Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Marius Weidner
Simon Dorer, Timpe Hörig

Universität Freiburg
Institut für Informatik
Wintersemester 2025

Übungsblatt 10

Abgabe: Mittwoch, 07.01.2026, 09:00 Uhr

Sie alle sind wahrscheinlich mit Sozialen Netzwerken vertraut, und da die Welt noch nicht genügend davon hat, erstellen wir jetzt unser eigenes. Durch eine qualifizierte Marktanalyse haben wir herausgefunden, dass es noch kein Soziales Netzwerk für ausschließlich Jugendliche gibt und das wollen wir ändern. Um die Handhabung ansprechender für Jugendliche zu machen, verwenden wir das in Jugendkreisen weit verbreitete Konzept von *Yeet*. Ein Yeet ist eine Nachricht, die eine Benutzerin an alle ihre Freundinnen sendet, ähnlich zu einem Tweet auf der ehemaligen Plattform Twitter. Yeets können von anderen Nutzerinnen geliked werden und neue Yeets als Antworten erhalten.

Ein Soziales Netzwerk besteht in der Regel aus zwei Teilen: Einem zentralen *Server*, auf dem alle Daten gespeichert sind, und aus mehreren *Clients*, die Anfragen an den Server senden. In diesem Übungsblatt schreibt jeder von Ihnen einen solchen Client, der mit dem von uns bereitgestellten Server interagiert.

Für dieses Übungsblatt benötigen Sie die in der Vorlesung vorgestellte `requests`-Library. Installieren Sie diese, indem Sie — wie in Blatt 06 beschrieben — zunächst eine neue virtuelle Umgebung (`venv`) anlegen. Im Anschluss können Sie dort mit dem Befehl `python3 -m pip install requests` das Modul `requests` installieren.[1]

Zum Bearbeiten der folgenden Aufgaben haben wir Ihnen ein Template `client.py` in Ihrem Repository[2] bereitgestellt. Bevor Sie mit den Aufgaben beginnen, laden Sie sich bitte das Template herunter. Dies können Sie entweder direkt über die Weboberfläche oder über die Kommandozeile mit `git pull` tun. **Verwenden Sie das Template, um Ihre Lösungen zu implementieren**.

Aufgabe 10.1 (Yeets; 5 Punkte; Datei: `client.py`)

(a) **Datenklasse Yeet; 1 Punkt**
Zunächst wollen wir ein Grundgerüst für Nachrichten (*Yeets*) in unserem sozialen Netzwerk erstellen. Definieren Sie eine Datenklasse `Yeet`, die eine ID `yeet_id` (`int`), ein Datum `date` (`float`), einen Inhalt `content` (`str`), einen Autor `author` (`str`) und eine *optionale* Referenz auf die ID eines anderen Yeets `reply_to` als Attribute besitzt.

[1] Sollten bei der Installation Probleme auftreten, melden Sie sich gerne im Chat. Alternativ können Sie `requests` auch systemweit installieren, indem Sie unter Ubuntu/WSL den Befehl `sudo apt install python3-requests` verwenden.

[2] siehe: https://git.laurel.informatik.uni-freiburg.de/

Das Attribute `date` ist dabei eine Unixzeit[3] und zählt die Sekunden seit Donnerstag, dem 1. Januar 1970, 00:00 Uhr UTC.

(b) **Timestamp zu String; 1 Punkt**
Schreiben Sie eine Methode `date_str` für die Datenklasse `Yeet`. Diese soll einen String zurückgeben, der die Unixzeit `date` in einem Format mit Tag, Monat, Jahr, Stunde, Minute und Sekunde repräsentiert.

Um eine Unixzeit in ein lesbares Datum umzuwandeln, können Sie die `datetime` Klasse aus dem gleichnamigen Modul `datetime` verwenden:

```
>>> dt = datetime.fromtimestamp(1767772800)
```

Benutzen Sie dann die `datetime` Methode `strftime`, um einen (nach ihrem Ermessen) schönen String mit Hilfe von Format Codes[4] zu erzeugen.

Hier ein Beispiel für die Verwendung von `strftime`:

```
>>> dt.strftime("Greetings from %B of the upcoming year %Y!")
'Greetings from January of the upcoming year 2026!'
```

(c) **Vergleich von Yeets; 1 Punkt**
Definieren Sie die Methode `__lt__` für die Klasse `Yeet`, sodass zwei Yeets anhand ihres `date` Attributs verglichen werden können. Ein Yeet ist kleiner als ein anderer, wenn sein `date` Attribut kleiner ist. Dadurch können Sie in den späteren Aufgabenteilen die `sorted`-Funktion verwenden, um Yeets nach ihrem Datum zu sortieren.

(d) **Dictionary zu Yeet; 2 Punkte**
Die Kommunikation zwischen Client und Server findet in Form von Dictionaries[5] statt. Um die Antworten des Servers verarbeiten zu können, müssen wir diese in `Yeet`-Objekte umwandeln können.

Schreiben Sie eine *Funktion* (keine Methode!) `yeet_from_dict`, die ein Dictionary `yeet_as_dict` als Argument nimmt und einen `Yeet` mit den entsprechenden Attributwerten zurückgibt. Das Dictionary `yeet_as_dict` enthält dabei die Namen der Attribute als Schlüssel. Für die Werte dürfen Sie `Any` in Ihrer Typannotation verwenden.

Die Attribute `yeet_id`, `date`, `content` und `author` müssen als Schlüssel in `yeet_as_dict` vorkommen. Das Attribut `reply_to` kommt nur vor, falls der Yeet eine Antwort auf einen anderen Yeet ist. Falls der Yeet jedoch keine Antwort auf einen anderen Yeet ist, enthält das Dictionary keinen Schlüssel `reply_to`. Setzen Sie in diesem Fall das Attribut `reply_to` der Yeet-Datenklasse auf `None`. Falls ein Schlüssel in `yeet_as_dict` fehlt oder der Wert eines Schlüssels nicht dem erwarteten Typ entspricht, soll die Funktion einen Fehler werfen (mit `assert False`).

[3] siehe: https://de.wikipedia.org/wiki/Unixzeit
[4] siehe: https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes
[5] Genauer gesagt: JSON-Objekte

Verwenden Sie Pattern Matching, um die Werte aus dem Dictionary zu extrahieren.

Um Ihre Implementierung zu testen, können Sie die Funktion `test_yeet_from_dict` aus der Datei `test_client.py` verwenden.

Aufgabe 10.2 (Authentifizierung; 3 Punkte; Datei: `client.py`)

In ihrem Repository befindet sich bereits eine Datei `prelude.py` mit einer Datenklasse `Session`. Diese kümmert sich um die Kommunikation zwischen dem Client und dem Server. Eine Session ist immer mit einer Benutzerin verbunden. Mit der `login`-Methode kann sich eine Benutzerin in die Session einloggen. Zusätzlich speichert die Property `authenticated` (auf die Sie mit der Methode `get_authenticated` zugreifen können), ob die Session bereits authentifiziert wurde.

Schreiben Sie eine Funktion `authenticate`, die ein Argument `session` vom Typ `Session` als Argument nimmt und eine Benutzerin authentifiziert. Dabei soll zunächst geprüft werden, ob die Benutzerin bereits authentifiziert ist. Falls ja, soll nichts passieren. Falls nein, soll die Benutzerin solange nach ihrem Benutzernamen und Passwort gefragt werden, bis der Login erfolgreich war. Benutzen Sie zum Abfragen des Passworts die Funktion `getpass` aus dem Modul `getpass` anstelle von `input`, damit das Passwort bei der Eingabe nicht als Klartext zu sehen ist. Informieren Sie die Benutzerin stets mit `print`s über den Status des Anmeldevorgangs. Anmelden können Sie sich mit Ihrem RZ-Account (wie bei Git).

Aufgabe 10.3 (Serverinteraktion; 0 Punkte)

Die Klasse `Session` stellt außerdem die zwei Methoden `get` und `post`[6] bereit, um mit dem Server zu interagieren. Die Methode `get` nimmt einen Pfad (String) als Argument, der bestimmt, was vom Server angefragt werden soll. Entsprechend des Pfads gibt die Methode eine Antwort zurück. Die Methode `post` nimmt einen Pfad und ein passendes Dictionary, um dieses an den Server zu senden. Der Server gibt hier keine Daten zurück. Eine Auflistung der möglichen Pfade finden Sie unter https://courses.laurel.informatik.uni-freiburg.de/yee.
Dieses Webinterface erlaubt es auch, die Pfade auszuprobieren (benutzen Sie dafür den '`Try it out`' Button). Es wird zum Beispiel angezeigt, wie die von `get` zurückgegebenen Daten aussehen, oder welche Form die Daten für `post` beim jeweiligen Pfad haben müssen. Spielen Sie vorab ein wenig mit der Funktionalität des Servers um ein Gefühl für das Verhalten zu bekommen!

Aufgabe 10.4 (Wrapper Funktionen; 5 Punkte; Datei: `client.py`)

Serverabfragen können in der Regel sehr lang sein und viele Parameter haben. Um die Lesbarkeit des Codes zu verbessern, definieren wir Wrapper Funktionen, die die Serverabfragen kapseln. Diese Funktionen rufen die entsprechenden Methoden der Klasse `Session` auf und geben die Antwort zurück.

[6] Diese Methoden verwenden intern `get` und `post` der `requests`-Library. Die Server-URL wird von der Klasse `Session` angehängt, sodass nur der Restpfad (beispielsweise `/yee/users/all`) angegeben werden muss.

Implementieren Sie die folgenden Funktionen (die jeweiligen Argumente sind in den Klammern angegeben):

- `get_yeets_all(session)`: Gibt eine Liste der IDs aller Yeets zurück.
- `get_yeet(session, yeet_id)`: Gibt die Yeet-Instanz mit der ID `yeet_id` zurück.
- `get_likes(session, yeet_id)`: Gibt eine Liste an Benutzernamen zurück, die den Yeet mit der ID `yeet_id` geliked haben.
- `get_replies(session, yeet_id)`: Gibt eine Liste von IDs alles Yeets zurück, die auf den Yeet mit der ID `yeet_id` geantwortet haben.
- `add_yeet(session, content, reply_to)`: Fügt einen neuen Yeet mit dem Inhalt `content` hinzu. `reply_to` ist dabei eine optionale ID auf einen anderen Yeet, auf den geantwortet wird. Ist diese Referenz gegeben, wird der neue Yeet als Antwort auf den Yeet mit der ID `reply_to` hinzugefügt. Ansonsten wird der Yeet als neuer Top-Level Yeet hinzugefügt.
- `remove_yeet(session, yeet_id)`: Löscht den Yeet mit der ID `yeet_id`.
- `get_users_all(session)`: Gibt eine Liste mit allen Benutzernamen zurück.
- `get_yeets_from_user(session, user)`: Gibt eine Liste der IDs aller Yeets zurück, die von der Benutzerin `user` erstellt wurden.
- `get_likes_from_user(session, user)`: Gibt eine Liste der IDs aller Yeets zurück, die die Benutzerin `user` geliked hat.
- `get_following_from_user(session, user)`: Gibt eine Liste der Benutzernamen zurück, denen die Benutzerin `user` folgt.
- `get_followers_from_user(session, user)`: Gibt eine Liste der Benutzernamen zurück, die der Benutzerin `user` folgen.
- `add_like(session, yeet_id)`: Liked den Yeet mit der ID `yeet_id`.
- `remove_like(session, yeet_id)`: Entliked den Yeet mit der ID `yeet_id`.
- `add_follower(session, user)`: Folgt der Benutzerin `user`.
- `remove_follower(session, user)`: Entfolgt der Benutzerin `user`.

Sie können sich an folgenden Beispielen orientieren (Die Beispiele finden Sie auch im Template):

```python
def get_likes(session: Session, yeet_id: int) -> list[str]:
    return session.get(f"/yee/yeets/{yeet_id}/likes")["response"]

def add_like(session: Session, yeet_id: int):
    session.post("/yee/likes/add", {"yeet_id": yeet_id})
```

Aufgabe 10.5 (Yeet zu String; 1 Punkte; Datei: `client.py`)

Schreiben Sie eine Funktion `yeet_to_string`, die eine Session vom Typ `Session` und einen Yeet vom Typ `Yeet` als Argumente erhält und den Yeet als String zurückgibt. Der String soll mindestens die ID, das Datum, den Autor, den Inhalt, die Anzahl der Likes und die Anzahl der Kommentare des Yeets enthalten. Verwenden Sie zur Implementierung Funktionen aus Teilaufgabe 4. Das genaue Format des Strings ist Ihnen überlassen, solange die Informationen klar erkennbar sind.

Optional: Falls Sie Teile des Strings farblich hervorheben wollen, können Sie ANSI-Farbcodes oder direct das Paket colorama verwenden.

Aufgabe 10.6 (Command Line Interface; 6 Punkte; Datei: `client.py`)

Um die Funktionalitäten unseres sozialen Netzwerks aus dem Terminal heraus nutzen zu können, wollen wir ein Command Line Interface (CLI) implementieren. Hierzu nutzen wir, dass bei der Ausführung von Python-Skripten externe Argumente angegeben werden können. Diese werden beim Aufruf des Skriptes mit Leerzeichen getrennt hinter dem Dateinamen anhängt werden. Rufen wir zum Beispiel ein Skript `foo.py` mit den externen Argumenten `arg1`, `arg2`, `arg` und `3` auf, so sieht der Aufruf wie folgt aus:

```
python3.12 foo.py arg1 arg2 arg 3
```

In der Datei `foo.py` können wir auf die externen Argumente über die `argv` Variable aus dem `sys` Modul zugreifen:

```
import sys
print(sys.argv) # ['./foo.py', 'arg1', 'arg2', 'arg', '3']
```

Fangen Sie die externen Argumente in einer `if __name__ == "__main__"` Verzweigung ab und implementieren Sie gleichnamige Hilfsfunktionen, die die Funktionalitäten des sozialen Netzwerks ausführen. Verwenden Sie dazu die Wrapper Funktionen aus Teilaufgabe 4, sowie die Funktion `yeet_to_str` aus Teilaufgabe 5 zum Konvertieren eines Yeets in einen String.

- `yeet`: Fordert die Benutzerin mit `input` auf, einen Text einzugeben, der im Anschluss geyeeted wird[7].
- `remove {yeet_id}`: Löscht den Yeet mit der ID `{yeet_id}`
- `like {yeet_id}`: Liked den Yeet mit der ID `{yeet_id}`
- `unlike {yeet_id}`: Entfernt einen selbst gesetzten Like von dem Yeet mit der ID `{yeet_id}`
- `liked {yeet_id}`: Zeigt an, wer den Yeet mit ID `{yeet_id}` geliked hat.
- `follow {user}`: Folgt der Benutzerin `{user}`
- `unfollow {user}`: Entfolgt der Benutzerin `{user}`

[7] Visualisierung des Hochladevorgangs: https://c.tenor.com/w96rgWmGS7gAAAAC/tenor.gif

- `replies {yeet_id}`: Zeigt alle *direkten* Antworten auf den Yeet mit der ID `{yeet_id}` an.
- `reply {yeet_id}`: Fordert die Benutzerin auf, mit `input` einen Text einzugeben, und diesen als Antwort auf den Yeet mit der ID `{yeet_id}` zu posten.
- `profile {user}`: Zeigt das Profil der Benutzerin `{user}` an (Yeets, Follower und Benutzerinnen, denen gefolgt wird)
- `latest {amount}`: Zeigt die neuesten `{amount}` Yeets.
- `feed`: Zeigt in zeitlicher Reihenfolge (sortieren!) alle Yeets der Personen an, denen Sie folgen.

Sie können sich bei Ihrer Implementierung an folgendem Beispiel orientieren (Den Code finden Sie auch im Template):

```
def like(session: Session, yeet_id: int):
    add_like(session, yeet_id)

if __name__ == "__main__":
    session = Session()
    authenticate(session)

    match sys.argv[1:]:
        case ['like', yeet_id]:
            like(session, int(yeet_id))
        ...
```

Aufgabe 10.7 (Erweiterungen; 10 Bonuspunkte; Datei: `client.py`)

Erweitern Sie ihr Command Line Interface um beliebige weitere externe Argumente. Hier ein paar Vorschläge:

- `search`: Fragt die Benutzerin nach einem Text und zeigt Yeets and, die diesen Text beinhalten.
- `date {day}`: Zeigt alle Yeets an, die am Tag `{day}` erstellt wurden.
- `tree {yeet_id}` Zeigt den Yeet mit der ID `yeet_id` in Baumansicht an. Das heißt alle Kommentare (auch Kommentare auf Kommentare) sind entsprechend ihrer Tiefe eingerückt.
- `replies`: Zeigt alle Antworten auf die eigenen Yeets an.

Wenn Sie besonders motiviert sind, können Sie beispielsweise auch ein Argument `tui` hinzufügen, das mit der `textualize`[8]-Library ein Terminal-User-Interface[9] bereitstellt. Wenn Ihr Programm zusätzliche Libraries verwendet, merken Sie dies bitte in der `NOTES.md` an.

[8] https://textual.textualize.io/
[9] https://en.wikipedia.org/wiki/Text-based_user_interface

Sie können sich hier beliebig austoben :). Damit wir wissen, was Sie gemacht haben, vermerken Sie die entsprechenden externen Argumente in die `NOTES.md` Datei!

Aufgabe 10.8 (Soziale Interaktionen; 0 Punkte; Datei)

Jetzt muss das Netzwerk nur noch gefüllt werden. Verfassen Sie gerne einen oder mehrere Yeets und kommentieren und liken Sie die Yeets Ihrer Kommilitoninnen. Bleiben Sie dabei höflich und respektvoll, Sie sind nicht anonym :)!

In diesem Sinne: Viel Spaß beim sozialen Netzwerken. Wir wünschen Ihnen frohe Weihnachten und einen guten Rutsch ins neue Jahr!

Aufgabe 10.9 (Erfahrungen; 0 Punkte; Datei: `NOTES.md`)

Notieren Sie Ihre Erfahrungen mit diesem Übungsblatt (benötigter Zeitaufwand, Probleme, Bezug zur Vorlesung, Interessantes, etc.).

Editieren Sie hierzu die Datei `NOTES.md` im Abgabeordner dieses Übungsblattes auf unserer Webplattform. Halten Sie sich an das dort vorgegebene Format, da wir den Zeitbedarf mit einem Python-Skript automatisch statistisch auswerten. Die Zeitangabe `7.5 h` steht dabei für 7 Stunden 30 Minuten.