DHBW Ravensburg
Duale Hochschule
Baden-Württemberg

18 Juni, 2023

EchoChat Dokumentation

Verteilte Systeme Testat, TIT20

Baldur Siegel **Jannik Herzner** **Jonas Alber**
Jonas Straub **Marvin Franke**

Inhaltsverzeichnis

1 Allgemeine Beschreibung

EchoChat ist ein webbasierter Chat-Service, der darauf beruht, Nutzer frei ihre Gedanken zu äußern zu lassen. Der Name besteht aus den zwei englischen Wörtern, Echo und Chat, die beschreiben, wie dieser Service agiert: der Nutzer bekommt nur die Wiederholung, sprich den Echo der Chatnachricht, von anderen Nutzern. Chatnachrichten werden nur einmal geschickt (sprich: ein Echo) und jeder Nutzer muss aktiv dabei sein, wenn er diese Nachricht bekommen möchte. Dabei werden Nachrichten nie serverside gespeichert und existieren nur so lange, wie die Browserfenster offen bleiben.

2 Technologiewahl

Als Gruppe wollten wir in erster Linie die Grenze zur Verwendung von unserem Publischer-Subscriber Modell so einfach wie möglich gestalten. Um dies zu erreichen haben wir entschieden, einen einfachen Web Client zu unser CLI zu gestalten. Webseiten sind in der heutigen Zeit die gängiste Methode mit kollaborativen Anwendungen zu interagieren.

Wir haben als Programmiersprache Python ausgewählt, da dies nicht nur sehr portabel ist (sprich: die Möglichkeit zur Weiterentwicklung erhöht), sondern auch die Programmiererfahrungen von der Gruppe wiederspiegelt.

3 Architekturbeschreibung

Die Architektur von dem System folgt dem Publisher-Subscriber-Modell, wo jeder Client sich mit einem zentralen Server verbindet (siehe Abbildung 1).

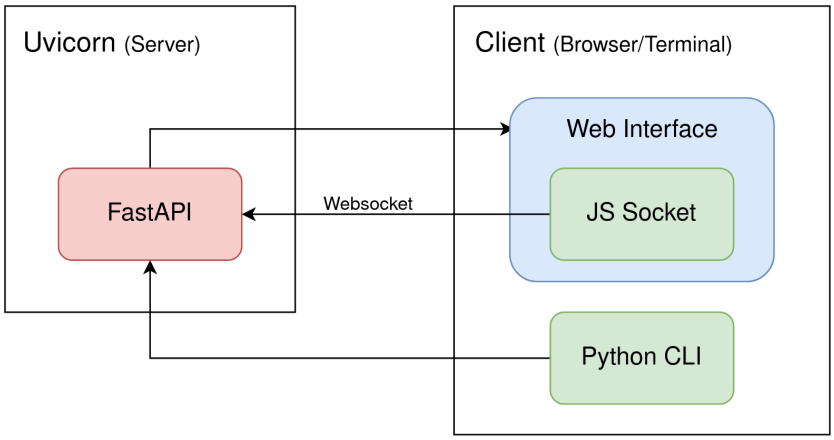

Abbildung 1: *Blockdiagramm von EchoChat*

Der Server stellt eine Weboberfläche sowie eine Websocket-Verbindung zur Verfügung . Die Clients verbinden sich mittels Websocket-Protocol zu dem Server, entweder durch das Webinterface, oder direkt mit einer Python CLI.

Im Folgenden wird die getroffene Entwurfsentscheidung begründet. Für die Entwicklung des Projekts haben wir uns für den Einsatz von Unicorn und FastAPI entschieden, da diese Kombination sich in der Praxis als äußerst effektiv erwiesen hat.

Basierend auf empirischen Daten und Erfahrungen konnten wir feststellen, dass Unicorn als HTTP-Server und FastAPI als Webframework gut miteinander harmonieren und eine hohe Leistungsfähigkeit bieten.

Ein weiterer wichtiger Aspekt unserer Entscheidung war die Verwendung von Multi-Thread und asynchronen Servertechnologien. Diese ermöglichen eine effiziente Verarbeitung von Anfragen und eine verbesserte Skalierbarkeit des Systems. Durch Multi-Threading können mehrere Anfragen parallel bearbeitet werden, während die asynchrone Serverarchitektur die Möglichkeit bietet, zeitintensive Operationen zu verwalten, ohne dabei die Serverreaktionszeit zu beeinträchtigen.

Die Wahl eine Webseite für die Benutzerinteraktion zu entwickeln, wurde getroffen, weil eine grafische Benutzeroberfläche (GUI) in vielen Fällen eine intuitivere und benutzerfreundlichere Interaktion ermöglicht. Im Vergleich dazu kann die Verwendung einer Kommandozeilenschnittstelle (CLI) als „schwieriger“ empfunden werden, insbesondere für Benutzer, die weniger technisch versiert sind. Außerdem kann man durch die CLI als Entwickler auch das System testen und überwachen.

Schließlich haben wir uns für die Verwendung von WebSocket als Kommunikationsprotokoll entschieden. Durch die Verwendung von WebSocket können Echtzeitkommunikation und bidirektionale Datenübertragung zwischen dem Server und dem Client ermöglicht werden. Dies ist besonders wichtig, wenn es um eine reaktionsschnelle und dynamische Benutzererfahrung geht.

4 Externe Bibliotheken (Dependencies)

Um jeweils den Server und die Clients zu betreiben, werden mehrere Dependencies verwendet. Diese sind unterschiedlich zwischen Server, CLI Client und Web Client.

4.1 Server

Verwendete Module:

- `FastAPI`: Webserver Routes und Websocket
- `Uvicorn`: Webserver Laufzeitumgebung
- `asyncio`: Erstellung asynchroner Funktionen
- `threading`: Erstellung nebenläufige Threads
- `uuid`: Eindeutige IDs für Websocket Verbindungen
- `json`: Serialiserung und Deserialisierung von JSON Strings
- `datetime`: Erstellung von Zeitstempel

Der Server verwendet primär `FastAPI` als ein Framework für die Web Routes sowie als Abstraktion für die Websocket Schnittstelle selbst. Zusätzlich wird `Uvicorn` als Laufzeitumgebung verwendet. Das `uuid` Modul wird benutzt, um jede Websocket Verbindung eindeutig zu identifizieren. Es werden auch einige Module von Python eingebaut, insbe-

sondere `asyncio` und `threading` für Nebenläufigkeit, und `json` und `datetime` für ihre jeweiligen Fähigkeiten.

4.2 CLI Client

Verwendete Module:

- `websockets`: Erstellung von einer websocket Verbindung zum server
- `argparse`: Auslesen von Kommandozeilenparameter
- `asyncio`: Erstellung asynchroner Funktionen
- `threading`: Erstellung nebenläufige Threads
- `json`: Serialiserung und Deserialisierung von JSON Strings

Der CLI Client verwendet `websockets`, um die Verbindung zum Webserver zu gestalten. Das Modul `argparse` wird angewendet, um eine statische CLI zu erstellen, in dem Kommandozeilenparameter ausgelesen werden. Zusätzlich werden asynchrone und nebenläufige Funktionen angewendet, die jeweils die `asyncio` und `threading` Module anwenden. Um die Kommunikation von dem Server zu vollenden, wird noch das `json` Modul gebraucht.

4.3 Web Client

Verwendete Module:

- `WebSocket API`: Erstellung von einer Websocketverbindung zum server
- `Bootstrap`: Frontend libraries für CSS und JS
- `Google Fonts`: Jeweilige darstellung von Text

Die Website verwendet die im Browser eingebaute `Websocket API`, um die Serververbindung zu erstellen. Zusätzlich werden `Bootstrap` und Schriftarten von `Google Fonts` geladen, um das Design zu gestalten.

5 Schnittstelle Server/Client (API Reference)

Die Kommunikation über die Websocketschnittstelle basiert auf einem Request-Response-System. Diese Requests und Responses können generisch abgebildet werden (siehe Listing 2, Listing 3).

Dadurch, dass alle Requests und Responses diesem Schema folgen, vereinfacht sich die implementierung von dem Server und Client.

```
{
  "function": "[function_name]",
  "parameters": {
    "[parameter]": "[value]",
    ...
  },
}
```

Listing 1: Ein generischer Request

```
{
  "status": "[status]",
  "function": "[function_name]",
  "data": {
    "[data_field]": "[value]",
    ...
  },
  "error": "[error_message]"
}
```

Listing 2: Ein generischer Response

Überblick der generischen Parameter

- `[function_name]` : einer der EchoChat Server funktionen, mögliche Werte:
  - `SUBSCRIBE_TOPIC` (siehe 5.1) : subscribet den Client zu einem Topic
  - `UNSUBSCRIBE_TOPIC` (siehe 5.2) : unsubscribet den Client von einem Topic
  - `PUBLISH_TOPIC` (siehe 5.3) : published eine Nachricht zu einem Topic
  - `LIST_TOPICS` (siehe 5.4) : gibt eine Liste verfügbarer Topics zurück
  - `GET_TOPIC_STATUS` (siehe 5.5) : gibt Informationen zu einem Topic zurück
  - `UPDATE_TOPIC` (siehe 5.6) : ist eine neue Nachricht zu einem Topic
- `[parameter]` : einer der EchoChat Server funktionen, mit möglichen Werten:
  - `name` : der Name von einem Topic
  - `message` : eine Nachtricht die an einem Topic gepublisht werden soll
- `[status]` : beschreibt ob ein Request erfolgreich erfüllt worden ist oder nicht
  - `"success"` : der Request würde erfolgreich erfüllt
  - `"failure"` : der Request würde nicht erfolgreich erfüllt; normal gibt es hierzu einen Error message mit einem Grund, warum der Request fehlgeschlagen ist
- `[data_field]` : ist ein Key bei der Rückgabe von Informationen, welche spezifisch vorhanden sind ist abhängig von dem jeweiligen Request
  - `topic` : der Topic der für den Request verändert worden ist
  - `topic_list` : eine Liste von vorhandenen Topics
  - `topic_status` : boolean, ob der aktuelle Client zu einem Topic subscribt ist
  - `message` : eine Nachricht die veröffentlicht worden ist
  - `timestamp` : der Zeitstempel von einer Nachricht (`YYYY-MM-DD HH:MM:SS`)
  - `last_update` : der Zeitstempel von dem letzten Update (`YYYY-MM-DD HH:MM:SS`)
  - `subscribers` : die Anzahl von Subscriber bei einem Topic
- `[error_message]` : ein Grund, weshalb ein Request fehlgeschlagen ist
  - `""` : der Request hat kein Error, der würde erfolgreich bearbeitet
  - `"topic does not exist"` : der Request hat ein Topic referenziert der nicht existiert

- `"you aren't subscribed to that topic"` : der Request hat ein Topic referenziert wo der Client nicht subscribet ist
- `"could not interpret request"` : der Request wurde nicht richtig formattiert, oder ruft probiert eine Funktion `[function_name]` aufzurufen die nicht existiert
- `"unknown server error"` : ein unbekannter Server Fehler ist passiert, der nicht gehandelt werden konnte; hier ist `"data":` einen String mit dem Error

5.1 Subscribe Topic

Client-Initiiert

```
{
  "function": "SUBSCRIBE_TOPIC",
  "parameters": {
    "name": "[topic_name]",
  },
}
```

Listing 3: Subscribe Request

```
{
  "status": "success",
  "function": "SUBSCRIBE_TOPIC",
  "data": {
    "topic": "[topic_name]"
  },
  "error": ""
}
```

Listing 4: Subscribe Response

Beschreibung der Parameter vom Request (Listing 3):

`function (str)` name der auszuführenden Funktion

`parameter (obj)` übergebene Daten

`└ name (str)` der Name von der Topic (aktuell keine Beschränkung bei verwendeten Zeichen (UTF8) oder Zeichenlänge)

Beschreibung der Parameter vom Response (Listing 4):

`status (str)` der Erfolg von dem Request, entweder `success` oder `failure`

`function (str)` die Rückgabe der angeforderten Funktion aus dem Request

`data (obj)` enthält die Daten für den Request

`└ topic (str)` den Topic namen von dem der Status abgefragt worden ist

`error(str)` erklärt den Fehlschlag, falls es einen gibt. Mögliche Werte: `""`, `"topic does not exist"`

5.2 Unsubscribe Topic

Client-Initiiert

```
{
  "function": "UNSUBSCRIBE_TOPIC",
  "parameters": {
    "name": "[topic_name]",
  },
}
```

Listing 5: Unsubscribe Request

```
{
  "status": "success",
  "function": "UNSUBSCRIBE_TOPIC",
  "data": {
    "topic": "[topic_name]"
  },
  "error": ""
}
```

Listing 6: Unsubscribe Response

Beschreibung der Parameter vom Request (Listing 5):
`function (str)` name der auszuführenden Funktion
`parameter (obj)` übergebene Daten
`└ name (str)` der Name von der Topic

Beschreibung der Parameter vom Response (Listing 6):
`status (str)` der Erfolg von dem Request, entweder `success` oder `failure`
`function (str)` die Rückgabe der angeforderten Funktion aus dem Request
`data (obj)` enthält die Daten für den Request
`└ topic (str)` den Topic namen von dem der Status abgefragt worden ist
`error (str)` erklärt den Fehlschlag, falls es einen gibt. Mögliche Werte: `""`, `"topic does not exist"`, `"you aren't subscribed to that topic"`

5.3 Publish Topic
Client-Initiiert

```
{
  "function": "PUBLISH_TOPIC",
  "parameters": {
    "name": "[topic_name]",
    "message": "[topic_message]"
  },
}
```

Listing 7: Publish request

```
{
  "status": "success",
  "function": "PUBLISH_TOPIC",
  "data": {
    "topic": "[topic_name]"
    "message": "[topic_message]"
  },
  "error": ""
}
```

Listing 8: Publish response

Beschreibung der Parameter vom Request (Listing 7):
`function (str)` name der auszuführenden Funktion
`parameter (obj)` übergebene Daten
`└ name (str)` der Name von der Topic
`└ message (str)` die Nachricht, welche man veröffentlichtlichen möchte

Beschreibung der Parameter vom Response (Listing 8):
`status (str)` der Erfolg von dem Request, entweder `success` oder `failure`
`function (str)` die Rückgabe der angeforderten Funktion aus dem Request
`data (obj)` enthält die Daten für den Request
`└ topic (str)` den Topic namen von dem der Status abgefragt worden ist
`└ message (str)` die Nachricht, die veröffentlichtlicht worden ist
`error (str)` erklärt den Fehlschlag, falls es einen gibt. Mögliche Werte: `""`, `"topic does not exist"`, `"you aren't subscribed to that topic"`

5.4 List Topic

Client-Initiiert

```
{

  "function": "LIST_TOPICS",
  "parameters": {},

}
```

Listing 9: List request

```
{
  "status": "success",
  "function": "LIST_TOPICS",
  "data": {
    "topic_list": ["[topic_name]", ..]
  },
  "error": ""
}
```

Listing 10: List response

Beschreibung der Parameter vom Request (Listing 9):

`function (str)` name der auszuführenden Funktion
`parameter (obj)` übergebene Daten (hier leer, jedoch required)

Beschreibung der Parameter vom Response (Listing 10):

`status (str)` der Erfolg von dem Request, entweder `success` oder `failure`
`function (str)` die Rückgabe der angeforderten Funktion aus dem Request
`data (obj)` enthält die Daten für den Request
`┗ topic_list (arr)` enthält die Daten für den Request
`error (str)` leerer String

5.5 Get Topic Status

Client-Initiiert

```
{
  "function": "GET_TOPIC_STATUS",
  "parameters": {
    "name": "[topic_name]"
  }
}
```

Listing 11: Topic Status Request

```
{
  "status": "success",
  "function": "GET_TOPIC_STATUS",
  "data": {
    "topic": "[topic_name]",
    "topic_status": "[topic_status]",
    "last_update": "[timestamp]",
    "subscribers": "[int]",
  },
  "error": ""
}
```

Listing 12: Topic Status Response

Beschreibung der Parameter vom Request (Listing 11):

`function (str)` name der auszuführenden Funktion
`parameter (obj)` übergebene Daten
`┗ name (str)` der Name von der Topic

Beschreibung der Parameter vom Response (Listing 12):

`status (str)` der Erfolg von dem Request, entweder `success` oder `failure`
`function (str)` die Rückgabe der angeforderten Funktion aus dem Request

`data (obj)` enthält die Daten für den Request
`└ topic (str)` den Topic namen von dem der Status abgefragt worden ist
`└ topic_status (str)` gibt zurück, ob dem Topic subscribed wurde oder nicht. Mögliche Werte: `"Subscribed"`, `"Not Subscribed"`
`└ last_update (str)` gibt den Zeitpunkt des zuletzt gesendeten Updates zurück, im Format `YYYY-MM-DD HH:MM:SS`
`└ subscribers (int)` Anzahl der Subscriber eines Topics
`error (str)` erklärt den Fehlschlag, falls es einen gibt. Mögliche Werte: `""`, `"topic does not exist"`

5.6 Update Topic

Server-Initiiert

```
{
  "status": "success",
  "function": "UPDATE_TOPIC",
  "data": {
    "name": "[topic_name]",
    "message": "[topic_message]",
    "timestamp": "[timestamp]"
  },
  "error": ""
}
```

Listing 13: Topic Update Response

Beschreibung der Parameter vom Response:
`status (str)` der Erfolg von dem Request, entweder `success` oder `failure`
`function (str)` die Rückgabe der angeforderten Funktion aus dem Request
`data (obj)` enthält die Daten für den Request
`└ name (str)` den Topic namen von dem der Status abgefragt worden ist
`└ message (str)` der neue Topic Message
`└ timestamp (str)` gibt den Zeitpunkt des zuletzt gesendeten Updates zurück, im Format `YYYY-MM-DD HH:MM:SS`
`error (str)` leerer String

6 Installationsanleitung

Das Repository kann unter folgendem Link heruntergeladen werden:
https://github.com/DHBW-FN-TIT20/EchoChat

In dem Repository ist ein Script für Linux (mit Ubuntu getestet) um schnell den Server zu starten, jedoch ist dafür `python-venv` notwendig. Mit folgenden Kommandos können EchoChat gestartet werden:

```
# Linux
apt-get install Python3.10 Python3.10-venv
git clone https://github.com/DHBW-FN-TIT20/EchoChat
cd EchoChat && ./start.sh
```

EchoChat kann jedoch auch manuell gestartet werden, jedoch mit mindestens Python 3.10.6 vorausgesetzt:

```
# Linux
pip3 install fastapi
pip3 install uvicorn
pip3 install websockets
pip3 install argparse

# Windows
pip install fastapi
pip install uvicorn
pip install websockets
pip install argparse
```

```
# Linux
cd EchoChat && uvicorn app:app

# Windows
cd EchoChat
uvicorn app:app
```

Dadurch wird der Webserver normal unter folgender Adresse zur Verfügung gestellt:

```
http://127.0.0.1:8000
```

Zusätzlich ist auch in dem Repo auch eine Python CLI vorhanden, in `./app/client.py`. Die Verwendung von diesen Clients (Web- sowie CLI) sind im folgenden Abschnitt 7 dokumentiert.

7 Verwendung Client

7.1 CLI Client

7.1.1 Statisch

Usage:

```
(python) client.py [options] <command> [args]
```

Commands:

`list` gibt alle verfübgarben Topics aus und beendet

`status <topic>` gibt Informationen über den gegebenen Topic aus

`subscribe <topic>` subscribet zu dem Topic und gibt Updates zurück, blockiert den Terminalfenster

`publish <topic> <message>` subscribet zu dem Topic, veröffentlicht die Message und gibt danach Updates zurück, blockiert den Terminalfenster

Options:

`-c, --cli` unterdrückt das Interaktive CLI

`-o <type>, --output <type>` setzt das Outputformat, entweder `json` (default) or `human`

`-h, --help` gibt den Help Message aus (siehe Listing 14) und beendet

Beispiele:

```
python client.py --cli list
```

```
python client.py -co human subscribe -t General
```

```
python client.py --cli publish General "Hello, World!"
```

7.1.2 Interaktiv

Usage:

```
> <command>
>> <parameter>   ' wird erst nach einem Prompt angezeigt
```

Commands:

`subscribe, s` zu einem Topic subscriben

`unsubscribe, u` von einem Topic unsubscriben

`publish, p` eine nachricht an einem Topic schicken

`list, l` alle vorhandenen Topic ausgeben

`status, t` informationen zu einem Topic bekommen

`help, h` druckt die Kommandoliste erneut aus (siehe Listing 15})

`exit, e, quit, q` stoppt den client

7.2 Web Client

Der Web Client besteht aus einem minimalistischem Inferface (siehe Abbildung 2).

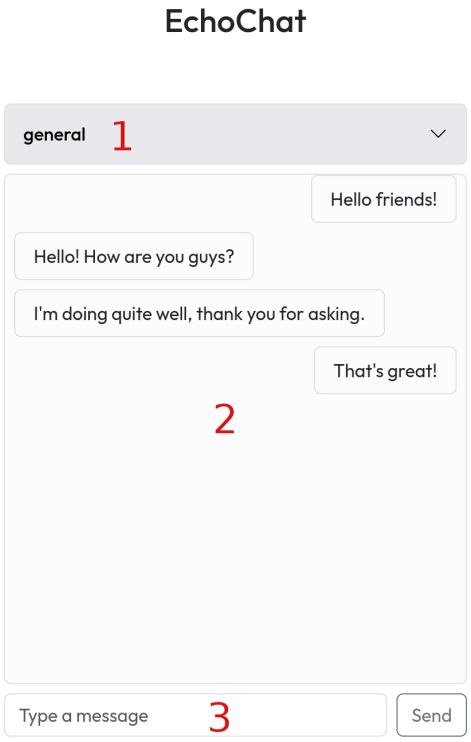

Abbildung 2: Das EchoChat Web Interface

1 Der Topic Selector. Hiermit kann per Linksclick der Topic Selection Modal geöffnet werden (s. Abbildung 3). Falls kein Topic ausgewählt ist, steht hier „Select a Topic“

2 Das Chatfenster. Hier stehen alle Updates zu einem Topic. Falls man selber einen Topic Update geschrieben hat, ist der Rechtsbündig, sonst Linksbündig.

3 Der Message Input. Mit dem Input können neue Topic Updates verfasst werden, die dann in dem ausgewählten Topic geschickt werden.

Nachdem ein Topic ausgewählt ist, werden Updates vom Topic automatisch in dem Chatfenster reingeschrieben. Neue Updates können in dem Message Input verfasst werden, und entweder mit dem „Send“ Knopf oder mit der Enter Taste verschickt werden.

Leere Nachrichten können nicht verschickt werden.

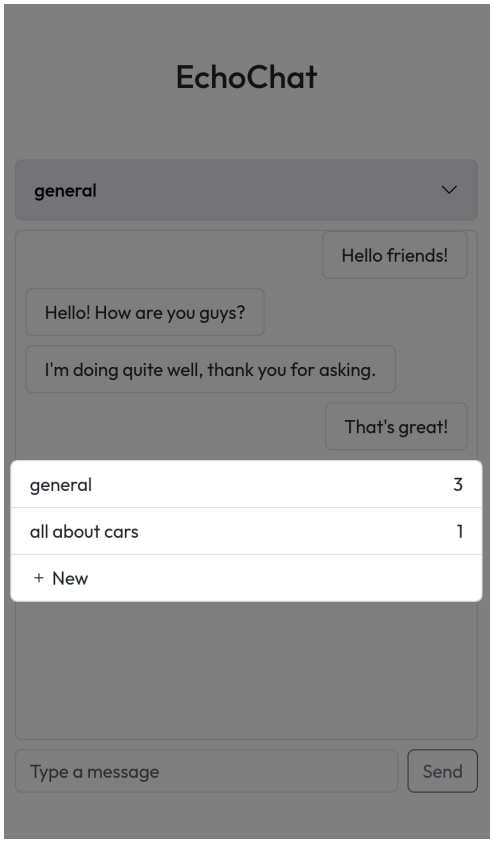

Abbildung 3: Das Topic Selection Modal

In dem Topic Selection Modal kann ausgewählt werden, von welchem Topic Nachrichten empfangen und gesendet werden. In dem Modal sind alle verfügbaren Topics sichtbar. Auf der rechten Seite von jedem Topic steht die Anzahl von aktiven Subscriber. Beide diese Informationen werden beim Aufruf vom Modal immer neu aktualisiert. Ein Topic kann einfach durch einem Click ausgewählt werden.

Neue Topics können mit einem Drück auf dem „+ New“ Knopf erstellt werden. Beim drücken von dieser Option wird ein Textfeld eingeblendet (siehe Abbildung 4).

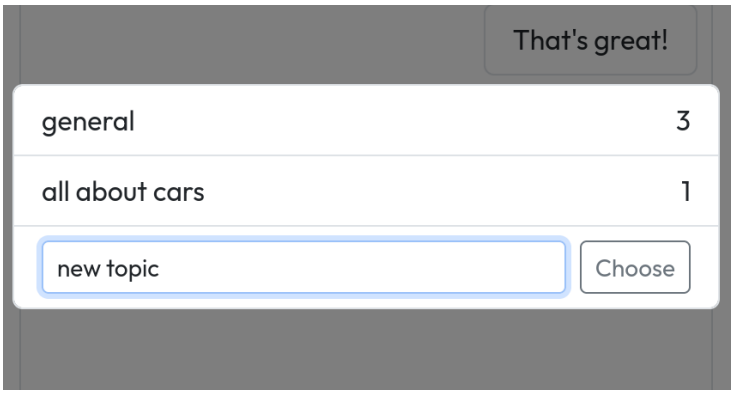

Abbildung 4: Erstellung von einem neuen Topic

In diesem Textfeld kann ein Text eingegeben werden, um einen neuen Topic zu erstellen. Mit druck auf dem „Choose“ Knopf oder der Enter Taste wird der Client auch automatisch zu diesem Topic subscribet.

8 Testumfang

Das Testkonzept ist manuell umgesetzt worden. Als erste priorität würde daruf gesetzt, die Schnittstelle ausgiebig zu testen. Dabei würden auf folgende Punkte geachtet:

- Richtigkeit der zurückgegeben Antworten (sprich: konformität zu den vorausgesetzten API Antworten die in diesem Dokument dokumentiert sind)
- Stabilität (sprich: Verhinderung von unauflösbare Server Errors)
- Abstraktion (sprich: Möglist leichte Verwendung der API, sodass die auch portable ist)

9 Anhang

9.1 EchoChat CLI Client Help

```
usage: client.py [-h] [-c] [-o [{json,human}]]
[{list,status,subscribe,publish}] [topic] [message]

EchoChat CLI Client

positional arguments:
  {list,status,subscribe,publish}
                        the command to be run, either:
                        list, status, subscribe, publish
  topic                 the topic parameter
  message               the message to publish

options:
  -h, --help            show this help message and exit
  -c, --cli             suppresses interactive interface
  -o [{json,human}], --output [{json,human}]
                        selects the output type, defaults to json
```

Listing 14: Generierte Help Message für die statische CLI

```
-----------------------------------------------------
======== EchoChat Interactive CLI Commands =========
-----------------------------------------------------
-> (s)ubscribe --- subscribe to a topic, creates
                   one if topic does not exist
-> (u)nsubscribe - unsubscribe from a topic
-> (p)ublish ----- send a message to a topic
-> (l)ist -------- list all existing topics
-> s(t)atus ------ check subscription to a topic
-> (h)elp -------- see this menu again
-> (e)xit/(q)uit - stop the client

Hint: run the client with --help to see static
      command line options
```

Listing 15: Help message für die interaktive CLI

```
------------------------------------------------------
=========== EchoChat Static CLI Commands ============
------------------------------------------------------

usage: (python) client.py [options] <command> [args]

>> Commands <<

list
lists all available topics and returns

status <topic>
outputs information about the given topic

subscribe <topic>
subscribes and outputs information when published,
blocks the terminal

publish <topic> <message>
subscribes to the topic and publishes the given
message, blocks the terminal



>> Options <<

-c, --cli
suppresses interactive interface

-o [type], --output [type]
selects the output type, either 'json' or 'human',
defaults to json

-h, --help
prints this list and returns


>> Examples <<

python client.py --cli list

python client.py --cli subscribe General

python client.py --cli publish General "Hello, World!"
```

Listing 16: Selbstgeschriebene Help Message für die statische CLI