ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ

«Интеграция программных модулей информационной системы с использованием API»

---

1. ОПИСАНИЕ API

Разработанный программный модуль предоставляет REST API для управления данными (CRUD-операции) в информационной системе.

- Язык реализации: Python 3.14
- Фреймворк: FastAPI
- Тип API: REST
- Формат данных: JSON
- База данных: SQLite (встроенная, файл integration.db)
- Документация API: автоматически генерируется по адресам:
  • Swagger UI: http://localhost:8000/docs
  • ReDoc: http://localhost:8000/redoc

Доступные конечные точки (endpoints):

GET / - информация о API
GET /health - проверка работы
GET /items - получить все записи
GET /items/1 - получить запись с id=1
POST /items - добавить запись
PUT /items/1 - изменить запись
DELETE /items/1 - удалить запись

Все ответы возвращаются в формате JSON с соответствующими HTTP-статусами (200, 201, 400, 404, 422 и т.д.).

---

2. ОПИСАНИЕ СТРУКТУРЫ ДАННЫХ

2.1. Модель данных (таблица items в БД)

Каждая запись содержит:

id - номер записи (создается автоматически)

name - название товара (обязательно)

description - описание (необязательно)

price - цена (необязательно, >= 0)

quantity - количество (необязательно, >= 0)

2.2. Формат JSON для запросов и ответов

Пример объекта (ответ GET /items):

```
{
  "id": 1,
  "name": "Ноутбук",
  "description": "Игровой ноутбук",
  "price": 1500.0,
  "quantity": 5,
  "created_at": "2026-02-11T09:10:54",
  "updated_at": "2026-02-11T09:10:54"
}
```

---

3. ПРИМЕРЫ ЗАПРОСОВ И ОТВЕТОВ

3.1. Создание новой записи (POST /items)

Запрос:

POST http://localhost:8000/items

Content-Type: application/json


```
{
  "name": "Монитор",
  "description": "27-дюймовый 4K",
  "price": 400.0,
  "quantity": 8
}
```


Ответ (201 Created):


```
{
  "id": 2,
  "name": "Монитор",
  "description": "27-дюймовый 4K",
  "price": 400.0,
  "quantity": 8,
  "created_at": "2026-02-11T09:12:30",
  "updated_at": "2026-02-11T09:12:30"
}
```

3.2. Получение всех записей (GET /items)

Запрос:

GET http://localhost:8000/items

Ответ (200 OK):


```
[
  {
    "id": 1,
    "name": "Ноутбук",
    "description": "Игровой ноутбук",
    "price": 1500.0,
    "quantity": 5,
    "created_at": "2026-02-11T09:10:54",
    "updated_at": "2026-02-11T09:10:54"
  },
  {
    "id": 2,
    "name": "Монитор",
    "description": "27-дюймовый 4K",
    "price": 400.0,
    "quantity": 8,
    "created_at": "2026-02-11T09:12:30",
    "updated_at": "2026-02-11T09:12:30"
  }
]
```

3.3. Получение записи по ID (GET /items/2)

Запрос:

GET http://localhost:8000/items/2

Ответ (200 OK):


```
{
  "id": 2,
  "name": "Монитор",
  "description": "27-дюймовый 4K",
  "price": 400.0,
  "quantity": 8,
  "created_at": "2026-02-11T09:12:30",
  "updated_at": "2026-02-11T09:12:30"
}
```

3.4. Обновление записи (PUT /items/2)

Запрос:

PUT http://localhost:8000/items/2

Content-Type: application/json


```
{
  "price": 350.0,
  "quantity": 10
}
```


Ответ (200 OK):

{
  "id": 2,
  "name": "Монитор",
  "description": "27-дюймовый 4K",
  "price": 350.0,
  "quantity": 10,
  "created_at": "2026-02-11T09:12:30",
  "updated_at": "2026-02-11T09:13:15"
}

3.5. Удаление записи (DELETE /items/2)

Запрос:

DELETE http://localhost:8000/items/2

Ответ (200 OK):

{
  "message": "Запись с ID 2 успешно удалена"
}

3.6. Обработка некорректного запроса

Запрос:

POST http://localhost:8000/items

Content-Type: application/json

```
{
  "name": "",
  "price": -100
}
```


Ответ (422 Unprocessable Entity):


```
{
  "detail": [
    {
      "type": "string_too_short",
      "loc": ["body", "name"],
      "msg": "String should have at least 1 character",
      "input": "",
      "ctx": {"min_length": 1}
    },
    {
      "type": "greater_than_equal",
      "loc": ["body", "price"],
      "msg": "Input should be greater than or equal to 0",
      "input": -100,
      "ctx": {"ge": 0.0}
    }
  ]
}
```


---

4. ВЫВОДЫ ПО РАБОТЕ МОДУЛЯ

В ходе выполнения практической работы был разработан и протестирован программный модуль, реализующий REST API для обмена данными между компонентами информационной системы.

Основные результаты:

1. Реализовано полнофункциональное API, обеспечивающее получение, добавление, изменение и удаление данных.

2. В качестве формата обмена используется JSON, что обеспечивает лёгкость интеграции с различными клиентами.

3. Применена встроенная СУБД SQLite – данные сохраняются в файл и доступны после перезапуска сервера.

4. Внедрена обработка ошибочных запросов: возвращаются соответствующие HTTP-статусы (400, 404, 422) и понятные сообщения об ошибках.

5. Клиентская часть (тестовый скрипт) успешно взаимодействует с API, отправляя корректные и некорректные запросы, обрабатывая ответы.

6. Архитектура приложения разделена на логические слои:
   — маршрутизация и валидация (FastAPI, Pydantic);
   — работа с базой данных (отдельный модуль database.py);
   — модели данных (models.py).

7. Тестирование подтвердило корректность работы всех endpoint’ов и устойчивость к некорректным данным.

Таким образом, разработанный модуль полностью соответствует требованиям технического задания. Получен практический опыт создания и интеграции программных модулей с использованием API.