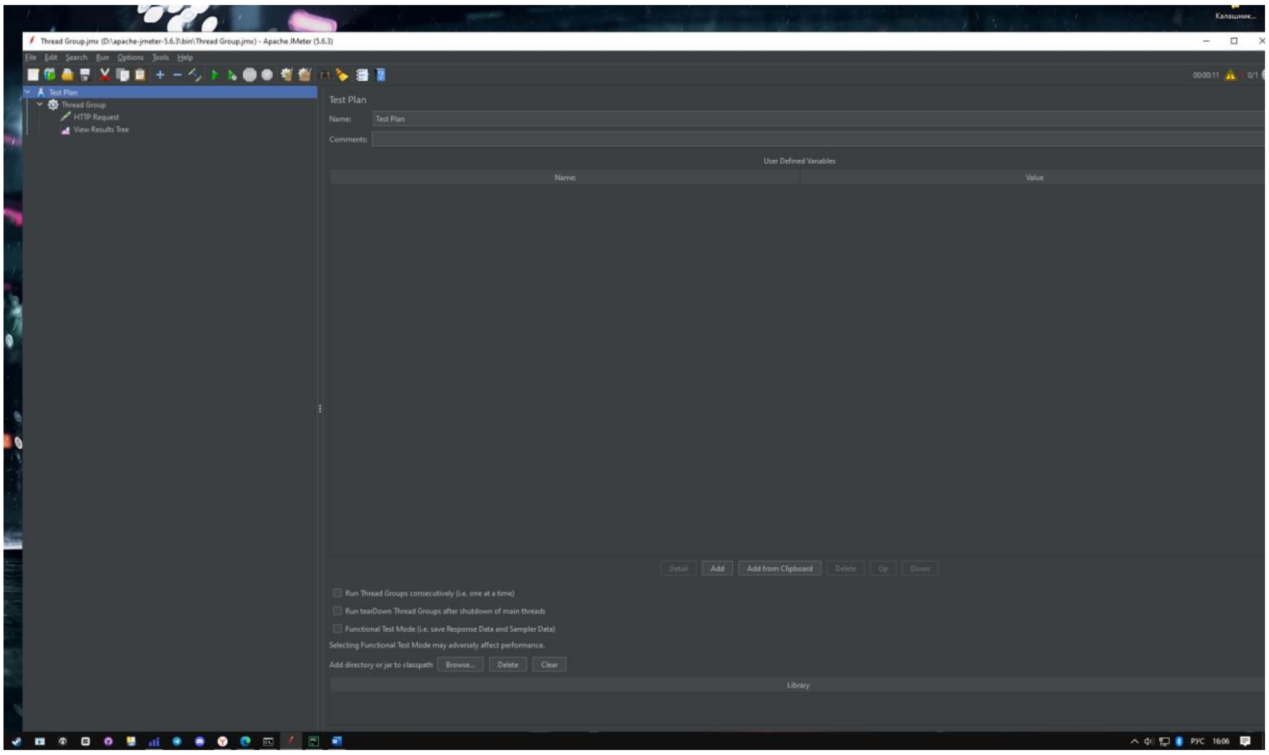
Лабораторная работа №5

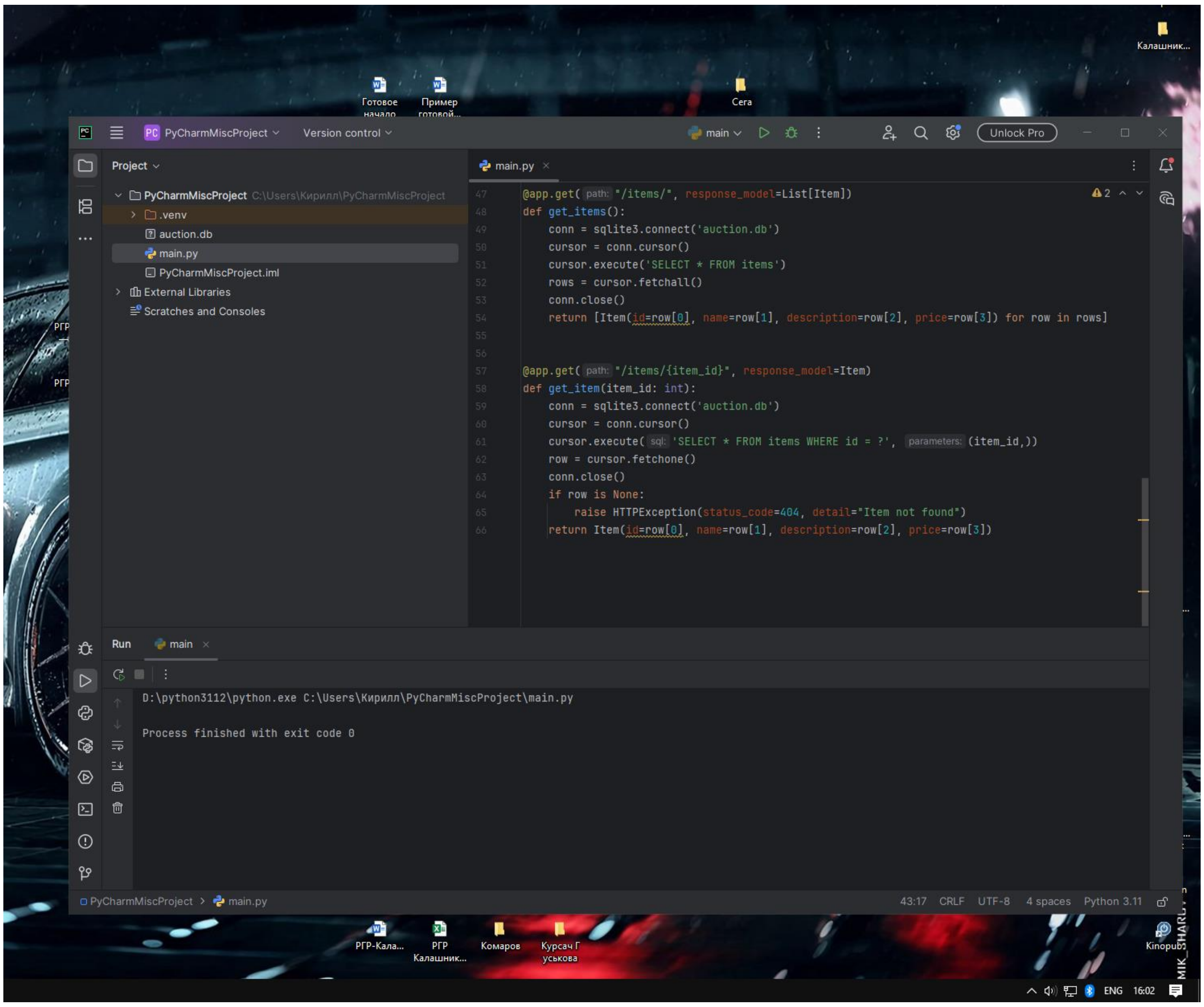
Скачиваем JMeter и создаем новый тестовый план (**Test Plan**).


Thread Group.jmx (D:\apache-jmeter-5.6.3\bin\Thread Group.jmx) - Apache JMeter (5.6.3)
File Edit Search Run Options Tools Help
Test Plan
Thread Group
HTTP Request
View Results Tree
Test Plan
Name: Test Plan
Comments:
User Defined Variables
Name:
Value
Detail
Add
Add from Clipboard
Delete
Up
Down
Run Thread Groups consecutively (i.e. one at a time)
Run tearDown Thread Groups after shutdown of main threads
Functional Test Mode (i.e. save Response Data and Sampler Data)
Selecting Functional Test Mode may adversely affect performance.
Add directory or jar to classpath
Browse...
Delete
Clear
Library

Запускаем сервер и создаем файл **main.py**, его листинг ниже:

```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import List
import sqlite3

app = FastAPI()

# Database setup
def init_db():
    conn = sqlite3.connect('auction.db')
    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
        )
    ''')
    conn.commit()
    conn.close()

init_db()

class Item(BaseModel):
    name: str
    description: str = None
    price: float

@app.post("/items/", response_model=Item)
def create_item(item: Item):
    conn = sqlite3.connect('auction.db')
    cursor = conn.cursor()
    cursor.execute('''
        INSERT INTO items (name, description, price)
        VALUES (?, ?, ?)
    ''', (item.name, item.description, item.price))
    conn.commit()
    conn.close()
    return item

@app.get("/items/", response_model=List[Item])
def get_items():
    conn = sqlite3.connect('auction.db')
    cursor = conn.cursor()
```

```python
    cursor.execute('SELECT * FROM items')
    rows = cursor.fetchall()
    conn.close()
    return [Item(id=row[0], name=row[1], description=row[2], price=row[3]) for row in rows]

@app.get("/items/{item_id}", response_model=Item)
def get_item(item_id: int):
    conn = sqlite3.connect('auction.db')
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM items WHERE id = ?', (item_id,))
    row = cursor.fetchone()
    conn.close()
    if row is None:
        raise HTTPException(status_code=404, detail="Item not found")
    return Item(id=row[0], name=row[1], description=row[2], price=row[3])
```



```python
@app.get("/items/", response_model=List[Item])
def get_items():
    conn = sqlite3.connect('auction.db')
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM items')
    rows = cursor.fetchall()
    conn.close()
    return [Item(id=row[0], name=row[1], description=row[2], price=row[3]) for row in rows]


@app.get("/items/{item_id}", response_model=Item)
def get_item(item_id: int):
    conn = sqlite3.connect('auction.db')
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM items WHERE id = ?', (item_id,))
    row = cursor.fetchone()
    conn.close()
    if row is None:
        raise HTTPException(status_code=404, detail="Item not found")
    return Item(id=row[0], name=row[1], description=row[2], price=row[3])
```

```
D:\python3112\python.exe C:\Users\Кирилл\PyCharmMiscProject\main.py

Process finished with exit code 0
```

Теперь создадим тесты для этого бэкенда в JMeter.

1. Создание тестового плана

- Открываем JMeter.
- Создаем новый тестовый план (Test Plan).

2. Добавление Thread Group

- Правый клик на Test Plan → Add → Threads (Users) → Thread Group.
- Установливаем "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:

-
  ```
  {
      "name": "Test Item",
      "description": "Test Description",
      "price": 100.0
  }
  ```

4. Добавление HTTP Request Sampler для GET запроса

o Правый клик на Thread Group → Add → Sampler → HTTP Request.

o В настройках HTTP Request:

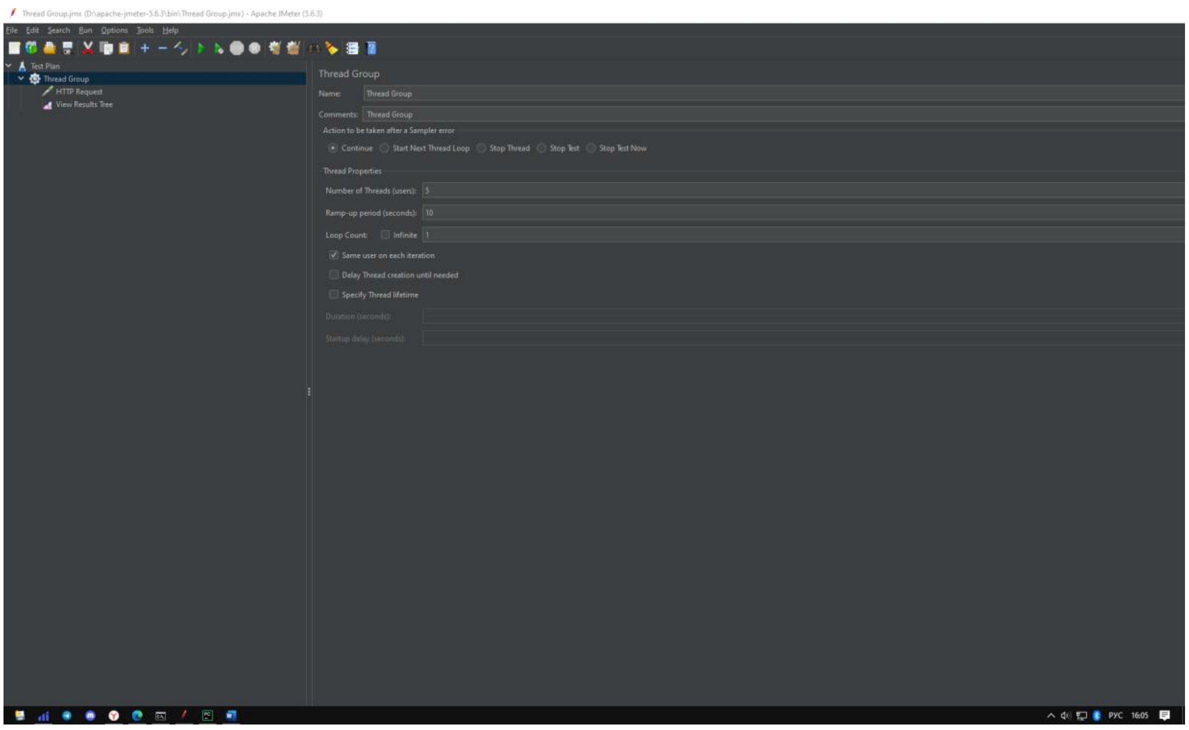
- Name: Get Items
- Server Name or IP: 127.0.0.1
- Port Number: 8000
- Method: GET
- Path: /items/

5. Добавление Listener

- o 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".


Thread Group.jmx (D:\apache-jmeter-5.6.3\bin\Thread Group.jmx) - Apache JMeter (5.6.3)

Thread Group

Name: Thread Group

Comments: Thread Group

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users): 5

Ramp-up period (seconds): 10

Loop Count: ☐ Infinite 1

☑ Same user on each iteration

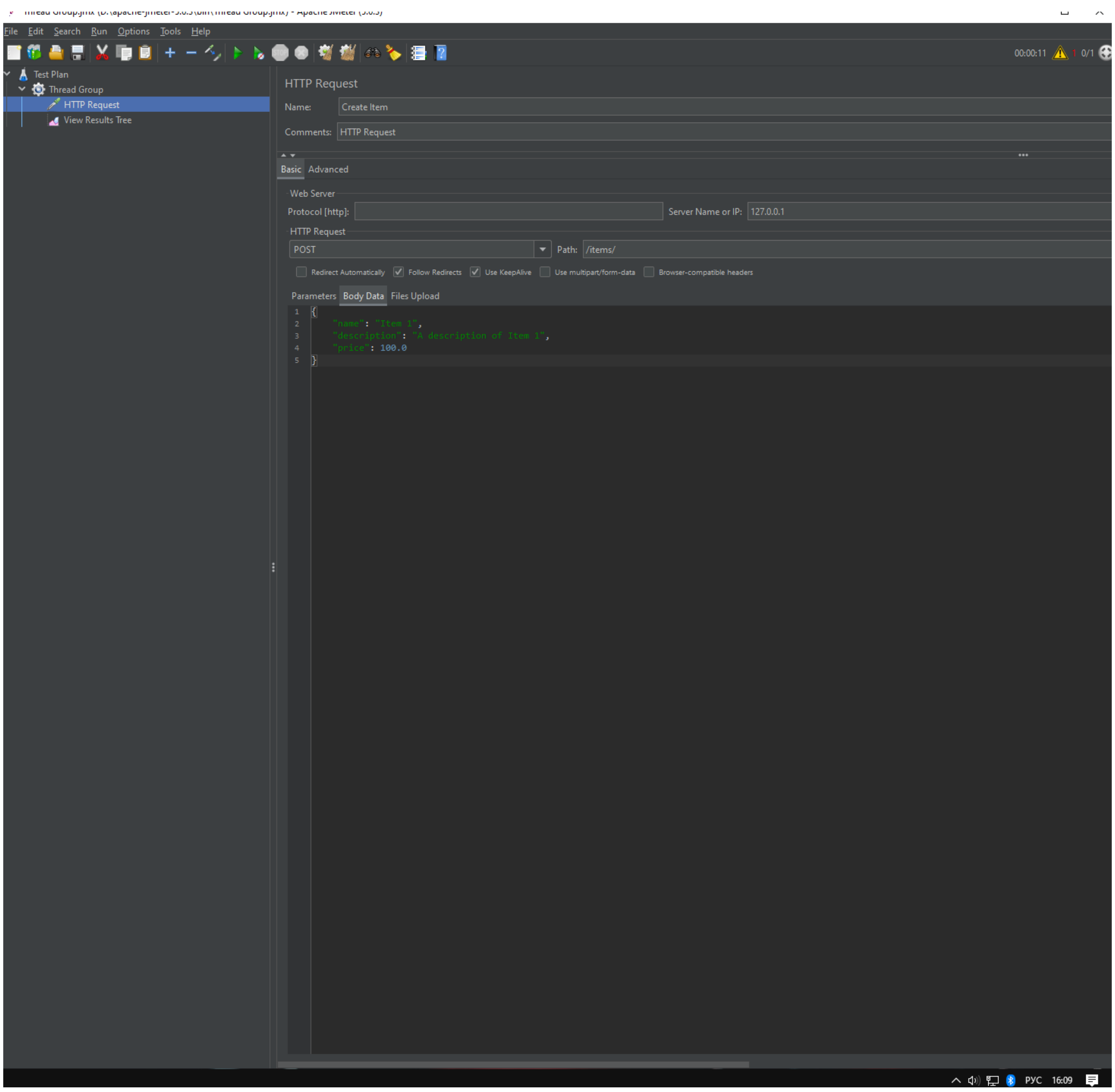
☐ Delay Thread creation until needed

☐ Specify Thread lifetime

Duration (seconds):

Startup delay (seconds):

Настройка HTTP Request Sampler для POST запроса:



HTTP Request

Name: Create Item

Comments: HTTP Request

Web Server

Protocol [http]:

Server Name or IP: 127.0.0.1

HTTP Request

POST

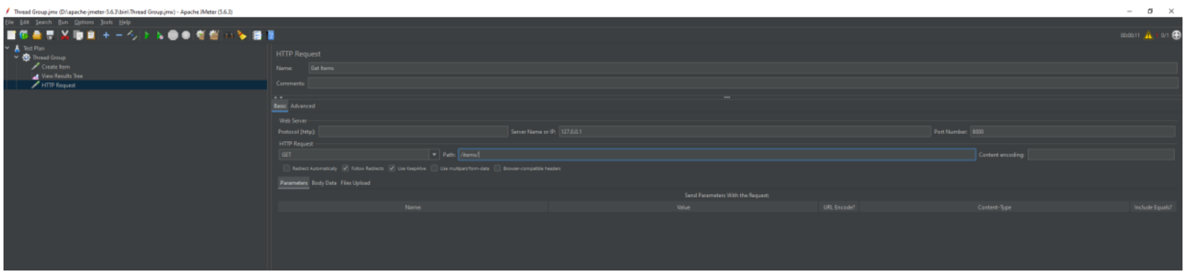
Path: /items/

Redirect Automatically, [✓] Follow Redirects, [✓] Use KeepAlive, Use multipart/form-data, Browser-compatible headers

Body Data

```
{
    "name": "Item 1",
    "description": "A description of Item 1",
    "price": 100.0
}
```

Настройка HTTP Request Sampler для GET запроса:



Thread Group.jmx (D:\apache-jmeter-5.6.3\bin\Thread Group.jmx) - Apache JMeter (5.6.3)

HTTP Request

Name: Get Items

Comments:

Web Server

Protocol [http]:

Server Name or IP: 127.0.0.1

Port Number: 8000

HTTP Request

GET

Path: /items/

Content encoding:

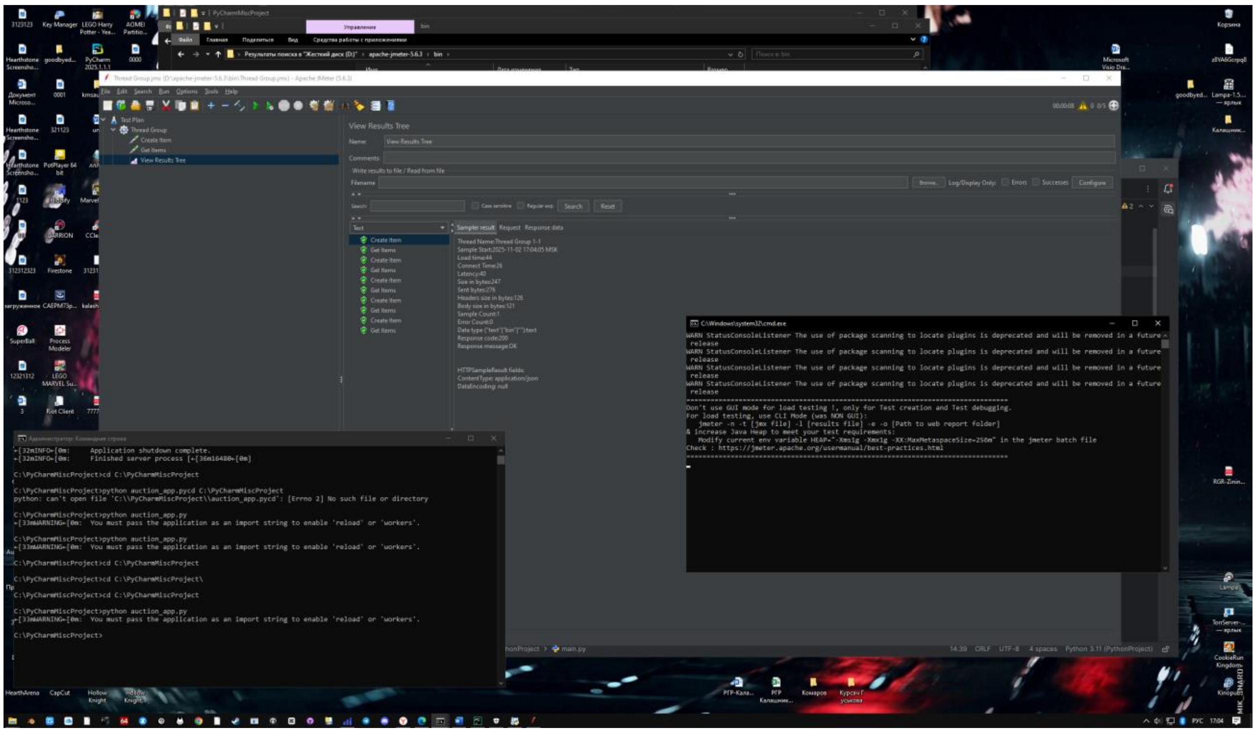
Redirect Automatically, [✓] Follow Redirects, [✓] Use KeepAlive, Use multipart/form-data, Browser-compatible headers

Parameters

Send Parameters With the Request:

Name	Value	URL Encode?	Content-Type	Include Equals?

Результаты выполнения тесто представлены ниже:



View Results Tree

Name: View Results Tree

Write results to file / Read from file

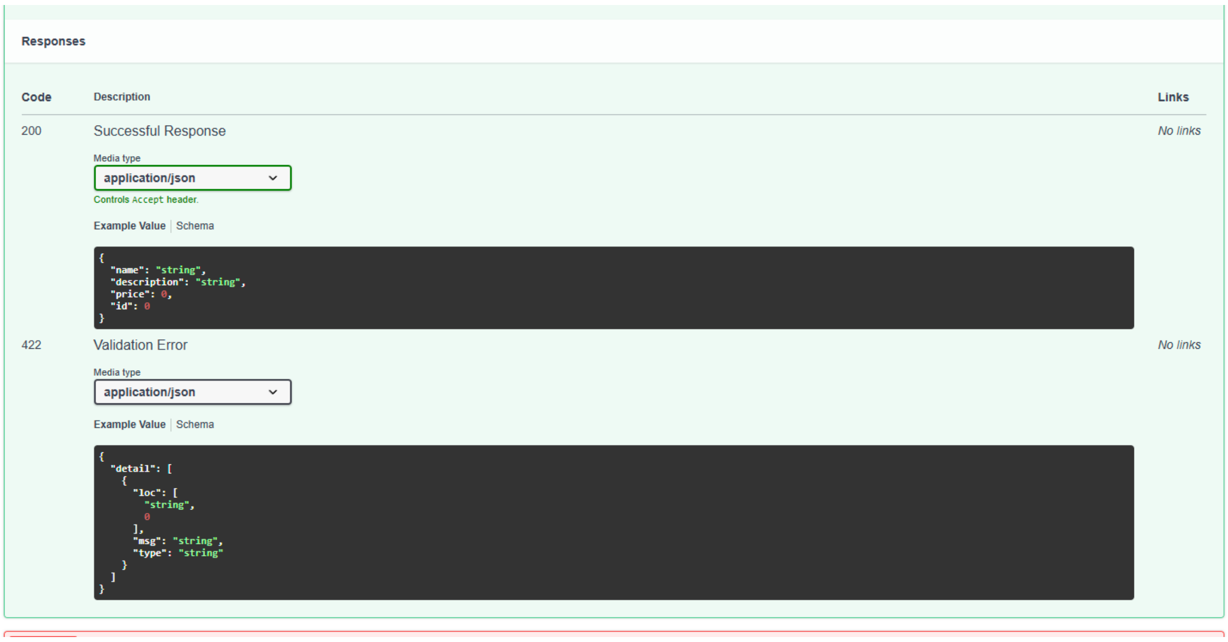
```
Create Item
Get Items
Create Item
Get Items
Create Item
Get Items
Create Item
Get Items
Create Item
Get Items
```

```
Thread Name:Thread Group 1-1
Sample Start:2025-11-02 17:04:05 MSK
Load time:44
Connect Time:26
Latency:40
Size in bytes:247
Sent bytes:276
Headers size in bytes:126
Body size in bytes:121
Sample Count:1
Error Count:0
Data type ("text"|"bin"|""):text
Response code:200
Response message:OK


HTTPSampleResult fields:
ContentType: application/json
DataEncoding: null
```

```
←[32mINFO←[0m:     Application shutdown complete.
←[32mINFO←[0m:     Finished server process [←[36m16480←[0m]

C:\PyCharmMiscProject>cd C:\PyCharmMiscProject

C:\PyCharmMiscProject>python auction_app.pycd C:\PyCharmMiscProject
python: can't open file 'C:\\PyCharmMiscProject\\auction_app.pycd': [Errno 2] No such file or directory

C:\PyCharmMiscProject>python auction_app.py
←[33mWARNING←[0m:  You must pass the application as an import string to enable 'reload' or 'workers'.

C:\PyCharmMiscProject>python auction_app.py
←[33mWARNING←[0m:  You must pass the application as an import string to enable 'reload' or 'workers'.

C:\PyCharmMiscProject>cd C:\PyCharmMiscProject

C:\PyCharmMiscProject>cd C:\PyCharmMiscProject\

C:\PyCharmMiscProject>cd C:\PyCharmMiscProject

C:\PyCharmMiscProject>python auction_app.py
←[33mWARNING←[0m:  You must pass the application as an import string to enable 'reload' or 'workers'.

C:\PyCharmMiscProject>
```

```
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future
 release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future
 release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future
 release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future
 release
================================================================================
Don't use GUI mode for load testing !, only for Test creation and Test debugging.
For load testing, use CLI Mode (was NON GUI):
   jmeter -n -t [jmx file] -l [results file] -e -o [Path to web report folder]
& increase Java Heap to meet your test requirements:
   Modify current env variable HEAP="-Xms1g -Xmx1g -XX:MaxMetaspaceSize=256m" in the jmeter batch file
Check : https://jmeter.apache.org/usermanual/best-practices.html
================================================================================
```

Проверка в браузере



Responses

Code	Description	Links
200	Successful Response	*No links*
422	Validation Error	*No links*

200 Successful Response

Media type: application/json

Controls `Accept` header.

Example Value | Schema

```
{
  "name": "string",
  "description": "string",
  "price": 0,
  "id": 0
}
```

422 Validation Error

Media type: application/json

Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

Готовый код для **auction_app**

```
from fastapi import FastAPI, HTTPException, Body
from pydantic import BaseModel
from typing import List, Optional
import sqlite3
app = FastAPI()
# Database setup
def init_db():
    conn = sqlite3.connect('auction.db')
    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
        )
    ''')
    conn.commit()
    conn.close()
init_db()
class Item(BaseModel):
    name: str
    description: Optional[str] = None
    price: float
class ItemResponse(Item):
    id: int
@app.post("/items/", response_model=ItemResponse)
def create_item(item: Item = Body(...)):
    conn = sqlite3.connect('auction.db')
    cursor = conn.cursor()
    cursor.execute('''
        INSERT INTO items (name, description, price)
        VALUES (?, ?, ?)
    ''', (item.name, item.description, item.price))
    item_id = cursor.lastrowid
    conn.commit()
    conn.close()

    return ItemResponse(
        id=item_id,
        name=item.name,
        description=item.description,
        price=item.price
    )
@app.get("/items/", response_model=List[ItemResponse])
def get_items():
    conn = sqlite3.connect('auction.db')
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM items')
    rows = cursor.fetchall()
    conn.close()

    return [
        ItemResponse(
            id=row[0],
            name=row[1],
            description=row[2],
            price=row[3]
        ) for row in rows
    ]
@app.get("/items/{item_id}", response_model=ItemResponse)
def get_item(item_id: int):
    conn = sqlite3.connect('auction.db')
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM items WHERE id = ?', (item_id,))
    row = cursor.fetchone()
    conn.close()
        if row is None:
        raise HTTPException(status_code=404, detail="Item not found")
        return ItemResponse(
        id=row[0],
        name=row[1],
        description=row[2],
        price=row[3]
    )
# Эндпоинт для проверки здоровья сервера
@app.get("/")
def read_root():
    return {"message": "Auction API is running"}
# Эндпоинт для удаления всех items (для тестирования)
@app.delete("/items/")
def delete_all_items():
    conn = sqlite3.connect('auction.db')
    cursor = conn.cursor()
    cursor.execute('DELETE FROM items')
    conn.commit()
    conn.close()
    return {"message": "All items deleted"}
if __name__ == "__main__":
```

```
import uvicorn
uvicorn.run(app, host="127.0.0.1", port=8000)
```