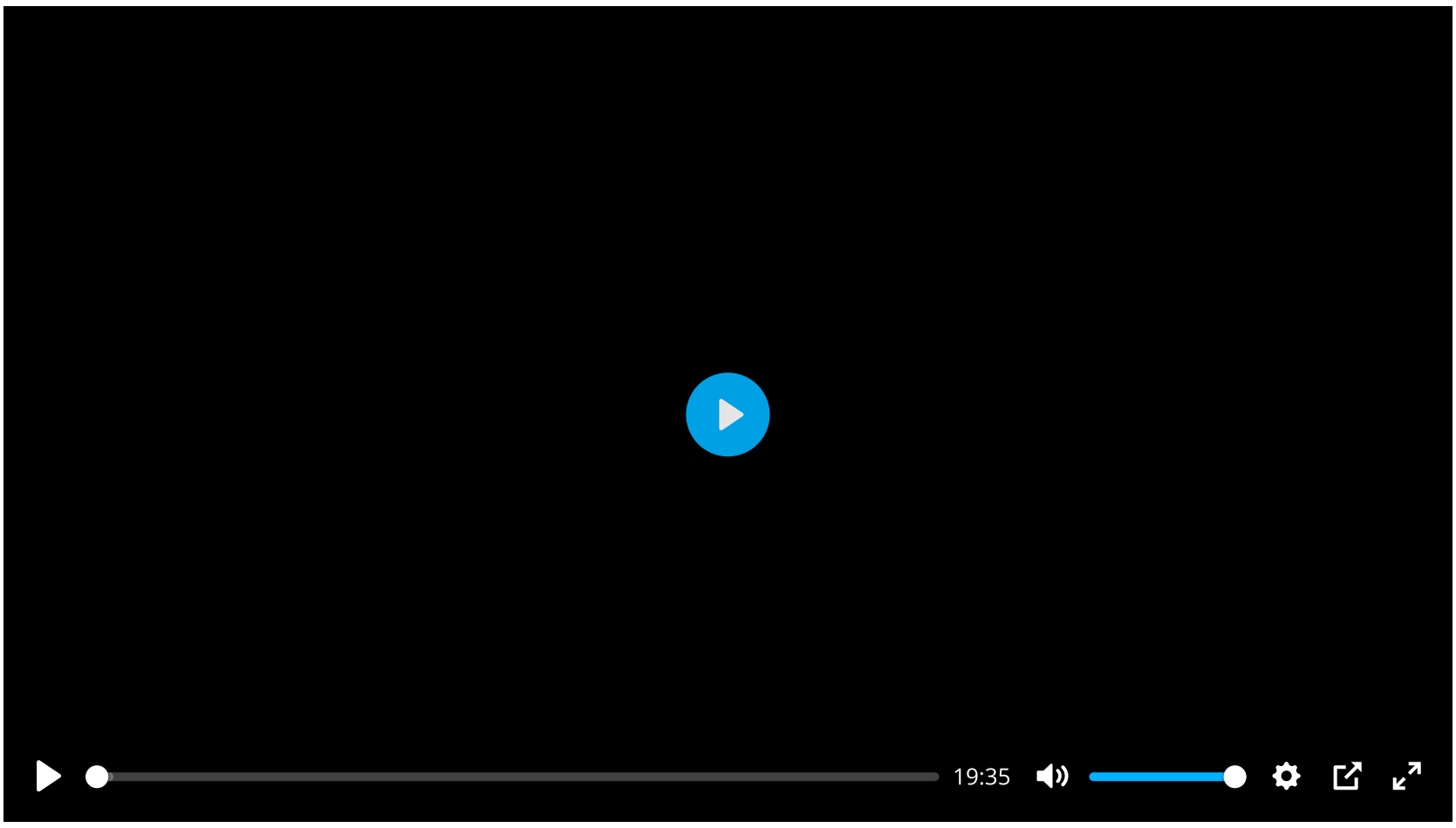
2.4 Web серверы, создание API



Текстовая расшифровка видео:

ПОПУЛЯРНЫЕ ФРЕЙМВОРКИ ДЛЯ PYTHON

План:

- WSGI;
- Django;
- Flask;
- Отложенные задания – Celery;
- Обратите внимание.

WSGI

WSGI (Web Server Gateway Interface) – это описывающая технология.

Так, например, есть веб-сервера (Apache, Nginx), а есть стандарты взаимодействия между ними и приложениями на конкретных языках программирования.

Для разных языков существуют разные привязки по серверам. В случае с Python появился WSGI, как потомок CGI специально для Python. WSGI поддерживается многими популярными фреймворками, например, Diango и Flaks'ом.

Рекомендованный способ разворачивания фрэймворков – использование WSGI в той или иной имплементации. В случае с Apache – это плагин «Mod» в CGI. В случае с Nginx есть несколько разных вариантов, один из самых популярных – интеграция с uWSGI.

Таких серверов много, более того, один из них встроен в стандартную библиотеку Python.

Мы предлагаем вам ознакомиться с самыми популярными реализациями WSGI-сервера:

- Bjoern;
- uWSGI;

- Gunicorn.

Необходимость WSGI:

Для гибкости настройки масштабирования используется промежуточный WSGI-сервер, реализующий дополнительные возможности по масштабированию.

Django

Django – один из первых фреймворков, использующий WSGI.

Рассмотрим пример типичного проекта:

```
mysite
├── manage.py
├── mysite
│   ├── asgi.py
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-310.pyc
│   │   └── settings.cpython-310.pyc
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── news
    ├── admin.py
    ├── apps.py
    ├── __init__.py
    ├── migrations
    │   └── __init__.py
    ├── models.py
    ├── tests.py
    └── views.py
```

Это типичный проект. При помощи команды «**django-admin startproject**» можно сгенерировать базовый костяк проекта. Далее, используется концепция реюзабельных аппов, то есть, там могут быть подмодули, которые отвечают за конкретную функцию. Из этих компонентов они представляют собой приложение.

Нюанс: несмотря на популярность, изначально Django не разрабатывался как фреймворк для написания легких веб-сервисов. Это фреймворк, на котором делали серьезные контентные веб-сайты.

Плюсы:

- Имеет админку, удобные средства для работы с БД, генератор форм и шаблонизатор HTML;
- Позволяет удобно переиспользовать одни и те же компоненты в разных бэкендах на том же фреймворке;
- Легко найти людей на поддержку.

Минусы:

- Не поддерживает REST из коробки и требует дополнительный модуль Django-rest-feamework (GraphQL);
- Встроенный ORM плохо подходит для работы с аналитическими БД;
- Раздутая кодовая база и избыточность.

Flask

Flask – полная противоположность Django. Долгое время Flask в Python назывался микрофреймворком. Это минималистичный фреймворк, решающий одну задачу.

У нас есть URL’ы, Handler’ы, которые должны что-либо возвращать на URL’ы. Здесь нам нужно маппить одно на другое.

Рассмотрим пример:

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
  return "<p>Hello, World!</p>"
```

Это полностью рабочее минимальное приложение на Flask. Здесь есть один эндпоинт, который возвращает «hello_world» с разметкой. По URL'у для корня отдаем его.

Его можно запустить через WSGI, код будет работать.

Для Flask есть множество различных плагинов, реализующих разные аспекты.

```
(dj) meow-nofer@pikachu ~ flask --app test run
 * Serving Flask app 'test'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [29/Apr/2023 17:11:17] "GET / HTTP/1.1" 200 -
```

Так, например, в REST можно делать на Flask из коробки, дополнительно там ставить ничего не нужно.

Если вам нужно больше автоматизации в создании CRUD на базе данных, то обратите внимание на плагины «Flask-Restless» и т.д.

Обратите внимание на данный код:

```
 meow-nofer@pikachu ~ curl http://127.0.0.1:5000/
<p>Hello, World!</p>%
```

Вы можете скопировать его и вставить в свой файл. Код будет работать.

В отличие от Django, Flask не имеет встроенных средств для работы с БД, чаще всего к нему в пару берут SQLAlchemy.

Отложенные задания – Celery

Необходимо упомянуть о реализации отложенных заданий.

Допустим, пользователи совершают какое-то действие на сайте, в результате которого к ним приходит email, например, они регистрируются. Мы хотим отправите email, написать «Привет, спасибо, что присоединились».

Вопрос: как именно это реализовать в ситуации, когда может быть перегружен smtp-сервер и т.д.?

Ответ: в мире Python есть несколько разных обёрток, фреймворков, модулей для реализации подобных задач. Самые популярные – Celery и Dramatiq.

Обратим внимание на Celery.

Celery подходит для чего-то долгоиграющего.

Основные положения:

- При работе с данными абсолютно нормально иметь долгоиграющие процессы для их обработки на бэкенде.
- В идеальном мире, они не должны мешать работе самого API.
- Таски должны быть персистентными – не исчезать при падении приложения или даже текущего сервера.

Выглядит это следующим образом:

```python
from celery import Celery

app = Celery(
  'tasks',
  backend='redis://localhost',
  broker='pyamqp://guest@localhost//'
)

@app.task
def add(x, y):
  return x + y
```

Есть специальная обертка, которая называется «Celery». Нам необходимо заранее указать функции Python, которые будем выполнять в отложенном режиме. То есть, мы описываем функции, запускаем воркер и используем подобие RPS, когда из нашего бэкенда прилетает указание на запуск определенного метода с указанными данными.

Здесь мы описываем наши таски и дальше этот файл сохраняем (условно, «task.py»).

Далее, запускаем экземпляр Worker Celery. Указываем «~$ Celery – A tasks worker –loglevel=INFO».

Далее, в самом бэкенде мы вызываем task с использованием специальной дополнительной обертки:

```python
from tasks import add
result = add.delay(4, 4)
print(result.ready())  # True/False
...
print(result.get())  # 8
```

Поскольку мы навесили декоратор, там появились разные дополнительные методы. Например, мы можем вызвать метод «Delay». Когда задача выполнится, мы можем получить ответ в Result.

Главное: после вызова метода метод «Delay» не обязательно ждать ответ. Мы сразу можем сообщить, что задача создана, после чего спокойно обработать на бэкенде.

Обратите внимание

Django – прост, подходит для старта;

Flask – сложнее, но больше гибкости;

Celery – подходит для реализации отложенных заданий.