To Do List API in Python mit Flask

Svea Wilkening, Tamina Adam, Markus Bacher, Georg Gohmann, Michael Herz

7. Juni 2024

Contents

1.0 Abstract

Dieses Programm entstand im Rahmen einer Projekt Gruppenarbeit. Die Aufgabenstellung beinhaltete das Programmieren einer API (“Application Programming Interface”) oder vereinfacht ausgedrückt einer Programmierschnittstelle.

Die exakte Aufgabe bestand darin eine To Do Listen API in der Programmiersprache Python zu schreiben und das ‘Flask’ Modul zur Erfüllung dieser Aufgabe zu verwenden.

Der Zeitrahmen war auf 2 1/2 Tage oder 100 Stunden Gesamt angesetzt. Weitere Anforderungen an die Applikation waren:

- Implementation der Vorgegebenen Endpunkte
- Härtung der Funktionen
- Fehlerbehandlung

Zur Optimalen Verwendung dieser Applikation empfehlen wir, die Postman App zum Abrufen und Manipulieren von Requests. Diese App ist kostenlos verfügbar unter https://www.postman.com/downloads.

Für die Optimale Darstellung empfehlen wir in der Postman App die Ausgabe in Body- raw.

Folgende Voraussetzungen müssen für die korrekte Funktionalität erfüllt sein: - Ein Python Interpreter oder eine entsprechende Entwicklungsumgebnung - PostmanApp oder BurpSuite - Installiertes ‘Flask’ Modul im verwendeten Python Interpreter - Installiertes ‘Datetime’ Modul im verwendeten Python Interpreter

Links zu allen in dieser Dokumentation erwähnten Programme und Hilfestellungen werden abschliesend im Appendix aufgeführt

Diese Applikation ist Open Source, dies bedeutet sowohl der Sourcecode als auch die Anwendung an sich sind öffentlich frei zugänglich und kostenlos verfügbar. Jedem steht es frei sich das entsprechende

GitHub Repository unter https://github.com/n3M3Z1Z/ToDo-List.git herunterladen und nach eigenen Bedürfnissen anpassen.

Der eigentliche Code in dem oben verlinkten Repository ist gründlich und ausführlich Kommentiert, dies dient der Vereinfachung und der Nachvollziehbarkeit in der Anwendung sowie der Erleichterung bei der Beseitigung Potentieller Fehler.

2.0 Allgemeine Anwendung

Im wesentlichen handelt es sicher in diser Anwendung um eine To Do Liste im JASON Format

Diese Applikation kann sowohl über eine Entwicklungsumgebung wie z.B. Visiual Studio Code oder über Python direkt verwendet werden.

Über Tools wie Postman oder BurpSuite kann nach erfolgreichem Start können nun unter der IP-Addresse 127.0.0.1:5000 die entsprechenden Endpunkte angesteuert und entsprechend ihrer Funktion bearbeitet werden.

Die IP 127.0.0.1 ist die Loopback Addresse welche einen PC direkt auf ihn selbst zurückverweist, der Teil :5000 der Addresse verweist auf den Port auf welchem das Programm läuft.

2.1 Überblick über die Allgemeine Funktionsweise

An dieser Stelle wird die Funktionsweise dieser Applikation kurz erläutert. Die Applikation wird im wesentlichen über die Eingabezeile des Browsers bzw. des verwendeten Darstellungsprogrammes gesteuert.

Um dies zu ermöglichen, wurden verschiedene Endpunkte erstellt und eingepflegt. Diese Endpunkte sind:

- home: der home Tab der Applikation gibt Anweisungen zur Benutzung des Programmes.
- tasks: dieser Tab gibt einen Überblick über alle hinterlegten Aufgaben.
- pending: dieser Tab zeigt alle Aufgaben an welche den Status ‘pending’ erfüllen-
- completed: dieser Tab zeigt alle Aufgaben an welche den Status ‘completed’ erfüllen.
- upload: hier können Dateien im JASON Format in die Applikation geladen werden.
- download: hier können hinterlegte Dateien im JASON Format heruntergeladen werden.
- endpoint_42: Der Ort zur ultimativen Entspannung.
- awesome: Der Endpunkt für absolut ‘awsome’ Pepole.

Verschiedene Funktionen und Endpunkte benötigen verschiedene Operationen um reibungslos und Vorgabenkonform zu funktionieren. Nachfolgend eine Auflistung der möglichen Operationen für diese Applikation:

- ‘GET’: wird verwendet um einen Endpunkt auf zu rufen.
- ‘POST’: wird verwendet um Inhalt der Applikation hinzu zu fügen.
- ‘PUT’: wird ähnlich angewendet wie die ‘POST’ Operation.
- ‘UPDATE’: wird verwendet um Aufgaben etc. zu aktualisieren.
- ‘DELETE’: wird Angewendet um einen Inhalt zu löschen.

3.0 Detaillierte Übersicht und Erklärung der Einzelnen Funktionen

In diesem Kapitel wird nun der gesamte Code in seine einzelnen Funktionen und Endpunkte aufgeteilt und eingehender erläutert.

Ausgespart werden hier lediglich die Möglichen Fehlermeldungen und deren Behandlung, diese werden in den Kapiteln **4.0** Potentielle Fehlermeldungen und **4.1** Fehlerbehandlung abgehandelt.

Die Eingangs erwähnten Kommentare werden hier ebenfalls nicht behandelt.

```python
# Import necessary modules, files & libraries
from flask import Flask, jsonify, request, redirect, url_for
import api_beispieldaten as ts
from datetime import datetime


# Define the API overall function
app = Flask(__name__)

# Import 'tasks' from 'tasks_templet.py'
tasks = ts.tasks
user_name = None

@app.before_request
def check_user_name():
    if request.endpoint not in ['hello_user', 'get_user_name'] and user_name == None:
        return redirect(url_for('hello_user'))
```

Figure 1 Benötigte Module und erste Funktionen

In dieser Abbildung ist zu sehen welche zusätzlichen Module zur korrekten Funktion benötigt werden und wie diese implementiert werden.

Die ‘app = Flask(**name**)’ definiert die Allgemeine Funktion der Applikation, welche die Instanz der ‘Flask’ Klasse erstellt.

Weiterführend muss an dieser Stelle die task liste in die API importiert werden, dies erledigt die funktion: ‘tasks = ts.tasks user_name = None’

Zur Personalisierung wurde eine Funktion eingeführt, welche den Anwender nach seinem Namen fragt und diesen in der Folge mit diesem anspricht.

```python
# ask for the user's name
@app.route("/user", methods=["POST"])
def get_user_name():

    data = request.get_json()
    if not data or 'user_name' not in data or not data['user_name']:
        return jsonify({"message": "Please provide a non-empty user name in the JSON payload with the key 'user_name'."}), 400

    global user_name
    user_name = "Zipfelklatscher"
    return jsonify({"message": f"Hallo {user_name}, schön dich kennenzulernen."})

        #return redirect(url_for('index'))

@app.route("/user", methods=["GET"])
def hello_user():
    return jsonify({"message": "Please set your user name by sending a POST request with JSON containing the key 'user_name' at the endpoint /user."})
```

Figure 2 weiterführender Code zur Personalisierung

```python
# set 'home' endpoint
@app.route("/home", methods=["GET"])
def home():
    home_content = {
        "message": f"Welcome to the ToDo List API {user_name}!",
        "actions": [
            "- create a new task",
            "- update an existing task",
            "- delete an existing task",
            "- mark a task as completed",
            "- filter or delete tasks by priority & status",
            "- upload a json file containing tasks in json file format",
            "- download a json file containing tasks in json file format",
        ],
        "endpoints": {
            "tasks": "/home/tasks" "- to display all tasks",
            "completed": "/home/completed" "- to display all completed tasks",
            "pending": "/home/pending" "- to display all pending tasks",
            "upload": "/home/upload" "- to upload a json file",
            "download": "/home/download" "- to download a json file",
            "endpoint_42": "/home/endpoint_42" "- the place to go if you need to relax ;)",
            "awsome": "/home/awsome" "- see something awsome",
        }
    }
    return jsonify(home_content), 200
"""
```

Figure 3 der erste Endpunkt /home

In *Figure 3* wird die Funktion und die Implementierung des home tabs dargestellt. Der Anfang eines jeden Endpunktes besteht aus der Definition der Route gefolgt von der Definition des Endpunktes und seines Inhaltes sowie gegebenenfalls der Definition der Erlauben Operationen, sowie der Reaktion auf Fehler.

```python
# Set endpoint to retrieve all tasks
@app.route("/home/tasks", methods=["GET"])
def get_tasks():
    if not tasks:
        return error_404()
    return jsonify(message=f"{user_name} look at your mess, way to many open tasks, get them done!",tasks=tasks)
```

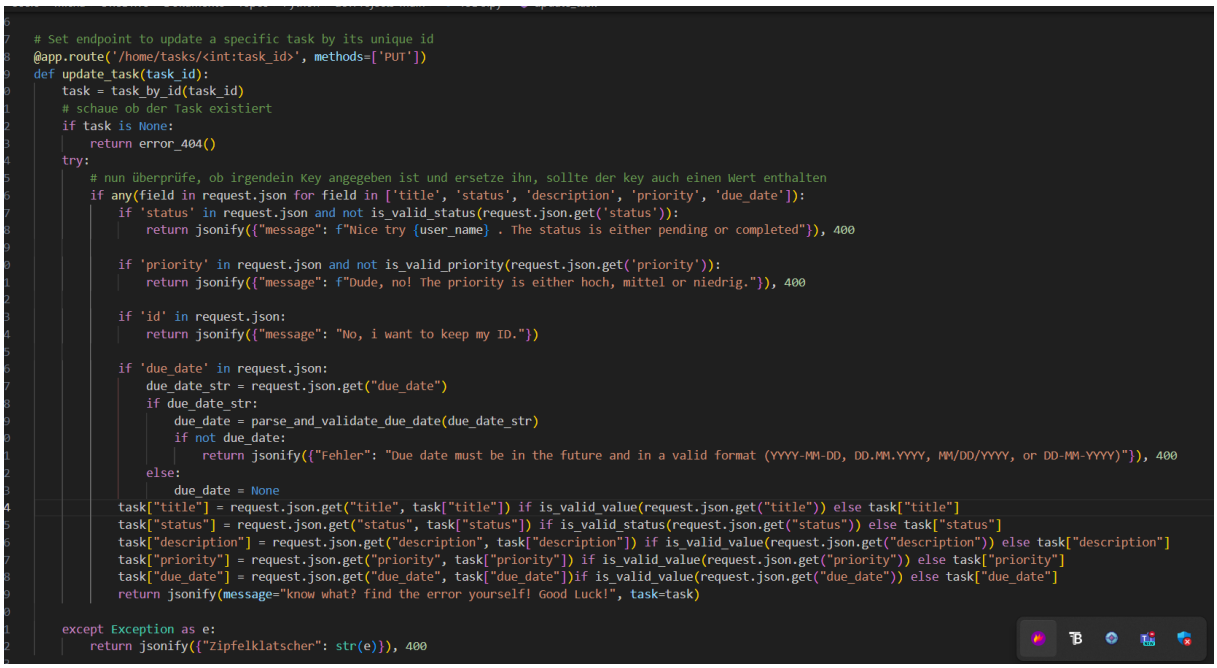
*Figure 4 Funktion zur Darstellung aller Aufgaben (tasks)

```python
# Define Endpoint to retrieve task by id
@app.route("/home/tasks/<int:task_id>", methods=["GET"])
def get_task(task_id):
    task = task_by_id(task_id)
    if task:
        return jsonify(message=f"Congrats {user_name} you just found yourself a task!", task=task)
    else:
        return error_404()

"""
    *set endpoint to create a new task*

    request JSON body:
    - title (string): Title of the task.
    - description (string): Description of the task.
    - priority (integer): Priority of the task.
    - due_date (string): Due date of the task.
"""

# Set endpoint to create a new task
@app.route('/home/tasks', methods=['POST'])
def create_task():
    try:
        request_json = request.get_json(force=True)
    except Exception as e:
        return jsonify({"Fehler": f"Failed to decode JSON object: {str(e)}"}), 400
```

Figure 5 Die Endpunkte zur Darstellung eines spezifisch Angeforderten tasks sowie zur Erstellung neuer taks

```python
    #if not request.json:
    #    return jsonify({"Fehler": "Request muss im JSON-Format sein"}), 400
    if 'title' not in request.json or 'description' not in request.json:
        return jsonify({"Fehler": "Es müssen sowohl der title, wie auch die description gesetzt sein. Sollstest du eigent wissen"}), 400

    due_date_str = request.json.get("due_date")
    if due_date_str:
        due_date = parse_and_validate_due_date(due_date_str)
        if not due_date:
            return jsonify({"Fehler": "Due date must be in the future and in a valid format (YYYY-MM-DD, DD.MM.YYYY, MM/DD/YYYY, or DD-MM-YYYY)"}), 400
    else:
        due_date = None


    if 'status' in request.json and not is_valid_status(request.json.get('status')):
            return jsonify({"message": "Nice try. The status is either pending or completed"}), 400

    if 'priority' in request.json and not is_valid_priority(request.json.get('priority')):
        return jsonify({"message": "Dude, no! The priority is either hoch, mittel or niedrig."}), 400
    try:
        new_task = {
            "id": tasks[-1]["id"] + 1,
            "title": request_json["title"],
            "description": request_json["description"],
            "status": request_json.get("status", "pending"), # Default value is pending
            "priority": request_json.get("priority", "niedrig"),
            "due_date": request_json.get("due_date", None)
        }
        tasks.append(new_task)
        return jsonify(message=f"oohpsie whoopsie - {user_name} you just created yourself more work!", new_task=new_task), 201

    except Exception as e:
        return jsonify({"Fehler": str(e)}), 400
```

Figure 6 der Restliche Abschnitt zur Erstellung neuer Aufgaben



```python
# Set endpoint to update a specific task by its unique id
@app.route('/home/tasks/<int:task_id>', methods=['PUT'])
def update_task(task_id):
    task = task_by_id(task_id)
    # schaue ob der Task existiert
    if task is None:
        return error_404()
    try:
        # nun überprüfe, ob irgendein Key angegeben ist und ersetze ihn, sollte der key auch einen Wert enthalten
        if any(field in request.json for field in ['title', 'status', 'description', 'priority', 'due_date']):
            if 'status' in request.json and not is_valid_status(request.json.get('status')):
                return jsonify({"message": f"Nice try {user_name} . The status is either pending or completed"}), 400

            if 'priority' in request.json and not is_valid_priority(request.json.get('priority')):
                return jsonify({"message": f"Dude, no! The priority is either hoch, mittel or niedrig."}), 400

            if 'id' in request.json:
                return jsonify({"message": "No, i want to keep my ID."})

            if 'due_date' in request.json:
                due_date_str = request.json.get("due_date")
                if due_date_str:
                    due_date = parse_and_validate_due_date(due_date_str)
                    if not due_date:
                        return jsonify({"Fehler": "Due date must be in the future and in a valid format (YYYY-MM-DD, DD.MM.YYYY, MM/DD/YYYY, or DD-MM-YYYY)"}), 400
                else:
                    due_date = None
            task["title"] = request.json.get("title", task["title"]) if is_valid_value(request.json.get("title")) else task["title"]
            task["status"] = request.json.get("status", task["status"]) if is_valid_status(request.json.get("status")) else task["status"]
            task["description"] = request.json.get("description", task["description"]) if is_valid_value(request.json.get("description")) else task["description"]
            task["priority"] = request.json.get("priority", task["priority"]) if is_valid_value(request.json.get("priority")) else task["priority"]
            task["due_date"] = request.json.get("due_date", task["due_date"])if is_valid_value(request.json.get("due_date")) else task["due_date"]
            return jsonify(message="know what? find the error yourself! Good Luck!", task=task)

    except Exception as e:
        return jsonify({"Zipfelklatscher": str(e)}), 400
```

Figure 7 Der Endpunkt und die Funktion um Aufgaben zu Aktualisieren

```python
# Set endpoint to delete task
@app.route('/home/tasks/<int:task_id>', methods=['DELETE'])
def delete_task(task_id):
    task = task_by_id(task_id)
    if task is None:
        return error_404()
    tasks.remove(task)
    return jsonify(message="System32 deleted - bye!", task=task)

"""
    *set endpont to mark a specific task as deleted*

    parameters:
    id (int): Unique identifier of the task.

    returns:
    JSON: deleted task
"""

# Set endpoint to mark a specific task
@app.route('/home/tasks/<int:task_id>/complete', methods=['PUT'])
def complete_task(task_id):
    task = task_by_id(task_id)
    if task is None:
        return error_404()
    task["status"] = "completed"
    return jsonify(Message=f"{user_name} you marked it you do it! I'll check it!", task=task)
```

Figure 8 Die Endpunkte um Aufgaben zu löschen oder spezifische Aufgaben zu ‘marken’

```python
# Set endpoint to retrieve all completed tasks
@app.route('/home/tasks/completed', methods=['GET'])
def get_completed_tasks():
    completed_tasks = [task for task in tasks if task["status"] == "completed"]
    if not completed_tasks:
        return error_404()
    return jsonify(message=f"You are a very good boy {user_name}!", completed_tasks=completed_tasks)

"""
    *set endpoint to retieve all pending tasks*

    returns:
    JSON: a list of all pending tasks.
"""

# Set endpoint to retrieve all pending tasks
@app.route('/home/tasks/pending', methods=['GET'])
def get_pending_tasks():
    pending_tasks = [task for task in tasks if task["status"] == "pending"]
    if not pending_tasks:
        return error_404()
    return jsonify(message="Do or do not - there is no pending!", pending_tasks=pending_tasks)
```

Figure 9 Endpunkte um alle Aufgaben der Kategorien 'pending' & 'completed' abzurufen

```python
# Set endpoint to retrieve tasks by priority level
@app.route('/home/tasks/priority/<string:level>', methods=['GET'])
def get_tasks_by_proirity_level(level):
    level_list = ["hoch", "niedrig", "mittel"]
    if level in level_list:
        priority_tasks = [task for task in tasks if task["priority"] == level]
        return jsonify(message=f"now look what you did {user_name} or what you have to to dosen't really matter to me", priority_tasks=priority_tasks)
    else:
        return error_400()

"""
    *set endpoint to delete all completed tasks*

    retrurns:
    JSON: a list of deleted completed tasks.
"""

# Set endpoint to delete all completed tasks
@app.route('/home/tasks/completed', methods=['DELETE'])
def delete_completed_tasks():
    completed_tasks = [task for task in tasks if task["status"] == "completed"]
    if not completed_tasks:
        return error_404()
    for task in completed_tasks:
        tasks.remove(task)
    return jsonify(message=f"say bye bye files {user_name} bye bye", completed_tasks=completed_tasks)

"""
```

Figure 10 Endpunkt zur Auflistung aller tasks nach Priorität & Endpunkt zur Entfernung aller tasks der Kategorie 'completed'

```python
# Set endpoint to delete all pending tasks
@app.route('/home/tasks/pending', methods=['DELETE'])
def delete_pending_tasks():
    pending_tasks = [task for task in tasks if task["status"] == "pending"]
    # wenn keine Tasks in pending vorliegen
    if not pending_tasks:
        return error_404()
    for task in pending_tasks:
        tasks.remove(task)
    return jsonify(message=f"say bye bye files {user_name} bye bye!", pending_tasks=pending_tasks)
```

Figure 11 Endpunkt zur Entfernung aller tasks der Kategorie ‘pending’


```python
# set and define endpoint_42 and print ASCII art for most error 400
@app.route('/home/endpoint_42', methods=['GET'])
def function_42():
    ascii_art = """
                                  M
                                  dM
                                  MMr
                                 4MMML                  .
                                 MMMMM.                xf
                                "MMMMM                .MM-
    Mh..                       +MMMMMM             .MMMM
    .MMM.                      .MMMMML.           MMMMMh
     )MMMh.                     MMMMMM           MMMMMMM
      3MMMMx.                  'MMMMMMf       xnMMMMMM"
      '*MMMMM                   MMMMMM.      nMMMMMMP"
        *MMMMMx                 "MMMMM\\     .MMMMMMM=
         *MMMMMh                "MMMMM"    JMMMMMMP
           MMMMMM                3MMMM.   dMMMMMM            .
            MMMMMM              "MMMM   .MMMMM(         .nnMP"
              *MMMMx            MMM"   dMMMM"     .nnMMMMM*
    "MMn...      'MMMMr 'MM    MMM"    .nMMMMMMM*"
     "4MMMMnn..    *MMM  MM   MMP"   .dMMMMMMM""
      ^MMMMMMMMx.   *ML "M .M*   .MMMMMM**"
         *PMMMMMMhn. *x > M   .MMMM**""
           ""**MMMMhx/.h/ .=*"
                    .3P"%....
                  nP"     "*MMnx
    """
    return ascii_art
```

Figure 12 Endpunkt 42

```python
# Set 'main' function to run the app
if __name__ == "__main__":
    app.run(debug=True)
```

Figure 13 die ‘main’ Funktion

Die in *Figure 13* abgebildete ‘main’ Funktion ist die Funktion, welche das Programm startet, wenn alle Voraussetzungen erfüllt sind

4.0 Potentielle Fehlermeldungen

Wie bei allen Anwendungen können auch bei der Verwendung dieser Applikation Fehlermeldungen auftreten. Diese können entweder versehentlich auftreten oder auch bewusst hervorgerufen werden.

An dieser Stelle daher eine Übersicht über die gängigsten Fehlermeldungen und ihre Bedeutung:

- Error 400 - Die Anfrage wurde nicht verstanden
- Error 404 - Die angeforderte Datei/ URL existiert nicht
- Error 405 - Die verwendete Methode ist nicht erlaubt

Weitere Fehlermeldungen:

- Syntax Error: Das Eingabe Format ist fehlerhaft
- Date Time Error. Das Datum ist nicht valide

4.1 Fehlerbehandlung

Um diese Fehler aufzufangen, wurden entsprechende Maßnahmen implementiert. Diese Maßnahmen umfassen sowohl die Erkennung als auch den Umgang mit diesen Fehlern.

So wurden unteranderem jeder Funktion wie i*Figure 1* bis *Figure 11* ersichtlich entsprechende Parameter mitgebeben, welche zwingend erfüllt sein müssen.

Zusätzlich dazu wurden drei Globale Errorhandler definiert und eingepflegt diese werden nachfolgend in *Figure 14* abgebildet.

```python
# Function to handle 400 Bad Request error
@app.errorhandler(400)
def bad_request(error):
    return error_400()

# Function to handle 404 Not Found error
@app.errorhandler(404)
def page_not_found(error):
    return error_404()

# Function to handle 405 Method Not Allowed error
@app.errorhandler(405)
def method_not_allowed(error):
    return error_405()
```

Figure 14 Globale Errorhandler für die Error Meldungen: 400, 404, 405

Einige der Fehlermeldungen werden über ‘jsonify’ individualisiert zurückgegeben, andere rufen die in *Figure 14* abgebildeten Funktionen auf und geben dem Anwender diese zurücl.

Diese Funktionen werden nun in diesem Kapitel abgebildet.

```
# define function to call ascii artwork for error 400  - Bad Request (at least some of them)
def error_400():
    error_400 = """
     .'/,-Y"     "~-.
 l.Y             ^.
 /\               _\_      "DOH!"
i              ___/"   "\
|          /"   "\   o !
l         ]     o !__./
 \ _  _    \.___./    "~\
  X \/ \             ___./
 ( \ ___.   _.--~~"   ~`-.
  ` Z,--   /               \
    \__.  (   /        ______)
      \   l  /-----~~" /      Error 400, Bad Request
       Y   \          /
       |    "x______.^  Try again, {user_name}
       |           \
       j            Y
"""
    return error_400
```

Figure 15 die Funktion ‘error_400’ zur Behandlung des gleichnamigen Errors

```python
# define function to call ascii artwork for error 404 - Page not found
def error_404():
    error_404 = f""" .-------..
    | -          -
   /             \\
  /                \\
 /    .--._    .---.   |
|  /      -__-     \\   |
| |                 |  |
||      ._   _.     ||
||       o   o      ||
||       _  |_      ||
C|      (o\\_/o)     |O     Uhhh, this computer
 \\       _____      /       is like, busted or
   \\ ( /#####\\ ) /        something {user_name}. So go away.
    \\  `====='  /
     | \\  -___-  /
     |  |       |           Error 404, page not found.
     | /-_____-\\
     /           \\
    /             \\
   /__|  AC / DC  |__\\
   | ||           |\\ \\
"""
    return error_404
```

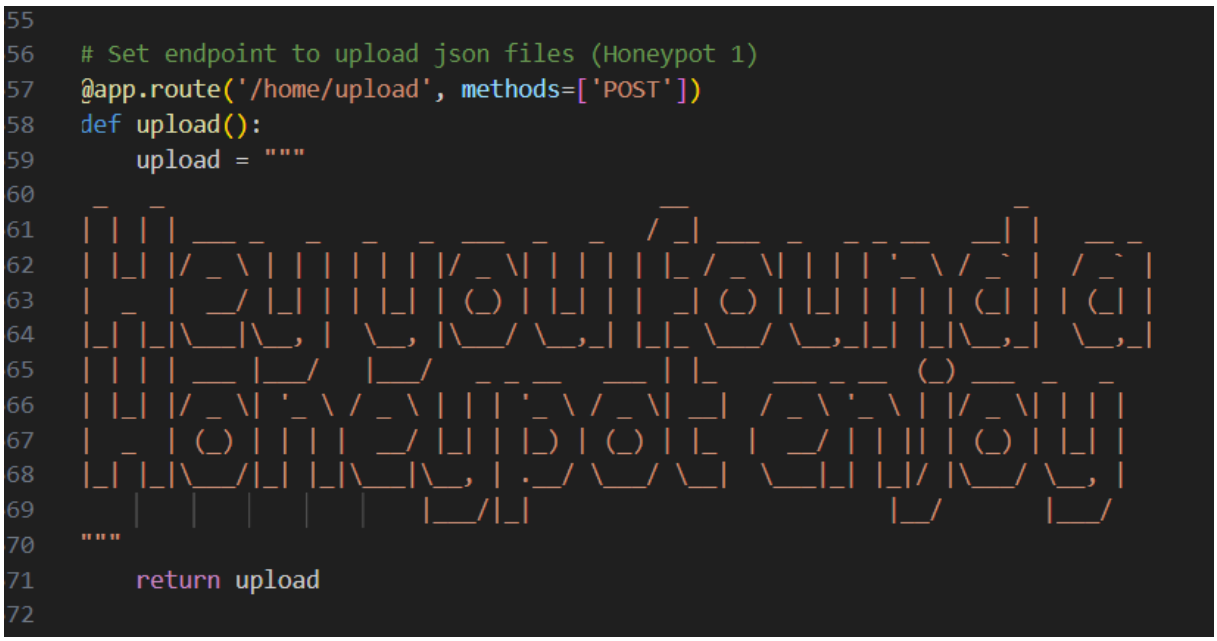
Figure 16 die Funktion ‘error_404’ zur Behandlung des gleichnamigen Errors

```
295
296  # define function to call ascii artwork for error 405 - Method Not Allowed
297  def error_405():
298      error_405 = f"""
299       |_____________|_
300      ||--------|| | _________
301      ||- _     || |(Bye Bye )
302      ||    - _ || | ---------
303      ||       -|| |     //
304      ||        || o\    __
305      ||        ||  \\  (..)
306      ||        ||   \\_|  |_
307      ||        ||    \  \/  )
308      ||        ||     :    :|
309      ||        ||     :    :|
310      ||        ||     :====:o
311      ||        ||     (    )
312      ||__@@@@__||     | `' |
313      || @|..|@ ||     | || |
314      ||o@`=='@o||     | || |
315      ||_@\/\/@_||     |_||_|
316       ---------------   '_'`_`
317  /_________________\----------\
318  |   GUILLOTINE   |-----------|
319   Error 405, Method Not Allowed
320  |_____________________________|
321  You are not allowed to do that, {user_name}
322  """
323      return error_405
```

Figure 17 die Funktion 'error_405' zur Behandlung des gleichnamigen Errors

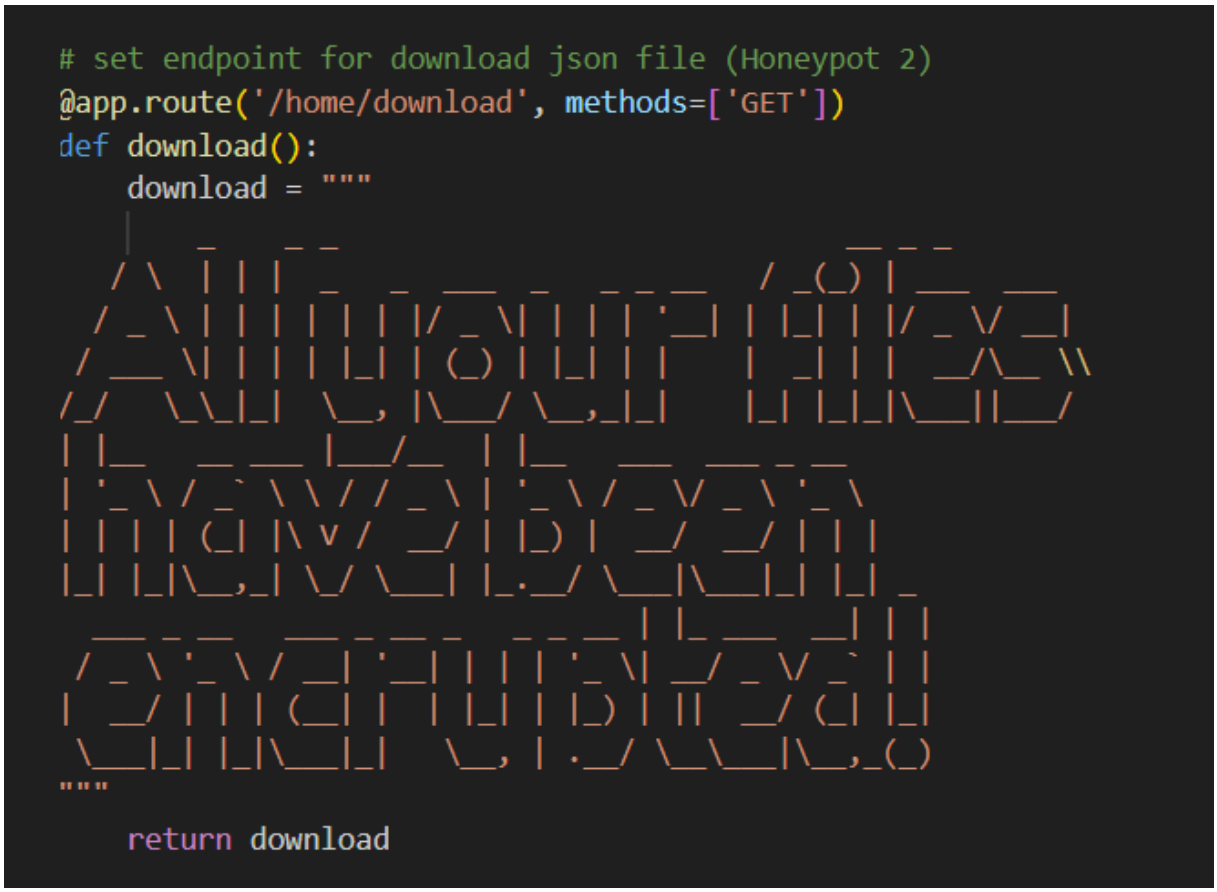
5.0 "Honeypots"

Wie einigen Lesern eventuell bereits Aufgefallen sein dürfte, wurden die Endpunkte upload und download bisher nicht abgebildet. Dies liegt in der Tatsache begründet, dass es hier um sogenannte "Honeypots" handelt, also Funktionen welche einen potentiellen Angreifer dazu verleiten sollen unvorsichtig zu sein und sich eventuell selbst zu verraten. In diesem Falle sind sie jedoch lediglich eine kleiner Scherz um unseren Dozenten zu erheitern.



```
55
56  # Set endpoint to upload json files (Honeypot 1)
57  @app.route('/home/upload', methods=['POST'])
58  def upload():
59      upload = """
60
61  | | | | ___ _   _   _   _ ___ _   _   / _| ___ _   _ _ __   __| |
62  | |_| |/ _ \ | | | | | | |/ _ \| | | | |_ / _ \| | | | '_ \ / _` |  / _` |
63  |  _  |  __/ |_| | | |_| | (_) | |_| | |  _| (_) | |_| | | | | (_| | | (_| |
64  |_| |_|\___|\__, |  \__, |\___/ \__,_| |_|  \___/ \__,_|_| |_|\__,_|  \__,_|
65  | | | | ___ |___/   |___/    _ _ __   ___ | |_    ___ _ __   (_) ___   _   _
66  | |_| |/ _ \| '_ \ / _ \ | | | '_ \ / _ \| __|  / _ \ '_ \  | |/ _ \ | | | |
67  |  _  | (_) | | | |  __/ |_| | |_) | (_) | |_  |  __/ | | | | | (_) | |_| |
68  |_| |_|\___/|_| |_|\___|\__, | .__/ \___/ \__|  \___|_| |_|/ |\___/ \__, |
69                          |___/|_|                          |__/       |___/
70  """
71      return upload
72
```

Figure 18 die ‘upload’ Funktion



```
# set endpoint for download json file (Honeypot 2)
@app.route('/home/download', methods=['GET'])
def download():
    download = """
     _    _ _                                  __ _ _
    / \  | | |  _   _  ___  _   _ _ __      / _(_) | ___  ___
   / _ \ | | | | | | |/ _ \| | | | '__|    | |_| | |/ _ \/ __|
  / ___ \| | | | |_| | (_) | |_| | |       |  _| | |  __/\__ \
 /_/   \_\_|_|  \__, |\___/ \__,_|_|       |_| |_|_|\___||___/
 | |__   __ ___ |___/__   | |__   ___  ___ _ __
 | '_ \ / _` \ \ / / _ \ | '_ \ / _ \/ _ \ '_ \
 | | | | (_| |\ V /  __/ | |_) |  __/  __/ | | |
 |_| |_|\__,_| \_/ \___| |_.__/ \___|\___|_| |_| _
   ___ _ __   ___ _ __ _   _ _ __ | |_ ___   __| | |
  / _ \ '_ \ / __| '__| | | | '_ \| __/ _ \ / _` | |
 |  __/ | | | (__| |  | |_| | |_) | ||  __/ (_| |_|
  \___|_| |_|\___|_|   \__, | .__/ \__\___|\__,_(_)
 """
     return download
```

Figure 19 die ‘download’ Funktion

Auf die tatsächliche Implementierung des automatisierten Downloads von Ranosmware wurde schweren Herzens aus Rücksicht auf den Anwender verzichtet. (Oder doch nicht?)

6.0 Appendix

An dieser Stelle sei erwähnt das sämtliche in dieser Dokumentation verwendeten Bilder (*Figure 1 - Figure 21*) sowie Tabellen (*Table 1*)von den Autoren selbst angefertigt wurden.

Verweise:

Anwendung	Link
Visiual Studio Code	https://code.visualstudio.com/download
Postman	https://www.postman.com/downloads
ASCII- Art	https://www.asciiart.eu/
GitHub Repository	https://github.com/n3M3Z1Z/ToDo-List.git
Python	https://www.python.org/downloads/

Table 1 Verwiese zu den verwendeten/ empfohlenen Anwendungen und dem GitHub Repository

Nun möchten wir nun noch den bisher undokumentierten Endpunkt ‘awesome’ vorstellen.

```python
# set an 'awsome' endpoint
@app.route('/home/awesome')
def awesome():
    awesome = """
       _nnnn_
      dGGGGMMb     ,""""""""""""""".
     @p~qp~~qMb    | Linux Rules! |
     M|@||@) M|   _;..............'
     @,----.JM| -'
    JS^\__/  qKL
   dZP        qKRb    and so do this awesome pepole that created this app!
  dZP          qKKb
 fZP            SMMb      Svea Kristina Wilkening
 HZM            MMMM          Tamina Adam
 FqM            MMMM              Markus Bacher
__| ".        |\dS"qML                 Georg Gohmann
|    `.       | `' \Zq                      Michael Herz
_)     \.___.,|     .'
\____   )MMMMM|   .' And last but not least a big thank you to our
     `-'       `--'   awsome victim eehhm i mean teacher 'Andreas Thomas Kellerer'
"""
    return awesome
```

Figure 20 Der Endpunkt für wirklich 'awesome' pepole

Um diese Dokumentation abzuschließen möchten sich die Autoren Herzlich bei ihrem Dozenten Andreas Kellerer für den guten Unterricht, sowie für das entgegengebrachte Vertrauen das wir sein System bei einem Test nicht frittieren bedanken. Ohne seine Arbeit wäre dieses Projekt so nicht möglich gewesen.

Aber da wir ihn nun schon einige Tage nicht gesehen haben und ihn nicht erreichen können um ihm persönlich zu Danken hier noch ein kleiner Suchaufruf.

Figure 21 Wer hat unseren Dozenten gesehen?