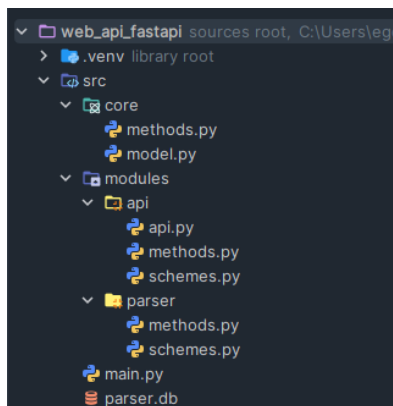


## Отчет на «Работа с БД при помощи SQLAlchemy»

**Задание:** Необходимо создать API которое будет фоном парсить данные с сайта и сохранять их в БД. Доступ к данным нужно осуществить через REST API с возможностью их редактирования и удаления.

Использовался python 3.12

### Обзор кода



Весь проект можно разделить на две части: парсер и api.

### Основные файлы

```
12  async def create_tables(): 1 usage
13      async with async_engine.begin() as conn:
14          await conn.run_sync(Product.metadata.create_all)
15
16
17  def main(): 1 usage
18      # Создание таблиц
19      with Runner(loop_factory=new_event_loop) as runner:
20          runner.run(create_tables())
21
22      # Запуск парсера
23      parse_process = Process(target=run_process)
24      parse_process.start()
25
26      # Запуск api
27      uvicorn.run(app)
28
29      # Закрытие парсера
30      parse_process.terminate()
31      parse_process.join()
32
33
34  if __name__ == '__main__':
35      main()
36
```

Данный файл запускается при запуске программы. Строки 12–14, 19–20 отвечают за создание таблиц в БД, если их еще нет. В 23–24 запускается парсер в другом процессе, чтобы не мешать основному процессу обрабатывать информацию из api. + чтобы можно было масштабировать api не затрагивая парсер. В 27 строке запускается api.

```
model.py x
1 from sqlalchemy import Integer, Text, UniqueConstraint
2 from sqlalchemy.orm import DeclarativeBase, mapped_column, Mapped
3
4
5 class Base(DeclarativeBase): 1 usage
6     pass
7
8
9 class Product(Base): 31 usages
10     __tablename__ = "products"
11
12     id: Mapped[str] = mapped_column(Integer, primary_key=True, autoincrement=True)
13
14     slug: Mapped[str] = mapped_column(Text, nullable=False)
15     name: Mapped[str] = mapped_column(Text, nullable=False)
16     price: Mapped[int] = mapped_column(Integer, nullable=False)
17
18     # noinspection PyTypeChecker
19     __table_args__ = (
20         UniqueConstraint(*columns.slug, name='products_uc'),
21     )
22

corelmethods.py x
1 from sqlalchemy.ext.asyncio import create_async_engine
2
3 async_engine = create_async_engine("sqlite+aiosqlite:///parser.db")
4
```

Слева расположен код, отображающий модель в БД. Справа код для получения асинхронного движка для SQLAlchemy.

## Парсер

Парсер состоит из двух файлов. В первом хранится сам парсер, который мы делали в прошлой лабораторной работе, а также вспомогательные методы. Во втором схема данных продукта.

```
129 def run_process(): 2 usages
130     signal(SIGINT, SIG_IGN)
131     signal(SIGTERM, default_int_handler)
132
133     with suppress(KeyboardInterrupt):
134         with Runner(loop_factory=new_event_loop) as runner:
135             runner.run(run_parser())
136
```

Этот вспомогательный метод нужен для того, чтобы при запуске парсера в другом процессе от корректно работал и завершался. 130 и 131 строка отлавливают сигналы и приводят к тому, что процесс завершиться только если вызвать terminate(). 133 сточка нужна также для корректного завершения программы (Отлавливает KeyboardInterrupt от terminate).

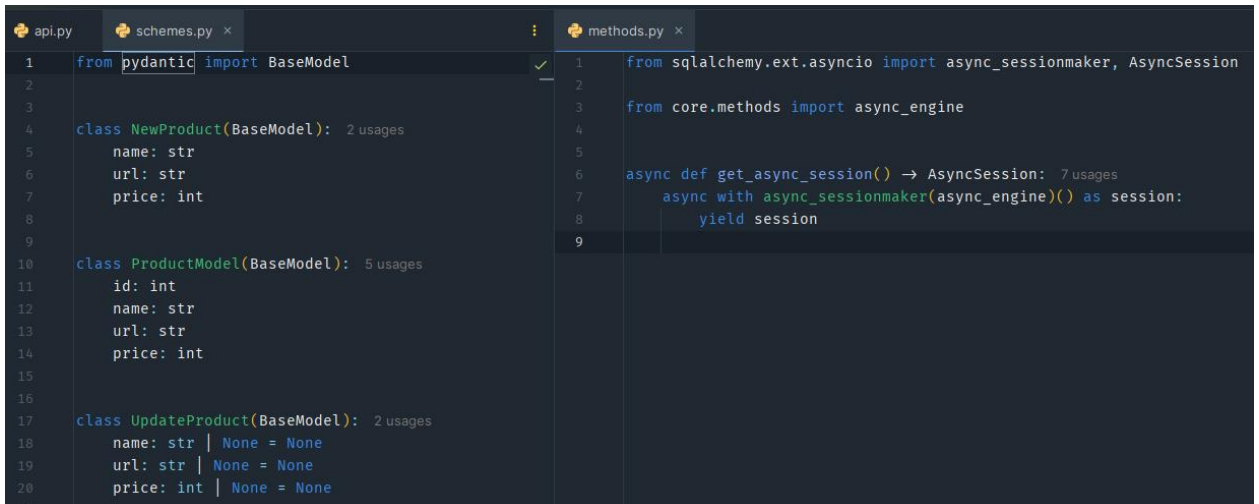
```

99  async def run_parser(): 1 usage
100      async with async_sessionmaker(async_engine)() as session:
101          with Parser() as parser:
102              while True:
103                  result = parser.parse_catalog('https://www.maxidom.ru/catalog/tovary-dlya-poliva/')
104
105                  if result:
106                      request = (
107                          insert(Product)
108                          .values([
109                              {
110                                  'slug': product.url.removeprefix('https://www.maxidom.ru/catalog/').removesuffix('/'),
111                                  'name': product.name,
112                                  'price': product.price
113                              } for product in result
114                          ])
115                      )
116
117                      await session.execute(
118                          request
119                          .on_conflict_do_update(
120                              index_elements=[Product.slug],
121                              set_=dict(name=request.excluded.name, price=request.excluded.price)
122                          )
123                      )
124                      await session.commit()
125
126                      await sleep(10)

```

Данный метод парсит сайт и записывает полученную информацию в бд. Причем из-за 119–121 строк в случае, если на странице поменялась цена он перезапишет данные (При этом данные не дублируются).

## API



The screenshot shows a code editor with three files: `api.py`, `schemes.py`, and `methods.py`. The `api.py` file contains Pydantic models for product data. The `methods.py` file contains an async function to create a database session.

```

1  from pydantic import BaseModel
2
3
4  class NewProduct(BaseModel): 2 usages
5      name: str
6      url: str
7      price: int
8
9
10 class ProductModel(BaseModel): 5 usages
11     id: int
12     name: str
13     url: str
14     price: int
15
16
17 class UpdateProduct(BaseModel): 2 usages
18     name: str | None = None
19     url: str | None = None
20     price: int | None = None

```

```

1  from sqlalchemy.ext.asyncio import async_sessionmaker, AsyncSession
2
3  from core.methods import async_engine
4
5
6  async def get_async_session() → AsyncSession: 7 usages
7      async with async_sessionmaker(async_engine)() as session:
8          yield session
9

```

Справа код моделей pydantic для валидации данных из api. Справа код для создания асинхронных сессий для подключения к бд.

В api поддерживаются 6 методов:

GET	/products	Get Products
POST	/products	Create Product
DELETE	/products	Delete Products
GET	/products/{product_id}	Get Product
PUT	/products/{product_id}	Update Product
DELETE	/products/{product_id}	Delete Product

1. Взять все данные всех продуктов.

Пример вывода:

200

Response body

```
[
  {
    "id": 176,
    "name": "Шланг GF ALPHA 1/2\" 15м",
    "url": "https://www.maxidom.ru/catalog/shlangi-dlya-poliva/1001488016/",
    "price": 1249
  },
  {
    "id": 172,
    "name": "Шланг GF BETA 1/2\" 15м",
    "url": "https://www.maxidom.ru/catalog/shlangi-dlya-poliva/1001488012/",
    "price": 1690
  },
  {
    "id": 173,
    "name": "Шланг GF BETA 1/2\" 25м",
    "url": "https://www.maxidom.ru/catalog/shlangi-dlya-poliva/1001488013/",
    "price": 2490
  },
  {
    "id": 174,
    "name": "Шланг GF BETA 3/4\" 25м",
    "url": "https://www.maxidom.ru/catalog/shlangi-dlya-poliva/1001488014/",
    "price": 4499
  },
  {

```

2. Создать новый продукт

При вводе правильных данных:

Request body required

```
{
  "name": "Шланг1 GF BETA 3/4\" 25м",
  "url": "https://www.maxidom.ru/catalog/shlangi-dlya-poliva/10014880114/",
  "price": 11111111
}
```

Будет получен ответ со статусом 201:

201

Response body

```
{
  "message": "Продукт добавлен",
  "id": 283
}
```

3. Удалить все продукты, которые были сохранены

200

Response body

```
{
  "message": "Все данные удалены"
}
```

4. Взять конкретный продукт по id

Если ввести существующий id:

Name	Description
<b>product_id</b> * required	
integer	<input type="text" value="136"/>
(path)	

То будет получен следующий ответ:

```
200
Response body
{
  "id": 136,
  "name": "шланг спиральный всасывающий СИБРТЕХ 32мм 30м 3атм",
  "url": "https://www.maxidom.ru/catalog/shlangi-dlya-poliva/1001432606/",
  "price": 3457
}
```

Если ввести несуществующий id:

```
404
Undocumented
Error: Not Found
Response body
{
  "message": "Продукт не найден"
}
```

5. Изменить конкретный продукт по id

Если изменить продукт 136, выставив, например только цену, то он успешно обработается.

```
product_id * required
integer
(path) 136

Request body required
{
  "price": 9999999999999999
}

200
Response body
{
  "message": "Продукт обновлен",
  "id": 136
}
```

Результаты после изменения можно увидеть тут:

**product\_id** \* required  
integer  
(path)

136

Execute

Responses

Curl

```
curl -X 'GET' \
'http://127.0.0.1:8000/products/136' \
-H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/products/136
```

Server response

Code	Details
200	<p>Response body</p> <pre>{   "id": 136,   "name": "шланг спиральный всасывающий СИБРТЕХ 32мм 30м Затм",   "url": "https://www.maxidom.ru/catalog/shlangi-dlya-poliva-1001432606/",   "price": 9999999999999999 }</pre>

6. Удалить конкретный продукт по id  
Если удалить существующий продукт:

**product\_id** \* required  
integer  
(path)

136

Execute

Responses

Curl

```
curl -X 'DELETE' \
'http://127.0.0.1:8000/products/136' \
-H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/products/136
```

Server response

Code	Details
200	<p>Response body</p> <pre>{   "message": "Продукт удален" }</pre>

Если удалить продукт, которого нет в бд:

product\_id \* required

integer

(path)

136

Responses

Curl

```
curl -X 'DELETE' \
'http://127.0.0.1:8000/products/136' \
-H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/products/136
```

Server response

Code	Details
404 <i>Undocumented</i>	Error: Not Found

Response body

```
{
  "message": "Продукт не найден"
}
```