Пособие: Урок 14 — Работа с API и GitHub

Что получится в конце

- Проект с виртуальным окружением `.venv` и структурой `src/`.
- `api_client.py`: работа с `requests` (GET-запросы, Session, обработка ошибок).
- `api_server.py`: FastAPI-сервер с эндпоинтами `/ping`, `/hello`, `/mult`.
- Репозиторий на GitHub с правильной историей коммитов, ветками и Pull Request.

Часть 1. Подготовка проекта в VS Code

1.1. Создай папку проекта и открой в VS Code

```
mkdir myproject
cd myproject
code .
```

1.2. Создай виртуальное окружение

Windows:

```
python -m venv .venv
.venv\Scripts\activate
```

Mac/Linux:

```
python3 -m venv .venv
source .venv/bin/activate
```

В VS Code выбери интерпретатор:

- `Ctrl+Shift+P` → Python: Select Interpreter → выбери `.venv`.

1.3. Создай структуру проекта

```
myproject/
├── .gitignore
├── requirements.txt
├── README.md
├── src/
│   └── myproject/
│       ├── __init__.py
│       ├── app.py
│       └── utils.py
└── tests/
    ├── __init__.py
    └── test_utils.py
```

1.4. Заполни базовые файлы

`.gitignore`

```
.venv/
__pycache__/
*.pyc
.vscode/
```

`requirements.txt`

```
requests
fastapi
uvicorn
pytest
```

`src/myproject/utils.py`

```
def greet(name: str) -> str:
    return f"Hello, {name}!"
```

`src/myproject/app.py`

```
from myproject.utils import greet
```

```
def main():
    print(greet("World"))

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

tests/test_utils.py

```
from myproject.utils import greet

def test_greet():
    assert greet("Ann") == "Hello, Ann!"
```

1.5. Установи библиотеки

```
pip install -r requirements.txt
```

1.6. Проверь запуск

Windows:

```
set PYTHONPATH=src && python -m myproject.app
```

Mac/Linux:

```
export PYTHONPATH=src && python -m myproject.app
```

Должно вывести: `Hello, World!`

Часть 2. Инициализация Git и первый коммит

2.1. Инициализируй Git

```
git init
```

2.2. Добавь файлы и сделай первый коммит

```
git add .
git commit -m "Initial commit: Project structure setup"
```

2.3. Создай репозиторий на GitHub

1. Перейди на
2. GitHub
3. → New repository.
4. Название: `python-lesson-14` (или любое).
5. Не создавай README, .gitignore (у тебя уже есть).
6. Нажми Create repository.

2.4. Привяжи локальный репозиторий к GitHub

```
git branch -M main
git remote add origin
https://github.com/<username>/python-lesson-14.git
git push -u origin main
```

Обнови страницу GitHub — файлы должны появиться!

Часть 3. Создание API-клиента (ветка `lesson-14-client`)

3.1. Создай новую ветку для клиента

```
git checkout -b lesson-14-client
```

Эта команда создаёт ветку `lesson-14-client` и сразу переключает на неё.

3.2. Создай файл `api_client.py`

В папке `src/myproject/` создай файл `api_client.py`.

Задания для учеников (без готового кода):

1. Совет дня — GET-запрос к `https://api.adviceslip.com/advice`, вывести текст совета.
2. Три совета через Session — использовать `Session()` для 3 запросов подряд.
3. Картинка собаки — получить URL случайной картинки с `https://dog.ceo/api/breeds/image/random`.
4. Обработка ошибок — запросить несуществующий ресурс (`https://jsonplaceholder.typicode.com/xyz`) и обработать ошибку 404 через `raise_for_status()`.

Важные правила:

- Всегда ставь `timeout=5`.
- Проверяй статус через `r.raise_for_status()` (выбросит ошибку при 4xx/5xx).
- Используй `Session()` для нескольких запросов подряд (эффективнее).

3.3. Запуск клиента

Windows:

```
set PYTHONPATH=src && python -m myproject.api_client
```

Mac/Linux:

```
export PYTHONPATH=src && python -m myproject.api_client
```

3.4. Сохрани изменения и отправь на GitHub

```
git add src/myproject/api_client.py
git commit -m "Add API client with requests"
git push origin lesson-14-client
```

Часть 4. Создание API-сервера (ветка `lesson-14-server`)

4.1. Вернись в `main` и создай новую ветку

```
git checkout main
```

```
git checkout -b lesson-14-server
```

4.2. Создай файл `api_server.py`

В папке `src/myproject/` создай файл `api_server.py`.

Задание для учеников (без готового кода):
Создай FastAPI-сервер с 3 эндпоинтами:

1. `/ping` — возвращает `{"status": "ok"}`.
2. `/hello` — принимает параметр `name` (по умолчанию `"world"`), возвращает `{"message": "Hello, {name}!"}`.
3. `/mult` — принимает `a` и `b` (целые числа), возвращает `{"result": a * b}`.

Подсказка:

```
from fastapi import FastAPI

app = FastAPI(title="Мой сервер Урок 14")

@app.get("/ping")
def ping():
    # твой код

@app.get("/hello")
def hello(name: str = "world"):
    # твой код

@app.get("/mult")
def mult(a: int, b: int):
    # твой код
```

4.3. Запуск сервера

```
bash

uvicorn myproject.api_server:app --reload
```

Флаг `--reload` автоматически перезапускает сервер при изменениях кода (только для разработки).

4.4. Проверка в браузере

- `http://127.0.0.1:8000/ping`
- `http://127.0.0.1:8000/hello?name=Мария`
- `http://127.0.0.1:8000/mult?a=5&b=3`
- `http://127.0.0.1:8000/docs` — интерактивная автодокументация (Swagger UI). Здесь можно тестировать все эндпоинты прямо в браузере!

Важно: если открыть `/` (корень), увидишь `{"detail":"Not Found"}` — это нормально, мы не создавали маршрут для `/`.

4.5. Сохрани изменения и отправь на GitHub

bash

```
git add src/myproject/api_server.py
git commit -m "Add FastAPI server with /ping, /hello, /mult"
git push origin lesson-14-server
```

Часть 5. Работа с GitHub: Pull Request и слияние

5.1. Создай Pull Request для клиента

1. Перейди на GitHub в свой репозиторий.
2. Нажми Compare & pull request (появится после `git push`).
3. Base: `main` ← Compare: `lesson-14-client`.
4. Заполни название: `"Добавлен API-клиент (requests)"`.
5. Нажми Create pull request.

5.2. Слей ветку клиента в `main`

1. На странице Pull Request нажми Merge pull request → Confirm merge.
2. Удали ветку `lesson-14-client` на GitHub (кнопка Delete branch).

5.3. Обнови локальную `main`

bash

```
git checkout main
git pull origin main
```

Теперь `api_client.py` появился в `main`!

5.4. Создай Pull Request для сервера

1. На GitHub нажми Compare & pull request для ветки `lesson-14-server`.
2. Base: `main` ← Compare: `lesson-14-server`.
3. Название: `"Добавлен FastAPI-сервер"`.
4. Нажми Create pull request → Merge pull request → Confirm merge.
5. Удали ветку `lesson-14-server`.

5.5. Финальное обновление `main`

```
git checkout main
git pull origin main
```

Готово! Теперь в `main` есть оба файла: `api_client.py` и `api_server.py`.

Часть 6. Обновление документации

6.1. Обнови `README.md`

Добавь в `README.md`:

```
## Урок 14: Работа с API

### Файлы
- `src/myproject/api_client.py` — клиент для работы с внешними API (requests).
- `src/myproject/api_server.py` — мини-сервер на FastAPI.

### Запуск клиента
**Windows:**
```bash
set PYTHONPATH=src && python -m myproject.api_client
```

Mac/Linux:

```bash
export PYTHONPATH=src && python -m myproject.api_client
```
```

Запуск сервера

```
uvicorn myproject.api_server:app --reload
```

Проверь в браузере: `http://127.0.0.1:8000/docs` (автодокументация).

Основные понятия

- requests — библиотека для отправки HTTP-запросов (клиент).
- FastAPI — фреймворк для создания API-сервера.
- timeout — максимальное время ожидания ответа (обязательно).
- raise_for_status() — выбрасывает ошибку при 4xx/5xx.
- Session() — повторное использование соединения (быстрее).
- /docs — автоматическая документация (Swagger UI).

```
### 6.2. Сохрани изменения
git add README.md
git commit -m "Update README: Lesson 14 documentation"
git push origin main
```

Краткий чеклист для проверки

Технические моменты

- VS Code открыт, выбран интерпретатор `.venv`.
- Установлены `requests`, `fastapi`, `uvicorn`.
- Созданы `api_client.py` и `api_server.py`.
- Клиент успешно запускается и выводит советы/собаку/ошибку.
- Сервер запускается через `uvicorn` и отвечает на `/ping`, `/hello`, `/mult`.
- Открывается `/docs` (Swagger UI) и можно протестировать эндпоинты.

Git и GitHub

- Репозиторий создан на GitHub.

- Базовая структура залита в `main`.
- Ветка `lesson-14-client` создана, код добавлен, PR слит в `main`.
- Ветка `lesson-14-server` создана, код добавлен, PR слит в `main`.
- `README.md` обновлён и отправлен на GitHub.

Понимание

- Ученики понимают разницу: requests = клиент (отправляем запросы), FastAPI = сервер (принимаем запросы).
- Используют `timeout` и `raise_for_status()` в `requests`.
- Умеют запустить FastAPI через `uvicorn` и проверить `/docs`.
- Понимают коды статусов: 2xx (успех), 4xx (ошибка клиента), 5xx (ошибка сервера).
- Умеют создавать ветки, делать коммиты, push, Pull Request и сливать ветки.