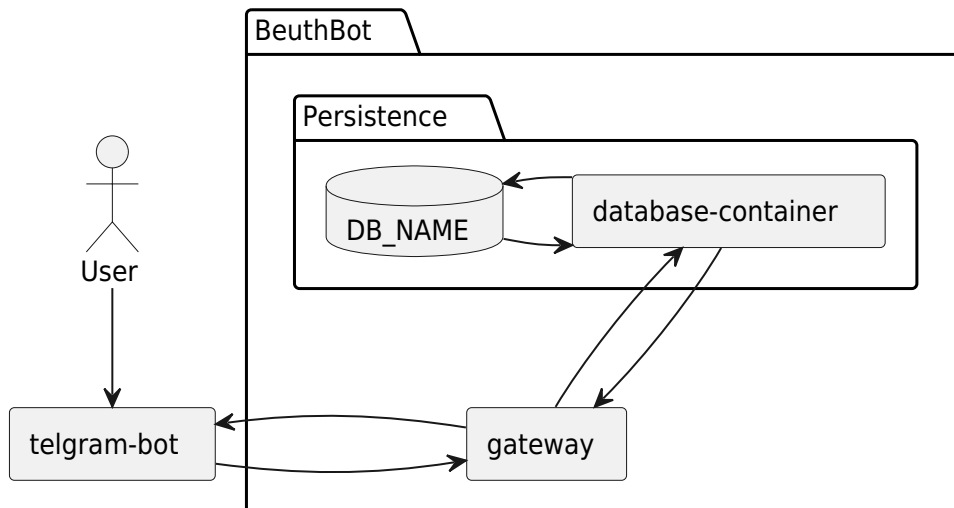
Function

Integration

An welcher Stelle wird die DB in das System eingebaut?

Wie wird sie eingebaut?

Wie sieht das Endprodukt aus?



Sequenzdiagramm mit angesteuertem Service

```

@startuml
participant gateway as GW
participant deconcentrator as DC
box
participant databasecontroller as DBC
database database as DB
endbox
participant service as SE

```

```

GW -> DC: Request
DC -> GW: Response (Intent + [Args])

```

```

GW -> DBC: Request User
DBC -> DB: Get User
DB -> DBC: Return User
DBC -> GW: Respones User

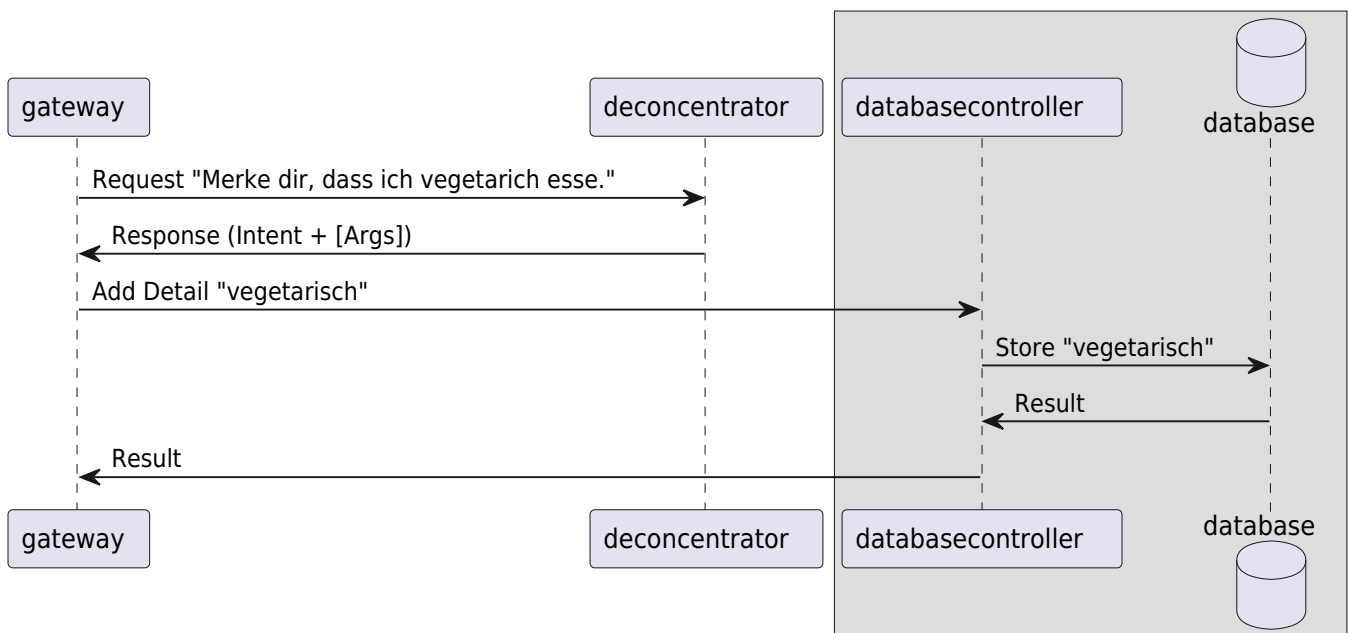
```

```

GW -> SE: Request (User + [Args])
SE -> GW: Answere
@enduml

```

Sequenzdiagramm nur Datenbank betreffend



Nächsten Schritte

- Projekt Ordner und GitHub Repository erstellen
- DB dem BeuthBot Projekt hinzufügen
- Geeignete Dockerfile formulieren mit MongoDB als Abhängigkeit
- Datenbankcontroller erstellen, welcher ADD, REMOVE, CHANGE Befehle für Details entgegen nimmt
- Trainingsmodell für RASA für die Datenbank erstellen
- Erste versuche mit dem Trainingsmodell von RASA

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.