

# Разработка API

Лектор: Матвеева Светлана

# Важные моменты

- > отмечаемся на портале
- > разбираем прошлый квиз
- > проходим новый квиз



# Что будет сегодня



- > API
- > Текстовые форматы
- > REST
- > Django Rest Framework

# Application Programming Interface

• • • • •

# Что такое API?

**Application Programming Interface** – описание способов, которыми компьютерная программа может взаимодействовать с другой программой. В случае веба речь идет о взаимодействии различных серверов.

# Backend ⇄ frontend

Web эволюционировал от монолитных приложений (Django + templates & forms) до фронтенда на JS с удобным фреймворком (React/Vue/Angular) и бэкендом с API (Django + Django Rest Framework)



# Backend API ⇄ frontend (пример GET запроса)

- Фронтенд отправляет запрос за данными
- Бэкенд получает запрос во вьюху, делает запрос в БД
- Бэкенд переводит кверисет в текстовый формат (JSON/XML)
- Ответ отправляется на фронтенд
- Фронтенд парсит ответ и вставляет на страницу контент



# Текстовые форматы

• • • • •

# Текстовый формат

При создании API должен быть выбран текстовый формат для обмена данными между клиентом и сервером.



# JSON

JSON (JavaScript Object Notation) – текстовый формат key-value типа (как объект в JS, как dict в Python). Ключ - некоторый атрибут, value - значение (строка, число, true/false, null, массив, объект)

Пример: заказ пиццы в JSON формате

```
{  
    "crust": "original",  
    "toppings": [ "cheese", "pepperoni", "garlic"],  
    "status": "cooking",  
    "customer": {  
        "name": "Brian",  
        "phone": "573-111-1111"  
    }  
}
```

# XML

XML (Extensible Markup Language) – текстовый формат «около HTML-ного» типа.  
Вместо ключей и значений «узлы»: открывающие тэги, значения и закрывающие тэги.

Пример: заказ пиццы в XML формате

```
<order>
    <crust>original</crust>
    <toppings>
        <topping>cheese</topping>
        <topping>pepperoni</topping>
        <topping>garlic</topping>
    </toppings>
    <status>cooking</status>
</order>
```

# Content-type

Не забываем, что headers запросов и ответов должны содержать content-type

Content-Type: application/json  
Content-Type: application/xml

The screenshot shows a network request in a browser's developer tools. The top navigation bar includes tabs for Headers, Payload, Preview, Response, Initiator, Timing, and Cookies. The Headers tab is selected. Below it, the Response Headers section is expanded, showing the following key-value pairs:

- allow:** OPTIONS, GET
- content-length:** 6679
- content-type:** application/json
- date:** Mon, 31 Oct 2022 20:40:17 GMT
- server:** nginx/1.14.2

# REST



# REST

**REST** (REpresentational State Transfer) - архитектурный стиль взаимодействия компонентов распределённого приложения в сети.

Содержит 5 обязательных правил и одно дополнительное, накладывающих ограничение на API.

API, написанное с учетом этих ограничений, называется RESTful API

(RESTful API есть в требованиях в большом количестве вакансий, кстати 😊)

# REST ограничения на API

- Клиент-серверная архитектура
- Stateless (отсутствие состояния)
- Кэширование
- Единообразие интерфейса
- Многоуровневая архитектура
- Код по запросу (необязательное ограничение)

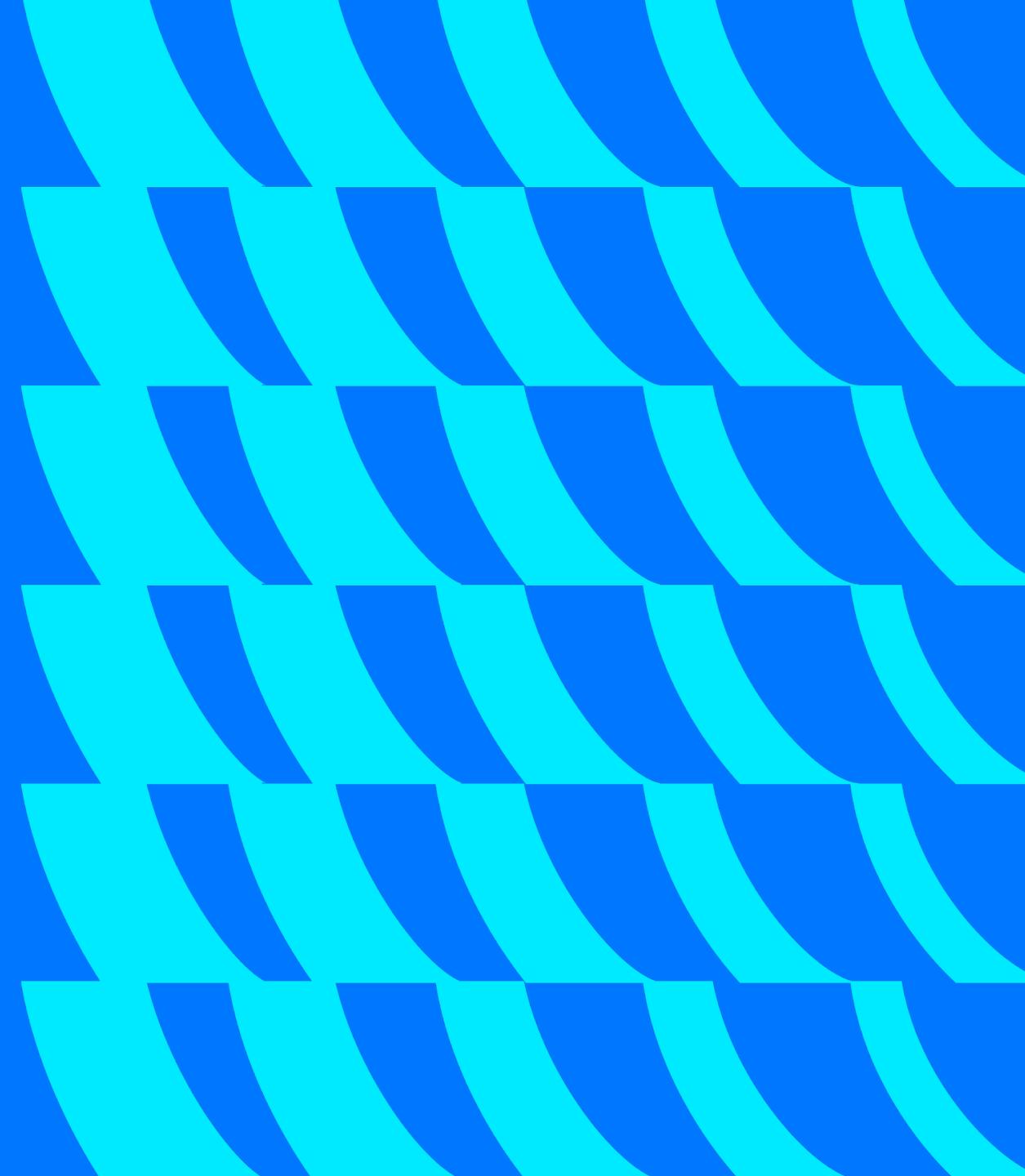
<https://systems.education/what-is-rest> - хорошее объяснение идей REST с примерами

# Преимущества использования REST

- производительность
- масштабируемость
- гибкость к изменениям
- отказоустойчивость
- простота поддержки

# Django Rest Framework

• • • • •



# Django Rest Framework

```
>>> pip install djangorestframework
```

Чем DRF может быть полезен:

- упрощает создание, чтение, изменение и удаление данных
- предоставляет встроенные инструменты для валидации входящих данных
- предлагает встроенные способы авторизации и аутентификации
- предлагает управление правами доступа к данным через API

# Serialization (сериализация)

**Сериализация** – это процесс перевода некоторой структуры данных в формат, который можно хранить (например в файлах) и передавать (например, по HTTP протоколу).

**Десериализация** - обратный процесс.

В нашем случае (Django и JSON в качестве формата передачи данных):

- сериализация = queryset → json
- десериализация = json → queryset

# DRF Serializers

```
>>> from rest_framework import serializers
```

serializers.Serializer - сериализер общего вида

serializers.ModelSerializer - сериализер для модели

# DRF Serializers

- BooleanField,
- CharField,
- ChoiceField,
- DateField,
- DateTimeField,
- DecimalField,
- DictField,
- DurationField,
- EmailField,
- FileField,
- FilePathField,
- FloatField,
- HiddenField,
- HStoreField,
- IPAddressField,
- ImageField,
- IntegerField,
- JSONField,
- ListField,
- ModelField,
- MultipleChoiceField,
- ReadOnlyField,
- RegexField,
- SerializerMethodField,
- SlugField,
- TimeField,
- URLField,
- UUIDField

# DRF Serializers

<https://www.djangoproject.com/en/2.0/topics/http/serializers/>

Документация по сериализаторам

# Django Base View

>>> from django.views import View

Альтернатива функциональным вьюхам

```
from django.http import HttpResponse
from django.views import View

class MyView(View):

    def get(self, request, *args, **kwargs):
        return HttpResponse('Hello, World!')
```

# Django Base View

## Атрибуты

http\_method\_names - принимаемые методы (по умолчанию - все методы)

## Маршрутизация urls.py

```
from django.urls import path

from myapp.views import MyView

urlpatterns = [
    path('mine/', MyView.as_view(), name='my-view'),
]
```

# DRF Viewsets

```
>>> from rest_framework import viewsets
```

Предоставляет 6 методов:

- list (получение списка)
- retrieve (получение отдельного экземпляра)
- create (создание экземпляра)
- update (полное обновление экземпляра)
- partial\_update (частичное обновление)
- destroy (удаление экземпляра)

В регистрации урла:

```
MyView.as_view({'get': 'retrieve', 'put': 'update',
'delete': 'destroy'})
```

```
class UserViewSet(viewsets.ViewSet):
    """
    Example empty viewset demonstrating the standard
    actions that will be handled by a router class.

    If you're using format suffixes, make sure to also include
    the `format=None` keyword argument for each action.
    """

    def list(self, request):
        pass

    def create(self, request):
        pass

    def retrieve(self, request, pk=None):
        pass

    def update(self, request, pk=None):
        pass

    def partial_update(self, request, pk=None):
        pass

    def destroy(self, request, pk=None):
        pass
```

# DRF Generic views

```
>>> from rest_framework import generics  
>>> from rest_framework.generics import ListCreateAPIView
```

Содержит:

- CreateAPIView - для создания экземпляра
- ListAPIView - для получения списка экземпляров
- RetrieveAPIView - для получения одного экземпляра
- DestroyAPIView - для обновления одного экземпляра
- UpdateAPIView - для обновления одного экземпляра
- ListCreateAPIView - для получения списка экземпляров
- RetrieveUpdateAPIView - для получения и обновления экземпляра
- RetrieveDestroyAPIView - для получения и удаления экземпляра
- RetrieveUpdateDestroyAPIView - для получения, удаления и обновления экземпляра

# DRF Generic views

## Атрибуты:

- queryset - кверисет, который будет использован
- serializer\_class - сериализер, который будет использован в этом вьюсете
- lookup\_field - поле модели, которое будет использовано для поиска отдельного экземпляра (pk по умолчанию)
- lookup\_url\_kwarg - тот квартг, в котором будет передано значение для поиска отдельного экземпляра
- pagination\_class - класс для пагинации
- filter\_backends - набор фильтров, которые могут быть использованы для фильтрации кверисета

# DRF Generic views

**Встроенные методы, доступные для переопределения:**

get\_object(self)

get\_queryset(self)

get\_serializer\_context(self)

get\_serializer(self, instance=None, data=None, many=False, partial=False)

get\_paginated\_response(self, data)

paginate\_queryset(self, queryset)

filter\_queryset(self, queryset)

# DRF filters

```
>>> from rest_framework.filters import BaseFilterBackend
```

Позволяет создавать кастомные фильтры

```
class BookYearFilter(BaseFilterBackend):
    def filter_queryset(self, request, queryset, view):
        year_gt = request.GET.get('year_gt')
        year_lt = request.GET.get('year_lt')
        if year_gt:
            queryset = queryset.filter(published_at__year__gt=year_gt)
        if year_lt:
            queryset = queryset.filter(published_at__year__lt=year_lt)
    return queryset
```

# Спасибо!

Домашнее задание:

Переписать views вашего приложения на DRF.

Очень прошу оставить отзыв на портале :)

