NSQL – Flask + MongoDB Helpdesk

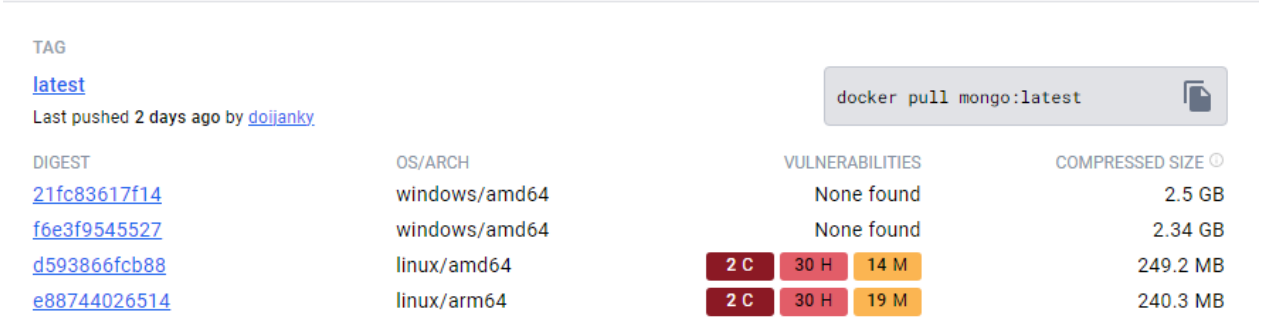
Zadání:

Na přednáškách jsme se dozvěděli, jak fungují technologie jako např. Flask, Docker, docker-compose, nebo šablonovací systém Jinja. Poté jsme si sami vyzkoušeli tyto technologie s pomocí aplikovat. Cílem úkolu bylo zpracovat webovou aplikaci za použití NoSQL DBMS.

Řešení:

Úkolem bylo sestrojit funkční webovou aplikaci, která bude využívat technologií z přednášek. Začal jsem tedy s analýzou souborů pro docker, které jsou v tomto projektu 2. První je docker-compose.yml, který se využívá pro orchestraci služeb, zároveň umožňuje spustit vícenásobné kontejnery. V našem případě v něm specifikujeme využité služby (flask, mongodb, zkoušel jsem i redis, nakonec jsem jej ale neaplikoval), spolu s jejich atributy jako je tagovatelná OS image (čerpáme z dockerhubu), port, závislost na jiných službách.

```yaml
version: '3'
services:
  flask:
    build: .
    container_name: flask
    ports:
      - "5000:5000"
    volumes:
      - ./code:/code
    depends_on:
      - mongodb

  mongodb:
    image: mongo:latest
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin
      MONGO_INITDB_ROOT_PASSWORD: admin
    ports:
      - 27017:27017
```



Pomocí dockerfile vytvoříme kontejner, a můžeme provádět operace jako instalace požadovaných balíčků, spuštění vlastní aplikace.

HTML:

Následně jsem začal s tvořením html stránek, kde se mi bohužel z mně neznámého důvodu nepovedlo za pomocí Flask.get_flashed_messages(with_categories=true) nepovedlo přiřadit zprávám barvu podle přiřazené kategorie (success, error)

```
<main class="p-5 text-center bg-light text-dark">    <!--
    {% with messages = get_flashed_messages(with_categories=true) %}
        {% if messages %}
            <ul class=flashes>
            {% for category, message in messages %}
                <p class="{{ category }}" >{{ message }}</p>
            {% endfor %}
            </ul>
        {% endif %}
    {% endwith %}
    -->
```

Python:

V pythonu jsem začal nastavením řetězce pro připojení k MongoDB a nastavením kolekcí. Dále kvůli množícím se datům při každé iteraci nejprve smažu obsah kolekcí a poté je znovu vytvořím. Následuje inicializace Flask.

```
mongo_client = pymongo.MongoClient( host: "mongodb://admin:admin@mongodb:27017", connect=False)
db = mongo_client['helpdesk']
task_collection = db['tasklist']
users_collection = db['helpdeskuser']

task_collection.drop()
users_collection.drop()

tasks_init = [
    {'id': '6', 'task': 'Nefunguje monitor', 'requestor': 'Skladnik2', 'assignee': 'IT1', 'status': 'New',
     'due_date': '2024-02-14'}
    , {'id': '5', 'task': 'Potrebuji pridat report', 'requestor': 'Manager1', 'assignee': 'ERP1',
      'status': 'In Progress', 'due_date': '2024-02-17'}
    , {'id': '4', 'task': 'pomoc excel', 'requestor': 'Acct', 'assignee': 'IT1', 'status': 'Completed',
      'due_date': '2024-02-05'}
    , {'id': '3', 'task': 'n€funguj€ mi €', 'requestor': 'QE2', 'assignee': 'IT2', 'status': 'Completed',
      'due_date': '2024-02-04'}
    , {'id': '2', 'task': 'neumim zapnout pc', 'requestor': 'Skladnik1', 'assignee': 'IT2', 'status': 'Completed',
      'due_date': '2024-02-01'}
    , {'id': '1', 'task': 'ahoj', 'requestor': 'QE1', 'assignee': 'ERP1', 'status': 'Completed',
      'due_date': '2024-02-01'}
]

users_init = [
    {'userid': '1', 'username': 'ERP1', 'password': 'helpdesk'}
    , {'userid': '2', 'username': 'IT1', 'password': 'helpdesk'}
    , {'userid': '3', 'username': 'IT2', 'password': 'helpdesk'}
]

task_collection.insert_many(tasks_init)
users_collection.insert_many(users_init)

app = Flask('__name__')
app.secret_key = "super secret key"
```

Následuje @app.route(‚/‘) a @app.route(‚/index‘) – což nám říká, že dekorátor @app.route přiřazuje URL ‚mujserver:port/‘ a ‚mujserver:port/index‘ funkci index(). Ta po jejím zavolání do proměnné tasks uloží všechny tasky z kolekce tasklist, seřazené sestupně podle id (pseudo). Následně mi vrátí všechny tasky uložené v připojené kolekci. Poté si vrátíme nově vytvořený html dokument. Pomocí proměnné tasklist zde předáváme parametry do html.

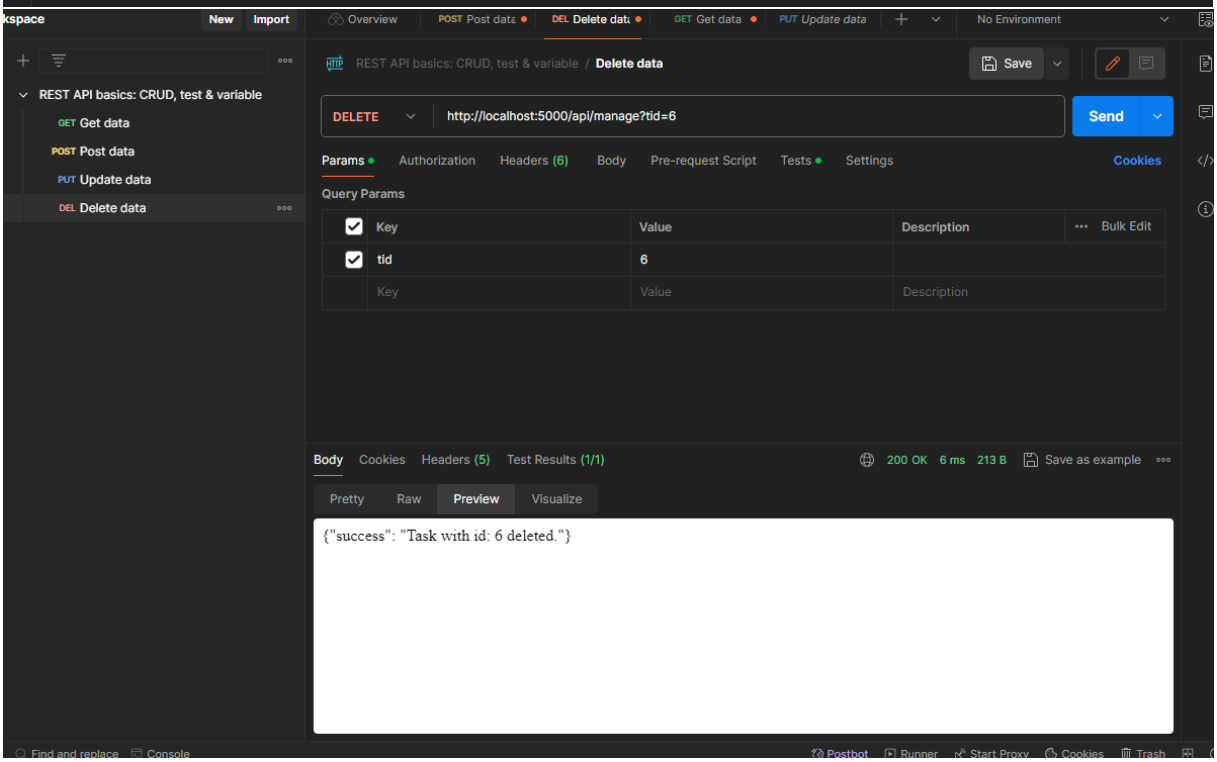
```python
@app.route('/')
@app.route('/index')
def index():
    tasks = task_collection.find(sort=[('id', pymongo.DESCENDING)])
    return render_template( template_name_or_list: 'index.html', tasklist=tasks)

```

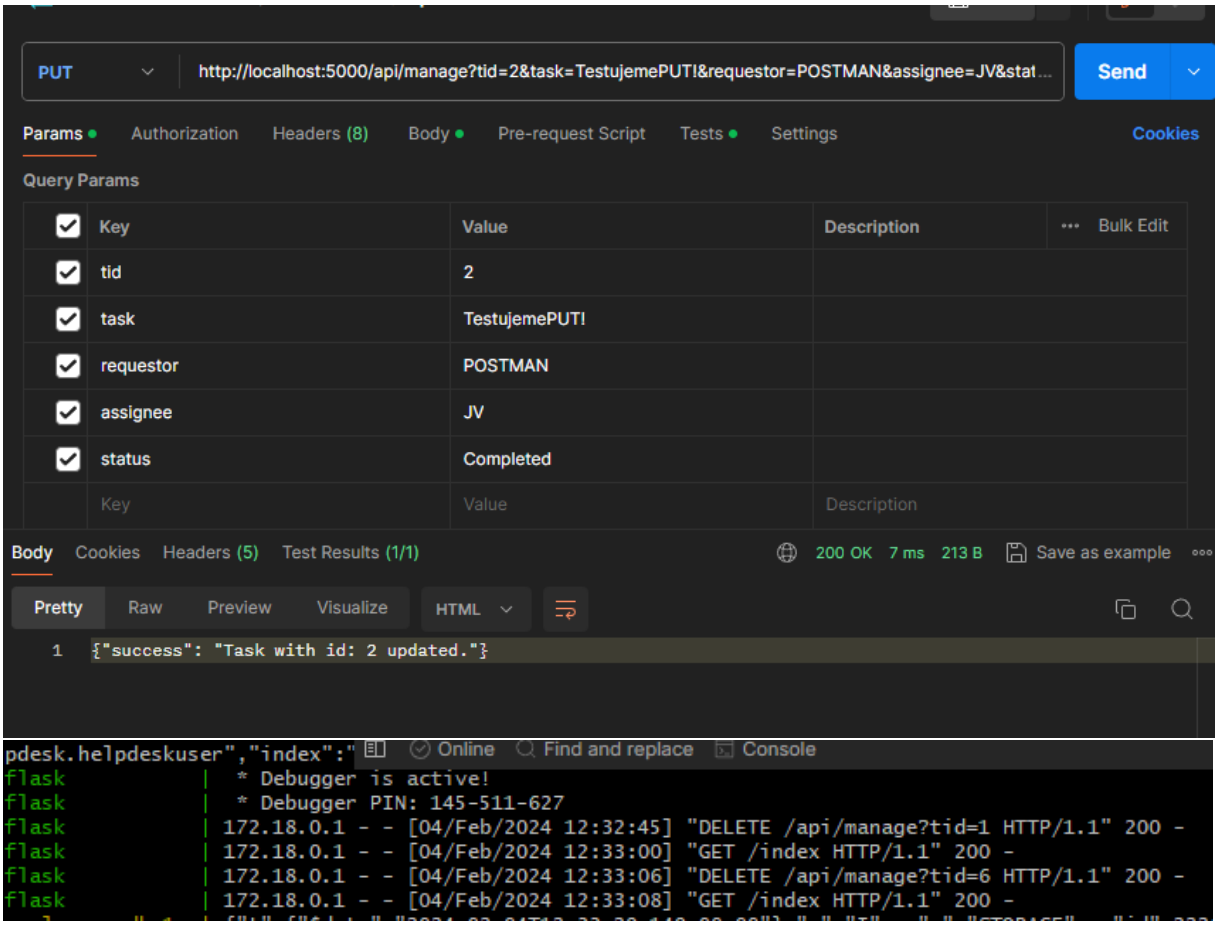
```
{% extends "template.html" %}

{% block content %}
<h2>Tasks</h2>
<ul class="list-group">
    <p>{% if session.user_is_authenticated %}
            ID -
        {% endif %}
        Task - Requestor - Assignee - Status - Due Date</p>
    {% for task in tasklist %}
    <li class="list-group-item">
        {% if session.user_is_authenticated %}
            {{ task["id"] }} -
        {% endif %}
         {{ task["task"] }} - {{ task["requestor"] }} - {{ task["assignee"] }} - {{ task["status"] }} - {{ task["due_date"] }}
    </li>
    {% endfor %}
</ul>
{% endblock %}
```

API:

Pro práci s MongoDB spolu s metodami PUT a DELETE jsem vytvořil API, které přijme jako adresu odkazu v mém případě http://localhost:5000/api/manage?{parametry}={klíče} a následně pomocí parametrů a jejich klíčů umožňují vymazat, či editovat záznam v Mongo. Tyto API jsem vyzkoušel přes Postman, kde jssem sledoval, s jakým výsledkem moje volání proběhlo, dále jsem kontroloval metody a odpovědi z backendu v příkazové řádce.

```python
J-Viola
@app.route( rule: '/api/manage', methods=['DELETE', 'PUT'])
def api_manage():
    if request.method == 'DELETE':
        ticket_id = request.args.get('tid')
        if task_collection.find_one({'id': ticket_id}):
            task_collection.delete_one({'id': ticket_id})
            return make_response( *args: json_util.dumps({'success': f'Task with id: {ticket_id} deleted.'}), 200)
        else:
            return make_response( *args: json_util.dumps({'error': 'Task not found!'}), 404)

    elif request.method == 'PUT':
        ticket_id = request.args.get('tid')
        updated_fields = {
            'task': request.args.get('task'),
            'requestor': request.args.get('requestor'),
            'assignee': request.args.get('assignee'),
            'status': request.args.get('status'),
        }
        if task_collection.find_one({'id': ticket_id}):
            task_collection.update_one( filter: {'id': ticket_id}, update: {'$set': updated_fields})
            return make_response( *args: json_util.dumps({'success': f'Task with id: {ticket_id} updated.'}), 200)
        else:
            return make_response( *args: json_util.dumps({'error': 'Task not found!'}), 404)
```




PUT
http://localhost:5000/api/manage?tid=2&task=TestujemePUT!&requestor=POSTMAN&assignee=JV&stat...
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
tid
2
task
TestujemePUT!
requestor
POSTMAN
assignee
JV
status
Completed
Key
Value
Description
Body
Cookies
Headers (5)
Test Results (1/1)
200 OK
7 ms
213 B
Save as example
Pretty
Raw
Preview
Visualize
HTML
1 {"success": "Task with id: 2 updated."}

pdesk.helpdeskuser","index":"
Online
Find and replace
Console
flask | * Debugger is active!
flask | * Debugger PIN: 145-511-627
flask | 172.18.0.1 - - [04/Feb/2024 12:32:45] "DELETE /api/manage?tid=1 HTTP/1.1" 200 -
flask | 172.18.0.1 - - [04/Feb/2024 12:33:00] "GET /index HTTP/1.1" 200 -
flask | 172.18.0.1 - - [04/Feb/2024 12:33:06] "DELETE /api/manage?tid=6 HTTP/1.1" 200 -
flask | 172.18.0.1 - - [04/Feb/2024 12:33:08] "GET /index HTTP/1.1" 200 -