



Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования «Уральский федеральный университет имени первого
Президента России Б. Н. Ельцина» (УрФУ)
Институт радиоэлектроники и информационных технологий – РТИ

ОТЧЕТ
по дисциплине: «Основы веб-протоколов»

ФИО: Копытов Е. А.

Академическая группа: РИ-320910

Екатеринбург

2024

Содержание

ВВЕДЕНИЕ	3
Основные файлы	4
Парсер	6
API	8
1. Взять все данные всех продуктов	8
2. Создать новый продукт	9
3. Удалить все продукты, которые были сохранены	11
4. Взять конкретный продукт по id	11
5. Изменить конкретный продукт по id	13
6. Удалить конкретный продукт по id	14
Websockets	17
Заключение	18

ВВЕДЕНИЕ

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 возможность их редактирования и удаления. Все манипуляции с данными должны генерировать уведомления при помощи WebSocket.

Использовался python 3.12. Код всего проекта можно разделить на два модуля: api, parser (рисунок 1).

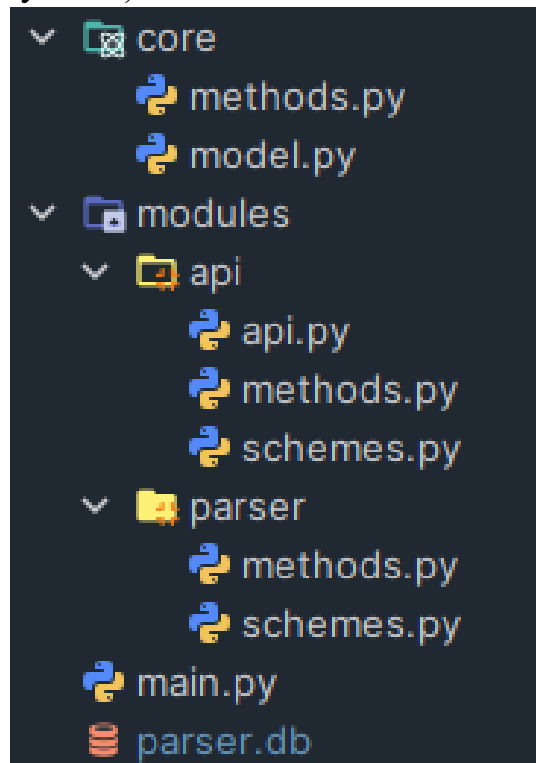


Рисунок 1. Код проекта

Основные файлы

Данный файл (рисунок 2) выполняет следующие функции:

- Создание таблиц в базе данных, если они ещё не существуют;
- Запуск парсера в другом процессе;
- Запуск api через uvicorn.

```
async def create_tables():
    async with async_engine.begin() as conn:
        await conn.run_sync(Product.metadata.create_all)

def main():
    # Создание таблиц
    with Runner(loop_factory=new_event_loop) as runner:
        runner.run(create_tables())

    # Запуск парсера
    parse_process = Process(target=run_process)
    parse_process.start()

    # Запуск api
    uvicorn.run(app)

    # Закрытие парсера
    parse_process.terminate()
    parse_process.join()

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

Рисунок 2. Основной файл

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.

Ниже на рисунке 3 расположен код, отображающий модели в БД. Product – модель отображающая продукт с сайта для парсинга. Event – модель отражает события, которые регистрируются при различных событиях

```

class Base(DeclarativeBase):
    pass

class Product(Base):
    __tablename__ = "products"

    id: Mapped[str] = mapped_column(Integer, primary_key=True, autoincrement=True)

    slug: Mapped[str] = mapped_column(Text, nullable=False)
    name: Mapped[str] = mapped_column(Text, nullable=False)
    price: Mapped[int] = mapped_column(Integer, nullable=False)

    # noinspection PyTypeChecker
    __table_args__ = (
        UniqueConstraint(slug, name='products_uc'),
    )

class Event(Base):
    __tablename__ = "events"

    id: Mapped[str] = mapped_column(Integer, primary_key=True, autoincrement=True)
    description: Mapped[str] = mapped_column(Text, nullable=False)
    created_at: Mapped[datetime] = mapped_column(
        TIMESTAMP, nullable=False, server_default=text("(STRFTIME('%Y-%m-%d %H:%M:%f',
'NOW'))")
    )

```

Рисунок 3. Модели проекта

Ниже на рисунке 4 расположен код для получения асинхронного движка для SQLAlchemy.

```

async_engine = create_async_engine("sqlite+aiosqlite:///parser.db")

```

Рисунок 4. Создание движка

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
```

Рисунок 5. Запуск парсера

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 завершится только если вызвать `terminate()`. 133 строчка 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)
```

Рисунок 6. Тело процесса парсера

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).

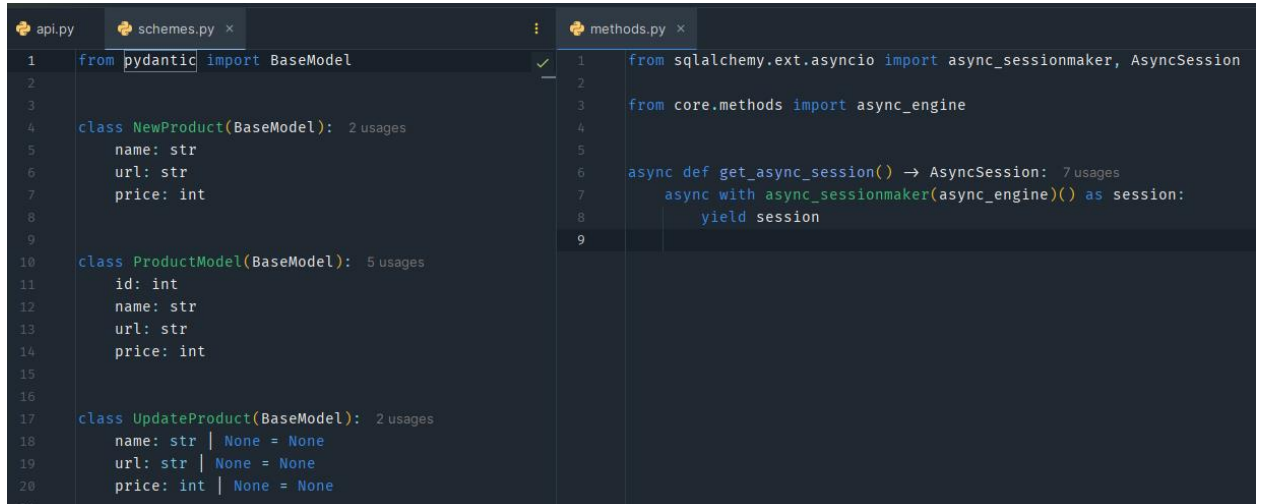
Также парсер регистрирует событие «Парсинг каталога успешно завершен» в конце парсинга.

```
await create_event('Парсинг каталога успешно завершен')
```

Рисунок 7. Создание ивента

API

На рисунке 8 слева код моделей pydantic для валидации данных из api. На рисунке 8 справа код для создания асинхронных сессий для подключения к бд.



```
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
```

Рисунок 8. Код

В api поддерживаются 6 методов работы с данными (рисунок 9).

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

Рисунок 9. Список всех методов

1. Взять все данные всех продуктов

Код можно посмотреть ниже на рисунке 10.


```

@app.get("/products")
async def get_products(session: Annotated[AsyncSession, Depends(get_async_session)]):
    response = await session.execute(
        select(Product.id, Product.slug, Product.name, Product.price)
        .order_by(Product.id)
    )

    return JSONResponse(
        content=[
            ProductModel(
                id=product_id,
                name=name,
                price=price,
                url=f'https://www.maxidom.ru/catalog/{slug}/'
            ).model_dump() for product_id, slug, name, price in response.all()
        ],
        background=BackgroundTask(create_event, f'Получен список продуктов')
    )

```

Рисунок 10. Реализация метода

Пример вывода метода:

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
  },
  {

```

Рисунок 11. Пример ответа

2. Создать новый продукт

Код можно посмотреть ниже на рисунке 12.

```

@app.post("/products", status_code=201)
async def create_product(product: NewProduct, session: Annotated[AsyncSession,
Depends(get_async_session)]):
    request = (
        insert(Product)
        .values(
            slug=product.url.removeprefix('https://www.maxidom.ru/catalog/').removesuffix('/'),
            name=product.name,
            price=product.price
        )
    )

    response = await session.execute(
        request
        .on_conflict_do_update(
            index_elements=[Product.slug],
            set_=dict(name=request.excluded.name, price=request.excluded.price)
        )
        .returning(Product.id)
    )
    await session.commit()

    id_ = response.first()[0]

    return JSONResponse(
        content={"message": "Продукт добавлен", "id": id_},
        background=BackgroundTask(
            create_event, f'Создан новый продукт. Url: {product.url}, Name: {product.name},
            Price: {product.price}'
        )
    )

```

Рисунок 12. Реализация метода

При вводе правильных данных:

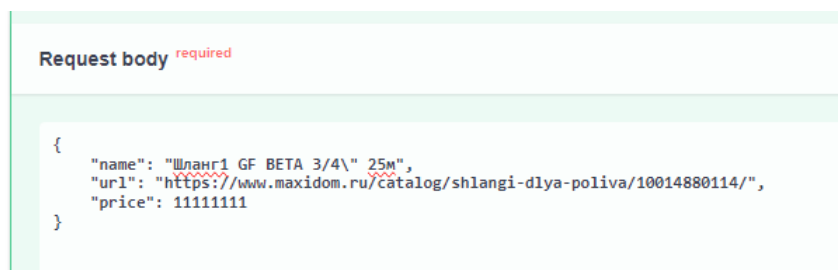


Рисунок 13. Пример вывода

Будет получен ответ со статусом 201:

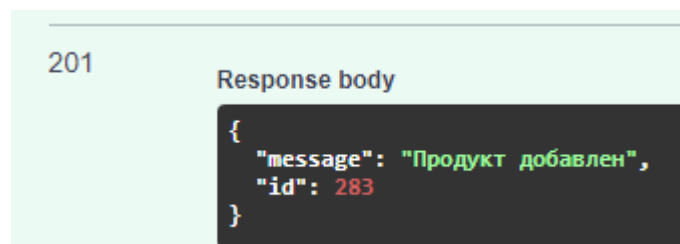


Рисунок 14. Пример вывода

3. Удалить все продукты, которые были сохранены

Код можно посмотреть ниже на рисунке 15.

```
@app.delete("/products")
async def delete_products(session: Annotated[AsyncSession, Depends(get_async_session)]):
    await session.execute(delete(Product))
    await session.commit()

    return JSONResponse(
        content={"message": "Все данные удалены"},
        background=BackgroundTask(create_event, 'Удалены все продукты')
    )
```

Рисунок 15. Реализация метода

Пример вывода программы.

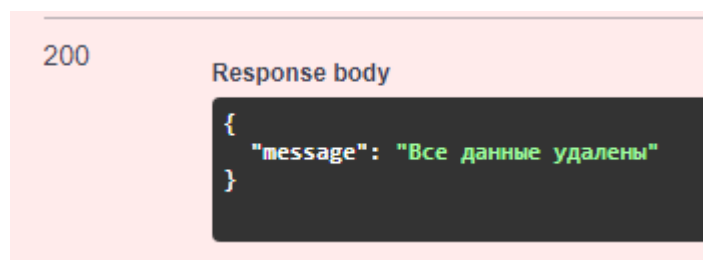


Рисунок 16. Пример вывода

4. Взять конкретный продукт по id

Код можно посмотреть ниже на рисунке 17.

```

@app.get("/products/{product_id}")
async def get_product(product_id: int, session: Annotated[AsyncSession,
Depends(get_async_session)]):
    response = await session.execute(
        select(Product.id, Product.slug, Product.name, Product.price)
        .where(product_id == Product.id)
    )

    result = response.first()

    if not result:
        return JSONResponse(
            content={"message": "Продукт не найден"}, status_code=404,
            background=BackgroundTask(
                create_event, f'Продукт не был найден. ID: {product_id}'
            )
        )

    return JSONResponse(
        content=ProductModel(
            id=result[0],
            name=result[2],
            price=result[3],
            url=f'https://www.maxidom.ru/catalog/{result[1]}/'
        ).model_dump(),
        background=BackgroundTask(
            create_event, f'Получен продукт. ID: {product_id}, Url: {result[1]}, Name:
{result[2]}, Price: {result[3]}'
        )
    )

```

Рисунок 17. Реализация метода

Если ввести существующий id:

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

Рисунок 18. Пример вывода

То будет получен следующий ответ:

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

Рисунок 19. Пример вывода

Если ввести несуществующий id:

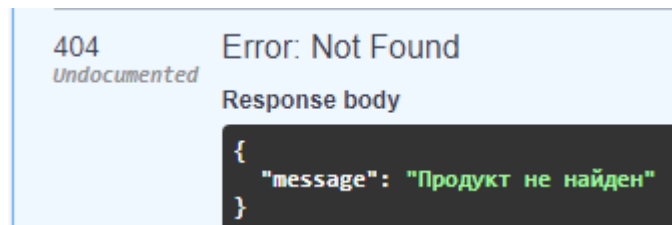


Рисунок 20. Пример вывода

5. Изменить конкретный продукт по id

Код можно посмотреть ниже на рисунке 21.

```
@app.put("/products/{product_id}")
async def update_product(
    product_id: int,
    product: UpdateProduct,
    session: Annotated[AsyncSession, Depends(get_async_session)]
):
    response = await session.execute(
        select(Product.id, Product.slug, Product.name, Product.price)
        .where(product_id == Product.id)
    )

    result = response.first()

    if not result:
        return JSONResponse(
            content={"message": "Продукт не найден"}, status_code=404,
            background=BackgroundTask(
                create_event, f'При обновлении продукт не был найден. ID: {product_id}'
            )
        )

    obj = dict(
        slug=product.url.removeprefix('https://www.maxidom.ru/catalog/').removesuffix('/') if
product.url else result[1],
        name=product.name or result[2],
        price=product.price or result[3]
    )

    await session.execute(
        update(Product)
        .values(**obj)
        .where(product_id == Product.id)
    )
    await session.commit()

    return JSONResponse(
        content={"message": "Продукт обновлен", "id": product_id},
        background=BackgroundTask(create_event, f'Продукт обновлен. ID: {product_id}')
```

Рисунок 21. Реализация метода

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

The screenshot shows a REST client interface. At the top, the **product_id** is set to 136, with a note that it is an integer and required. Below this, the **Request body** is a JSON object: `{ "price": 9999999999999999 }`. The **Response** status is 200, and the **Response body** is a JSON object: `{ "message": "Продукт обновлен", "id": 136 }`.

Рисунок 22. Пример вывода

Результаты после изменения можно увидеть тут:

The screenshot shows a REST client interface. At the top, the **product_id** is set to 136. Below the input field is an **Execute** button. Under the **Responses** section, the **Curl** command is shown: `curl -X 'GET' \ 'http://127.0.0.1:8000/products/136' \ -H 'accept: application/json'`. The **Request URL** is `http://127.0.0.1:8000/products/136`. The **Server response** status is 200. The **Response body** is a JSON object: `{ "id": 136, "name": "шланг спиральный всасывающий СИБРТЕХ 32мм 30м Зата", "url": "https://www.maxidom.ru/catalog/shlangi-dlya-poliva/1001432606/", "price": 9999999999999999 }`.

Рисунок 23. Пример вывода

6. Удалить конкретный продукт по id

Код можно посмотреть ниже на рисунке 24.

```

@app.delete("/products/{product_id}")
async def delete_product(product_id: int, session: Annotated[AsyncSession,
Depends(get_async_session)]):
    response = await session.execute(
        select(Product.id)
        .where(product_id == Product.id)
    )

    if not response.first():
        return JSONResponse(
            content={"message": "Продукт не найден"}, status_code=404,
            background=BackgroundTask(create_event, f'При удалении продукт не был найден ID:
{product_id}'))

    await session.execute(delete(Product).where(product_id == Product.id))
    await session.commit()

    return JSONResponse(
        content={"message": "Продукт удален"},
        background=BackgroundTask(create_event, f'Продукт удален. ID: {product_id}'))

```

Рисунок 24. Реализация метода

Если удалить существующий продукт:

The screenshot shows a REST client interface with the following sections:

- product_id * required**: integer (path) with value 136.
- Execute**: A blue button to execute the request.
- Responses**: A section showing the response details.
 - Curly**: A code block containing the curl command:


```
curl -X 'DELETE' \
'http://127.0.0.1:8000/products/136' \
-H 'accept: application/json'
```
 - Request URL**: A code block containing the URL:


```
http://127.0.0.1:8000/products/136
```
 - Server response**: A table with two columns: **Code** and **Details**.

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

Рисунок 25. Пример вывода

Если удалить продукт, которого нет в бд:

The screenshot shows a REST client interface with a form for a DELETE request. The **product_id** field is marked as ** required* and has a type of *integer (path)*. The value **136** is entered in the input field. Below the form is a blue button. The **Responses** section shows the **Curl** command: `curl -X 'DELETE' \ 'http://127.0.0.1:8000/products/136' \ -H 'accept: application/json'`. The **Request URL** is `http://127.0.0.1:8000/products/136`. The **Server response** section shows a **Code** of **404** (labeled *Undocumented*) and **Details** of **Error: Not Found**. The **Response body** is a JSON object: `{ "message": "Продукт не найден" }`.

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

Response body

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

Рисунок 26. Пример вывода

Websockets

Данный метод отправляет пользователю все ивенты, созданные методами обрабатывающие данные с сайта maxidom.

```

@app.websocket('/events/ws')
async def ws_events(websocket: WebSocket, session: Annotated[AsyncSession,
Depends(get_async_session)]):
    await websocket.accept()

    last_time_point = datetime.now(UTC)

    try:
        while True:
            response = await session.execute(
                select(Event.created_at, Event.description)
                .where(last_time_point <= Event.created_at)
            )

            for created_at, description in response.all():
                await websocket.send_text(description)
                last_time_point = created_at

            await sleep(0.1)
    except WebSocketDisconnect:
        pass

```

Рисунок 27. Реализация метода

Заключение

Был создан парсер сайта maxidom, сохраняющий все данные в бд. Также была создана api с возможностью редактировать, получать и удалять эти данные. Также любые действия с данными генерируют уведомления и их можно получить через websocket.