

# Django Starter

Django и REST. Обзор REST, обзор Django Rest Framework

# Django Starter

## Introduction



**Лазорык Михаил**

Software developer, 3 года опыта

 mykhailo.lazoryk

 mykhailo-lazoryk



## Django и REST. Обзор REST, обзор Django Rest Framework

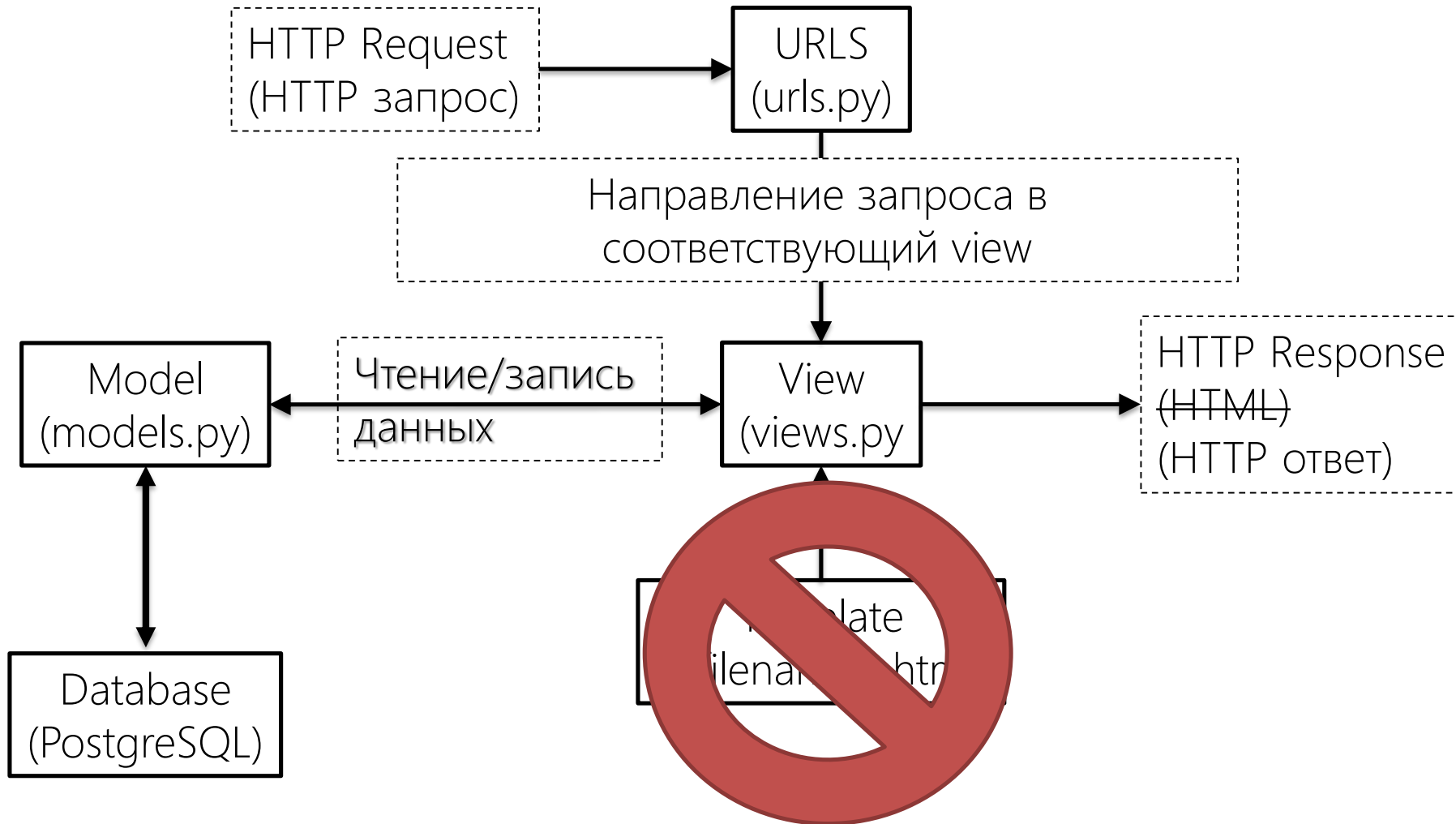
# Django Starter

## План урока

1. Архитектурный стиль REST
2. История возникновения
3. Каким требованиям отвечает RESTful API
4. Преимущества RESTful API
5. Недостатки
6. Обзор инструментов для реализации REST в Django
7. Установка и подключение

# Django Starter

## Структура файлов в Django REST API



# Django Starter

## Архитектурный стиль REST

**REST** (от англ. **Representational State Transfer** — «передача состояния представления») — архитектурный стиль взаимодействия компонентов распределенного приложения в сети.

**REST** представляет собой согласованный набор ограничений, учитываемых при проектировании распределенной системы. В определенных случаях (интернет-магазины, поисковые системы, прочие системы, основанные на данных) это приводит к повышению производительности и упрощению архитектуры.

В широком смысле компоненты в **REST** взаимодействуют наподобие взаимодействия клиентов и серверов во Всемирной паутине.

**REST** является альтернативой **RPC**.

Удалённый вызов процедур, реже Вызов удалённых процедур (от англ. **Remote Procedure Call, RPC**) — класс технологий, позволяющих компьютерным программам вызывать функции или процедуры в другом адресном пространстве (на удалённых компьютерах, либо в независимой сторонней системе на том же устройстве).

# Django Starter

## Архитектурный стиль REST

В сети Интернет вызов удаленной процедуры может представлять собой обычный **HTTP-запрос** (обычно «**GET**» или «**POST**»; такой запрос называют «**REST-запрос**»), а необходимые данные передаются в качестве параметров запроса. Для веб-служб, построенных с учетом **REST** (то есть не нарушающих накладываемых им ограничений), применяют термин «**RESTful**».

**REST** - набор архитектурных принципов построения сервис-ориентированных систем. **RESTful** - прилагательное, употребляющееся по отношению к сервисам, которые следуют принципам **REST**.

В отличие от веб-сервисов (веб-служб) на основе **SOAP**, не существует «официального» стандарта для **RESTful** веб-API. Дело в том, что **REST** является архитектурным стилем, в то время как **SOAP** является протоколом.

Несмотря на то, что **REST** не является стандартом сам по себе, большинство **RESTful-реализаций** используют такие стандарты, как **HTTP**, **URL**, **JSON** и **XML**. **SOAP** (от англ. **Simple Object Access Protocol** — простой протокол доступа к объектам) — протокол обмена структурированными сообщениями в распределенной вычислительной среде.

Первоначально **SOAP** предназначался в основном для реализации удаленного вызова процедур (**RPC**). Сейчас протокол используется для обмена произвольными сообщениями в формате **XML**, а не только для вызова процедур.

# Django Starter

## История возникновения

- Хотя данная концепция лежит в самой основе Всемирной паутины, термин «**REST**» был введён Роем Филдингом (англ. **Roy Fielding**), одним из создателей протокола «**HTTP**», лишь в **2000** году. В своей диссертации «Архитектурные стили и дизайн сетевых программных архитектур» («Architectural Styles and the Design of Network-based Software Architectures») в Калифорнийском университете в Ирвайне он подвел теоретическую основу под способ взаимодействия клиентов и серверов во Всемирной паутине, абстрагировав его и назвав «передачей представительного состояния» («Representational State Transfer»).

Филдинг описал концепцию построения распределенного приложения, при которой каждый запрос (**REST-запрос**) клиента к серверу содержит в себе исчерпывающую информацию о желаемом ответе сервера (желаемом представительном состоянии), и сервер не обязан сохранять информацию о состоянии клиента («клиентской сессии»).

- Стил «**REST**» развивался параллельно с «**HTTP 1.1**», разработанным в 1996—1999 годах, основываясь на существующем дизайне «**HTTP 1.0**», разработанном в 1996 году.





# Django Starter

## Каким требованиям отвечает RESTful API

- **Client-Server.** Система должна быть разделена на клиентов и на серверов. Разделение интерфейсов означает, что, например, клиенты не связаны с хранением данных, которое остается внутри каждого сервера, так что мобильность кода клиента улучшается.

Серверы не связаны с интерфейсом пользователя или состоянием, так что серверы могут быть проще и масштабируемы. Серверы и клиенты могут быть заменяемы и разрабатываться независимо, пока интерфейс не изменяется.

- **Stateless.** Сервер не должен хранить какой-либо информации о клиентах. В запросе должна храниться вся необходимая информация для обработки запроса и, если необходимо, идентификации клиента.
- **Cache.** Каждый ответ должен иметь отметку - является ли он кэшируемым или нет, для предотвращения повторного использования клиентами устаревших или некорректных данных в ответ на дальнейшие запросы.

# Django Starter

## Каким требованиям отвечает RESTful API

- **Uniform Interface.** Единый интерфейс определяет интерфейс между клиентами и серверами. Это упрощает и отделяет архитектуру, которая позволяет каждой части развиваться самостоятельно.
- **Layered System.** В REST допускается разделить систему на иерархию слоев, но с условием, что каждый компонент может видеть компоненты только непосредственно следующего слоя. Например, если вы вызываете службу PayPal, а он в свою очередь вызывает службу Visa, вы о вызове службы Visa ничего не должны знать.
- **Code-On-Demand** (опционально). В REST допускается загрузка и выполнение кода или программы на стороне клиента.

# Django Starter

## Каким требованиям отвечает RESTful API

Принципы единого интерфейса:

- **Identification of resources** (основан на ресурсах). В REST ресурсом является все то, чему можно дать имя. Например, пользователь, изображение, предмет (майка, голодная собака, текущая погода) и т.д.

Каждый ресурс в REST должен быть идентифицирован посредством стабильного идентификатора, который не меняется при изменении состояния ресурса. Идентификатором в REST является URI.

- **Manipulation of resources through representations.** (Манипуляции над ресурсами через представления). Представление в REST используется для выполнения действий над ресурсами. Представление ресурса представляет собой текущее или желаемое состояние ресурса. Например, если ресурсом является пользователь, то представлением может являться XML или HTML описание этого пользователя.

# Django Starter

## Каким требованиям отвечает RESTful API

Принципы единого интерфейса:

- **Self-descriptive messages** (само-документируемые сообщения). Под само-описательностью подразумевается, что запрос и ответ должны хранить в себе всю необходимую информацию для их обработки. Не должно быть дополнительных сообщений или кэшей для обработки одного запроса. Другими словами - отсутствие состояния, сохраняемого между запросами к ресурсам. Это очень важно для масштабирования системы.
- **HATEOAS** (hypermedia as the engine of application state). Статус ресурса передается через содержимое body, параметры строки запроса, заголовки запросов и запрашиваемый URI (имя ресурса). Это называется гипермедиа (или гиперссылки с гипертекстом).

HATEOAS также означает, что, в случае необходимости, ссылки могут содержаться в теле ответа (или заголовках) для поддержки URI, извлечения самого объекта или запрошенных объектов.

# Django Starter

## Преимущества RESTful API

Филдинг указывал, что приложения, не соответствующие приведенным условиям, не могут называться REST-приложениями. Если же все условия соблюдены, то, по его мнению, приложение получит следующие преимущества:

- **Надёжность** (за счёт отсутствия необходимости сохранять информацию о состоянии клиента, которая может быть утеряна);
- **Производительность** (за счёт использования кэша);
- **Масштабируемость**;
- **Прозрачность системы взаимодействия** (особенно необходимая для приложений обслуживания сети);
- **Простота интерфейсов**;
- **Портативность компонентов**;
- **Легкость внесения изменений**;
- **Способность эволюционировать**, приспособляясь к новым требованиям (на примере Всемирной паутины).

# Django Starter

## Недостатки

Проблемы связанные с REST:

- До сих пор нет общего согласования того, что такое **RESTful API**;
- Словарь **REST** поддерживается не полностью;
- Словарь **REST** недостаточно насыщен;
- **RESTful API** трудно дебажить;
- **RESTful API** привязаны к протоколу HTTP.

# Django Starter

## Обзор инструментов для реализации REST в Django

**Django Rest Framework (DRF)** — это библиотека, которая работает со стандартными моделями Django для создания гибкого и мощного API для проекта.

API DRF состоит из 3-х слоев: сериализатора, вида и маршрутизатора:

- **Сериализатор**: преобразует информацию, хранящуюся в базе данных и определенную с помощью моделей Django, в формат, который легко и эффективно передается через API.
- **Вид (ViewSet)**: определяет функции (чтение, создание, обновление, удаление), которые будут доступны через API.
- **Маршрутизатор**: определяет URL-адреса, которые будут предоставлять доступ к каждому виду.



# Django Starter

## Обзор инструментов для реализации REST в Django

- **Модели Django** интуитивно представляют данные, хранящиеся в базе, но API должен передавать информацию в менее сложной структуре.

Хотя данные будут представлены как экземпляры классов Model, их необходимо перевести в формат **JSON** для передачи через API. **Сериализатор DRF** производит это преобразование.

Когда пользователь передает информацию (например, создание нового экземпляра) через API, сериализатор берет данные, проверяет их и преобразует в нечто, что Django может сложить в экземпляр модели. Аналогичным образом, когда пользователь обращается к информации через API, соответствующие экземпляры передаются в **сериализатор**, который преобразовывает их в формат, который может быть легко передан пользователю как JSON.





# Django Starter

## Обзор инструментов для реализации REST в Django

Настройки **fields** позволяют точно указать, какие поля доступны этому **сериализатору**. В качестве альтернативы, может быть установлен **exclude** вместо **fields**, которое будет включать все поля модели, кроме тех, которые указаны в **exclude**.

**Сериализаторы** — это невероятно гибкий и мощный компонент DRF. Хотя подключение **сериализатора** к модели является наиболее распространенной практикой, **сериализаторы** могут использоваться для создания любой структуры данных Python через API, в соответствии с определенными параметрами.

**Сериализатор** анализирует информацию в обоих направлениях (чтение и запись), тогда как **ViewSet** - это тот код, в котором определены доступные операции.

# Django Starter

## Обзор инструментов для реализации REST в Django

Наиболее распространенным **ViewSet** является **ModelViewSet**, который имеет следующие встроенные операции:

- Создание экземпляра: **create ()**
- Получение / чтение экземпляра: **retrieve ()**
- Обновление экземпляра (все или только выбранные поля): **update ()** или **partial\_update ()**
- Уничтожение / Удаление экземпляра: **destroy ()**
- Список экземпляров (с разбивкой по страницам по умолчанию): **list ()**

И наконец, **маршрутизаторы**: они предоставляют верхний уровень **API**. Чтобы избежать создания бесконечных URL-адресов вида: «**списки**», «**детали**» и «**изменить**», маршрутизаторы DRF объединяют все URL-адреса, необходимые для данного вида, в одну строку для каждого ViewSet.

# Django Starter

Установка и подключение



# Django Starter

## План урока

1. Архитектурный стиль REST
2. История возникновения
3. Каким требованиям отвечает RESTful API
4. Преимущества RESTful API
5. Недостатки
6. Обзор инструментов для реализации REST в Django
7. Установка и подключение

# Проверка знаний

TestProvider.com



Проверьте как Вы усвоили данный материал на [TestProvider.com](http://TestProvider.com)

TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.

# Django Starter

Спасибо за внимание! До новых встреч!



Лазорык Михаил  
Software developer



# Информационный видеосервис для разработчиков программного обеспечения

