# Практическа работа №5

## Ход работы:

Установили JMeter

Запустили сервер на python

**Код**

```python
from fastapi import FastAPI, HTTPException, Depends
from pydantic import BaseModel
from typing import List, Optional
import sqlite3
from contextlib import contextmanager

app = FastAPI()

# Database connection manager
@contextmanager
def get_db_connection():
    conn = sqlite3.connect('auction.db')
    conn.row_factory = sqlite3.Row  # This allows accessing columns by name
    try:
        yield conn
    finally:
        conn.close()

# Database setup with better error handling
def init_db():
    try:
        with get_db_connection() as conn:
            cursor = conn.cursor()
            cursor.execute('''
            CREATE TABLE IF NOT EXISTS items (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                description TEXT,
                price REAL NOT NULL CHECK(price >= 0),
                created_at DATETIME DEFAULT CURRENT_TIMESTAMP
            )
            ''')
            conn.commit()
    except sqlite3.Error as e:
        print(f"Database initialization error: {e}")

init_db()

class ItemCreate(BaseModel):
    name: str
    description: Optional[str] = None
    price: float

    class Config:
        schema_extra = {
            "example": {
                "name": "Vintage Watch",
                "description": "A beautiful vintage timepiece",
                "price": 150.50
            }
        }

class Item(ItemCreate):
```

```
    id: int
    created_at: Optional[str] = None

    class Config:
        orm_mode = True

# Dependency for database connection
def get_db():
    with get_db_connection() as conn:
        yield conn

@app.post("/items/", response_model=Item, status_code=201)
def create_item(item: ItemCreate, conn: sqlite3.Connection =
Depends(get_db)):
    try:
        if item.price < 0:
            raise HTTPException(status_code=400, detail="Price cannot be
negative")

        cursor = conn.cursor()
        cursor.execute('''
        INSERT INTO items (name, description, price)
        VALUES (?, ?, ?)
        ''', (item.name, item.description, item.price))
        conn.commit()

        # Get the created item with all fields
        cursor.execute('SELECT * FROM items WHERE id = ?',
(cursor.lastrowid,))
        row = cursor.fetchone()

        return dict(row)

    except sqlite3.Error as e:
        raise HTTPException(status_code=500, detail=f"Database error:
{str(e)}")

@app.get("/items/", response_model=List[Item])
def get_items(conn: sqlite3.Connection = Depends(get_db)):
    try:
        cursor = conn.cursor()
        cursor.execute('SELECT * FROM items ORDER BY created_at DESC')
        rows = cursor.fetchall()
        return [dict(row) for row in rows]
    except sqlite3.Error as e:
        raise HTTPException(status_code=500, detail=f"Database error:
{str(e)}")

@app.get("/items/{item_id}", response_model=Item)
def get_item(item_id: int, conn: sqlite3.Connection = Depends(get_db)):
    try:
        cursor = conn.cursor()
        cursor.execute('SELECT * FROM items WHERE id = ?', (item_id,))
        row = cursor.fetchone()

        if row is None:
            raise HTTPException(status_code=404, detail="Item not found")

        return dict(row)
    except sqlite3.Error as e:
        raise HTTPException(status_code=500, detail=f"Database error:
{str(e)}")

@app.put("/items/{item_id}", response_model=Item)
```

```
def update_item(item_id: int, item: ItemCreate, conn: sqlite3.Connection =
Depends(get_db)):
    try:
        cursor = conn.cursor()
        cursor.execute('''
        UPDATE items SET name = ?, description = ?, price = ?
        WHERE id = ?
        ''', (item.name, item.description, item.price, item_id))
        conn.commit()

        if cursor.rowcount == 0:
            raise HTTPException(status_code=404, detail="Item not found")

        cursor.execute('SELECT * FROM items WHERE id = ?', (item_id,))
        row = cursor.fetchone()
        return dict(row)

    except sqlite3.Error as e:
        raise HTTPException(status_code=500, detail=f"Database error:
{str(e)}")

@app.delete("/items/{item_id}")
def delete_item(item_id: int, conn: sqlite3.Connection = Depends(get_db)):
    try:
        cursor = conn.cursor()
        cursor.execute('DELETE FROM items WHERE id = ?', (item_id,))
        conn.commit()

        if cursor.rowcount == 0:
            raise HTTPException(status_code=404, detail="Item not found")

        return {"message": "Item deleted successfully"}
    except sqlite3.Error as e:
        raise HTTPException(status_code=500, detail=f"Database error:
{str(e)}")

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="127.0.0.1", port=8000)
```

Сервер стал доступен по адресу http://127.0.0.1:8000

## Создание тестов в JMeter

Теперь создадим тесты для этого бэкенда в JMeter.

1. **Создание тестового плана** Откройте JMeter.
   - Создайте новый тестовый план (Test Plan).
2. **Добавление Thread Group**
   - Правый клик на Test Plan → Add → Threads (Users) → Thread Group.
   - Установите "Number of Threads (users)" на 5, "Ramp-Up Period (seconds)" на 10 и "Loop Count" на 1.
3. **Добавление HTTP Request Sampler для POST запроса**
   - Правый клик на Thread Group → Add → Sampler → HTTP Request.
   - В настройках HTTP Request:

**Name**: Create Item
**Server Name or IP**: 127.0.0.1

**Port Number**: 8000
**Method**: POST
**Path**: /items/
**Body Data**: Вставьте пример данных:
json
Копировать код
{
"name": "Item 1",
"description": "A description of Item 1",
"price": 100.0
}

**4. Добавление HTTP Request Sampler для GET запроса**

- Правый клик на Thread Group → Add → Sampler → HTTP Request.
- В настройках HTTP Request: **Name**: Get Items

**Server Name or IP**: 127.0.0.1
**Port Number**: 8000
**Method**: GET
**Path**: /items/

**5. Добавление Listener**

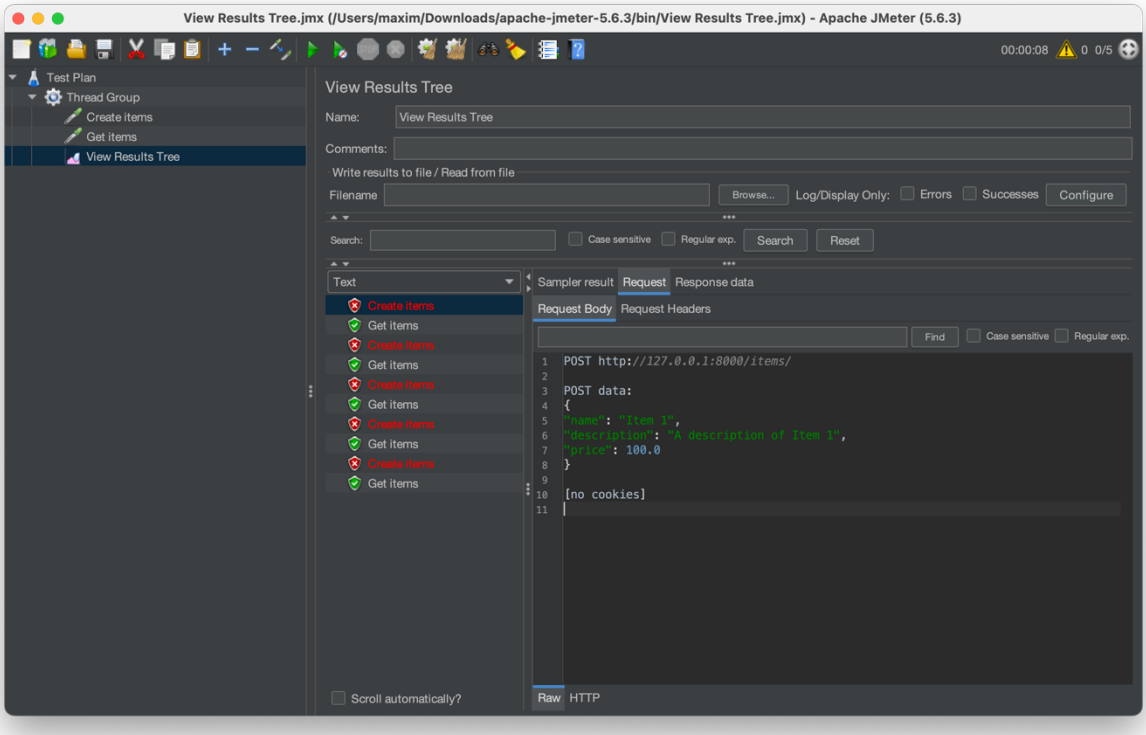
- Правый клик на Thread Group → Add → Listener → View Results Tree.

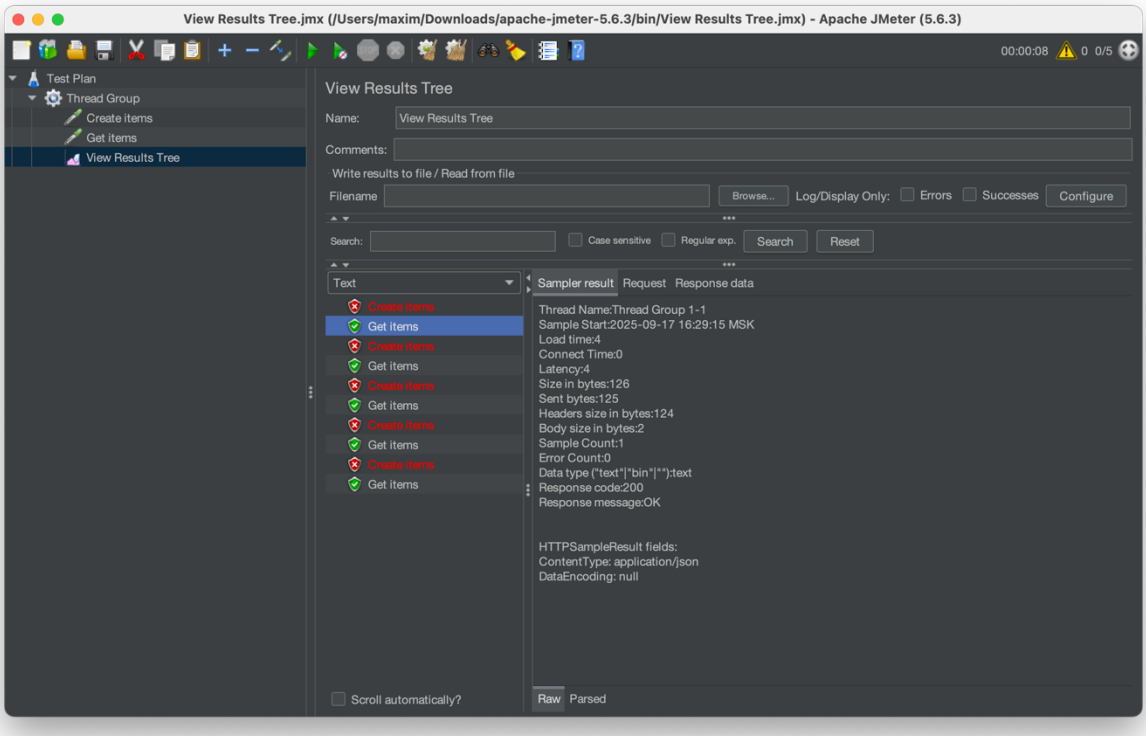
**6. Запуск тестов**

- Нажмите кнопку "Start" на панели инструментов JMeter.
- После завершения теста, просмотрите результаты в "View Results Tree".

**Пример тестов**

**POST запрос** добавляет новый элемент на аукцион и возвращает его данные.
**GET запрос** получает список всех элементов на аукционе.


View Results Tree.jmx (/Users/maxim/Downloads/apache-jmeter-5.6.3/bin/View Results Tree.jmx) - Apache JMeter (5.6.3)
00:00:08 0 0/5
Test Plan
Thread Group
Create items
Get items
View Results Tree
View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Search:
Case sensitive
Regular exp.
Search
Reset
Text
Create items
Get items
Sampler result
Request
Response data
Request Body
Request Headers
Find
Case sensitive
Regular exp.
1 POST http://127.0.0.1:8000/items/
2
3 POST data:
4 {
5 "name": "Item 1",
6 "description": "A description of Item 1",
7 "price": 100.0
8 }
9
10 [no cookies]
11
Scroll automatically?
Raw
HTTP


View Results Tree.jmx (/Users/maxim/Downloads/apache-jmeter-5.6.3/bin/View Results Tree.jmx) - Apache JMeter (5.6.3)
00:00:08 0 0/5
Test Plan
Thread Group
Create items
Get items
View Results Tree
View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Search:
Case sensitive
Regular exp.
Search
Reset
Text
Create items
Get items
Sampler result
Request
Response data
Thread Name:Thread Group 1-1
Sample Start:2025-09-17 16:29:15 MSK
Load time:4
Connect Time:0
Latency:4
Size in bytes:126
Sent bytes:125
Headers size in bytes:124
Body size in bytes:2
Sample Count:1
Error Count:0
Data type ("text"|"bin"|""):text
Response code:200
Response message:OK
HTTPSampleResult fields:
ContentType: application/json
DataEncoding: null
Scroll automatically?
Raw
Parsed