Министерство науки и высшего образования РФ
ФГАОУ ВПО
Национальный исследовательский технологический университет «МИСИС»

Институт компьютерных наук (ИКН)

Кафедра Инфокоммуникационных технологий (ИКТ)

Отчет по лабораторной работе №8
по дисциплине «Разработка сетевых приложений
на языке программирования Python»
на тему «Работа с базой данных и реализация CRUD в Flask с использованием SQLAlchemy ORM»

Выполнил:
студент группы БИСТ-22-1

Медведев Д. Р.

Проверил:
доц. каф. ИКТ

Стучилин В.В.

Москва, 2025

Цель работы: освоить принципы работы с базой данных в веб-приложении на Flask.

Научиться:

- подключать и конфигурировать базу данных;
- создавать модели данных через SQLAlchemy;
- выполнять CRUD-операции (Create, Read, Update, Delete);
- сохранять данные между запусками сервера.

Задание:

1. Настроить Flask-приложение с SQLAlchemy и SQLite-базой данных tasks.db.
2. Создать файл models.py с моделью Task с полями:
   - id: целое число, первичный ключ;
   - title: строка, обязательное.
3. Модифицировать методы для работы с данными в рамках внешней БД (использовать проект ЛР7):
   - GET /tasks — получить список задач;
   - POST /tasks — добавить задачу;
   - GET /tasks/<id> — получить задачу по ID;
   - DELETE /tasks/<id> — удалить задачу.
4. Реализовать маршрут для обновления задач:
   - PUT /tasks/<id> — обновить задачу
5. Перезапустить сервис и получить все существующие задачи и информацию по ним.

Дополнительное задание (без защиты)

1. Выполнить лабораторную работу в структуре Task из ЛР7:
   - description;
   - status с возможными значениями: "pending" (по умолчанию), "done".

2. Реализовать:
   - GET /tasks/search?q=слово — поиск по заголовку
   - GET /tasks?sort=created_at — сортировка по дате
3. Создать файл init_db.py для инициализации базы и добавления тестовых данных
4. Написать автотест (на requests) для проверки CRUD-операций: добавить → обновить → удалить задачу → проверить результат

Решение

App.py

```python
from flask import Flask, jsonify
from views import tasks_blueprint
from models import db

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///tasks.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db.init_app(app)

app.register_blueprint(tasks_blueprint)

@app.errorhandler(404)
def not_found(error):
    return jsonify({"error": "Not found"}), 404

if __name__ == '__main__':
    with app.app_context():
        db.create_all()
    app.run(debug=True)
```

Data.py

```python
from models import db, Task
from typing import Dict, List, Optional

def get_all_tasks(status_filter: Optional[str] = None) -> List[Dict]:
    query = Task.query
    if status_filter:
        query = query.filter_by(status=status_filter)
    return [task.to_dict() for task in query.all()]

def get_task_by_id(task_id: int) -> Optional[Dict]:
    task = Task.query.get(task_id)
    return task.to_dict() if task else None

def add_task(title: str, description: Optional[str] = None, status: str = "pending") -> Dict:
    if not title:
        raise ValueError("Title cannot be empty")
    task = Task(title=title, description=description, status=status)
    db.session.add(task)
    db.session.commit()
    return task.to_dict()

def update_task(task_id: int, title: Optional[str] = None,
                description: Optional[str] = None, status: Optional[str] = None) -> Optional[Dict]:
    task = Task.query.get(task_id)
    if not task:
        return None

    if title is not None:
        if not title:
            raise ValueError("Title cannot be empty")
        task.title = title
    if description is not None:
        task.description = description
    if status is not None:
        task.status = status

    db.session.commit()
    return task.to_dict()

def delete_task(task_id: int) -> bool:
    task = Task.query.get(task_id)
    if not task:
        return False
    db.session.delete(task)
    db.session.commit()
    return True

def get_task_stats() -> Dict[str, int]:
    stats = {"pending": 0, "done": 0}
    for status in stats.keys():
        stats[status] = Task.query.filter_by(status=status).count()
    return stats
```

Init_db.py

```python
from app import app
from models import db, Task

def init_db():
    with app.app_context():
        db.create_all()

        # Добавляем тестовые данные, если таблица пуста
        if not Task.query.first():
            tasks = [
                Task(title='Task 1', description='description'),
                Task(title='Task 2', status='pending'),
                Task(title='Task 3', status='done')
            ]
            db.session.add_all(tasks)
            db.session.commit()
            print("Тестовые данные добавлены")

if __name__ == '__main__':
    init_db()
```

Models.py

```python
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime

db = SQLAlchemy()

class Task(db.Model):
    __tablename__ = 'tasks'

    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    description = db.Column(db.String(500))
    status = db.Column(db.String(20), default='pending')
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    updated_at = db.Column(db.DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)

    def to_dict(self):
        return {
            "id": self.id,
            "title": self.title,
            "description": self.description,
            "status": self.status,
            "created_at": self.created_at.isoformat(),
            "updated_at": self.updated_at.isoformat()
        }
```

Test_tasks.py

```python
import requests
import pytest

BASE_URL = 'http://localhost:5000'

def test_crud_operations():
    # Создание задачи
    new_task = {
        'title': 'New task',
        'description': 'New description',
        'status': 'pending'
    }
    create_response = requests.post(f'{BASE_URL}/tasks', json=new_task)
    assert create_response.status_code == 201
    task_data = create_response.json()
    task_id = task_data['id']

    # Проверка создания
    get_response = requests.get(f'{BASE_URL}/tasks/{task_id}')
    assert get_response.status_code == 200
    assert get_response.json()['title'] == new_task['title']

    # Обновление задачи
    updated_data = {
        'title': 'Updated',
        'status': 'done'
    }
    update_response = requests.put(f'{BASE_URL}/tasks/{task_id}', json=updated_data)
    assert update_response.status_code == 200
    assert update_response.json()['title'] == updated_data['title']

    # Удаление задачи
    delete_response = requests.delete(f'{BASE_URL}/tasks/{task_id}')
    assert delete_response.status_code == 200

    # Проверка удаления
    get_deleted_response = requests.get(f'{BASE_URL}/tasks/{task_id}')
    assert get_deleted_response.status_code == 404

def test_search_and_sort():
    # Тест поиска
    search_response = requests.get(f'{BASE_URL}/tasks?q=лабораторную')
    assert search_response.status_code == 200
    assert len(search_response.json()) > 0

    # Тест сортировки
    sort_response = requests.get(f'{BASE_URL}/tasks?sort=created_at')
    assert sort_response.status_code == 200
    tasks = sort_response.json()
    if len(tasks) > 1:
        assert tasks[0]['created_at'] <= tasks[1]['created_at']

if __name__ == '__main__':
    pytest.main([__file__])
```

Views.py

```python
from flask import Blueprint, request, jsonify
from data import (
    get_all_tasks, get_task_by_id, add_task,
    update_task, delete_task, get_task_stats
)

tasks_blueprint = Blueprint('tasks', __name__)

@tasks_blueprint.route('/tasks', methods=['GET'])
def list_tasks():
    status_filter = request.args.get('status')
    search_query = request.args.get('q')
    sort_by = request.args.get('sort')

    tasks = get_all_tasks(status_filter)

    # Поиск
    if search_query:
        tasks = [task for task in tasks if search_query.lower() in task['title'].lower()]

    # Сортировка
    if sort_by == 'created_at':
        tasks = sorted(tasks, key=lambda x: x['created_at'])

    return jsonify(tasks)

@tasks_blueprint.route('/tasks', methods=['POST'])
def create_task():
    data = request.get_json()
    if not data or 'title' not in data:
        return jsonify({"error": "Title is required"}), 400

    try:
        task = add_task(
            title=data['title'],
```

```
            description=data.get('description'),
            status=data.get('status', 'pending')
        )
        return jsonify(task), 201
    except ValueError as e:
        return jsonify({"error": str(e)}), 400

@tasks_blueprint.route('/tasks/<int:task_id>', methods=['GET'])
def get_task(task_id):
    task = get_task_by_id(task_id)
    if not task:
        return jsonify({"error": "Task not found"}), 404
    return jsonify(task)

@tasks_blueprint.route('/tasks/<int:task_id>', methods=['PUT'])
def modify_task(task_id):
    data = request.get_json()
    if not data:
        return jsonify({"error": "No data provided"}), 400

    try:
        updated_task = update_task(
            task_id=task_id,
            title=data.get('title'),
            description=data.get('description'),
            status=data.get('status')
        )
        if not updated_task:
            return jsonify({"error": "Task not found"}), 404
        return jsonify(updated_task)
    except ValueError as e:
        return jsonify({"error": str(e)}), 400

@tasks_blueprint.route('/tasks/<int:task_id>', methods=['DELETE'])
def remove_task(task_id):
    if delete_task(task_id):
        return jsonify({"message": "Task deleted"}), 200
    return jsonify({"error": "Task not found"}), 404

@tasks_blueprint.route('/tasks/stats', methods=['GET'])
def task_statistics():
    stats = get_task_stats()
    return jsonify(stats)
```

Результат

Добавление задач

```
C:\Users\dmrme>curl -X POST -H "Content-Type: application/json" -d "{\"title\":
\"New task\"}" http://localhost:5000/tasks
{
  "created_at": "2025-05-05T11:50:21.524627",
  "description": null,
  "id": 4,
  "status": "pending",
  "title": "New task",
  "updated_at": "2025-05-05T11:50:21.524627"
}
```

Получение всех задач

```
C:\Users\dmrme>curl http://localhost:5000/tasks
[
  {
    "created_at": "2025-05-04T11:58:27.846393",
    "description": "description",
    "id": 1,
    "status": "pending",
    "title": "Task 1",
    "updated_at": "2025-05-04T11:58:27.846393"
  },
  {
    "created_at": "2025-05-04T11:58:27.876500",
    "description": null,
    "id": 2,
    "status": "pending",
    "title": "Task 2",
    "updated_at": "2025-05-04T11:58:27.876500"
  },
  {
    "created_at": "2025-05-04T11:58:27.876500",
    "description": null,
    "id": 3,
    "status": "done",
    "title": "Task 3",
    "updated_at": "2025-05-04T11:58:27.876500"
  }
]
```

Получение по id

```
C:\Users\dmrme>curl http://localhost:5000/tasks/1
{
  "created_at": "2025-05-04T11:58:27.846393",
  "description": "description",
  "id": 1,
  "status": "pending",
  "title": "Task 1",
  "updated_at": "2025-05-04T11:58:27.846393"
}
```

Удаление

```
C:\Users\dmrme>curl -X DELETE http://localhost:5000/tasks/1
{
  "message": "Task deleted"
}
```

Изменение задачи по id

```
C:\Users\dmrme>curl -X PUT -H "Content-Type: application/json" -d "{\"title\":\
"New title\",\"status\":\"done\"}" http://localhost:5000/tasks/2
{
  "created_at": "2025-05-04T11:58:27.876500",
  "description": null,
  "id": 2,
  "status": "done",
  "title": "New title",
  "updated_at": "2025-05-05T11:53:41.638585"
}
```

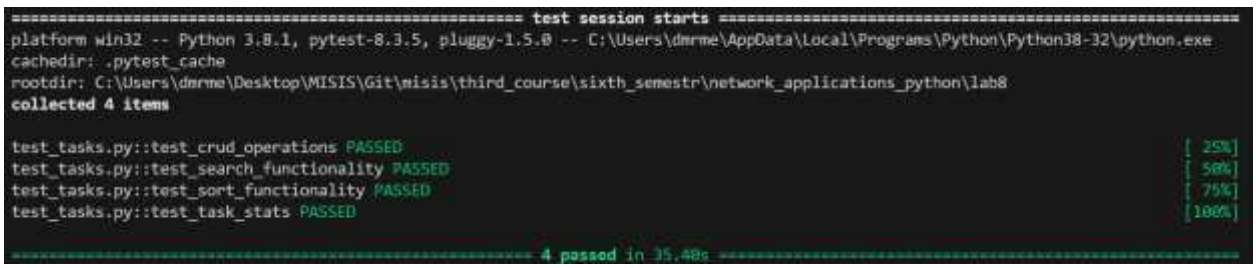
Поиск по заголовку

```
C:\Users\dmrme>curl "http://localhost:5000/tasks?q=New"
[
  {
    "created_at": "2025-05-04T11:58:27.876500",
    "description": null,
    "id": 2,
    "status": "done",
    "title": "New title",
    "updated_at": "2025-05-05T11:53:41.638585"
  },
  {
    "created_at": "2025-05-05T11:50:21.524627",
    "description": null,
    "id": 4,
    "status": "pending",
    "title": "New task",
    "updated_at": "2025-05-05T11:50:21.524627"
  },
  {
    "created_at": "2025-05-05T11:56:43.532947",
    "description": null,
    "id": 5,
    "status": "pending",
    "title": "New",
    "updated_at": "2025-05-05T11:56:43.532947"
  }
]
```

Сортировка по дате

```
C:\Users\dmrme>curl "http://localhost:5000/tasks?sort=-create
[
  {
    "created_at": "2025-05-04T11:58:27.876500",
    "description": null,
    "id": 2,
    "status": "done",
    "title": "New title",
    "updated_at": "2025-05-05T11:53:41.638585"
  },
  {
    "created_at": "2025-05-04T11:58:27.876500",
    "description": null,
    "id": 3,
    "status": "done",
    "title": "Task 3",
    "updated_at": "2025-05-04T11:58:27.876500"
  },
  {
    "created_at": "2025-05-05T11:50:21.524627",
    "description": null,
    "id": 4,
    "status": "pending",
    "title": "New task",
    "updated_at": "2025-05-05T11:50:21.524627"
  },
  {
    "created_at": "2025-05-05T11:56:43.532947",
    "description": null,
    "id": 5,
    "status": "pending",
    "title": "New",
    "updated_at": "2025-05-05T11:56:43.532947"
  }
]
```

Python скрипт тестирования:



Вывод: я освоил принципы работы с базой данных в веб-приложении на Flask.