

 **ТЕХНОСФЕРА**

Фреймворки и API

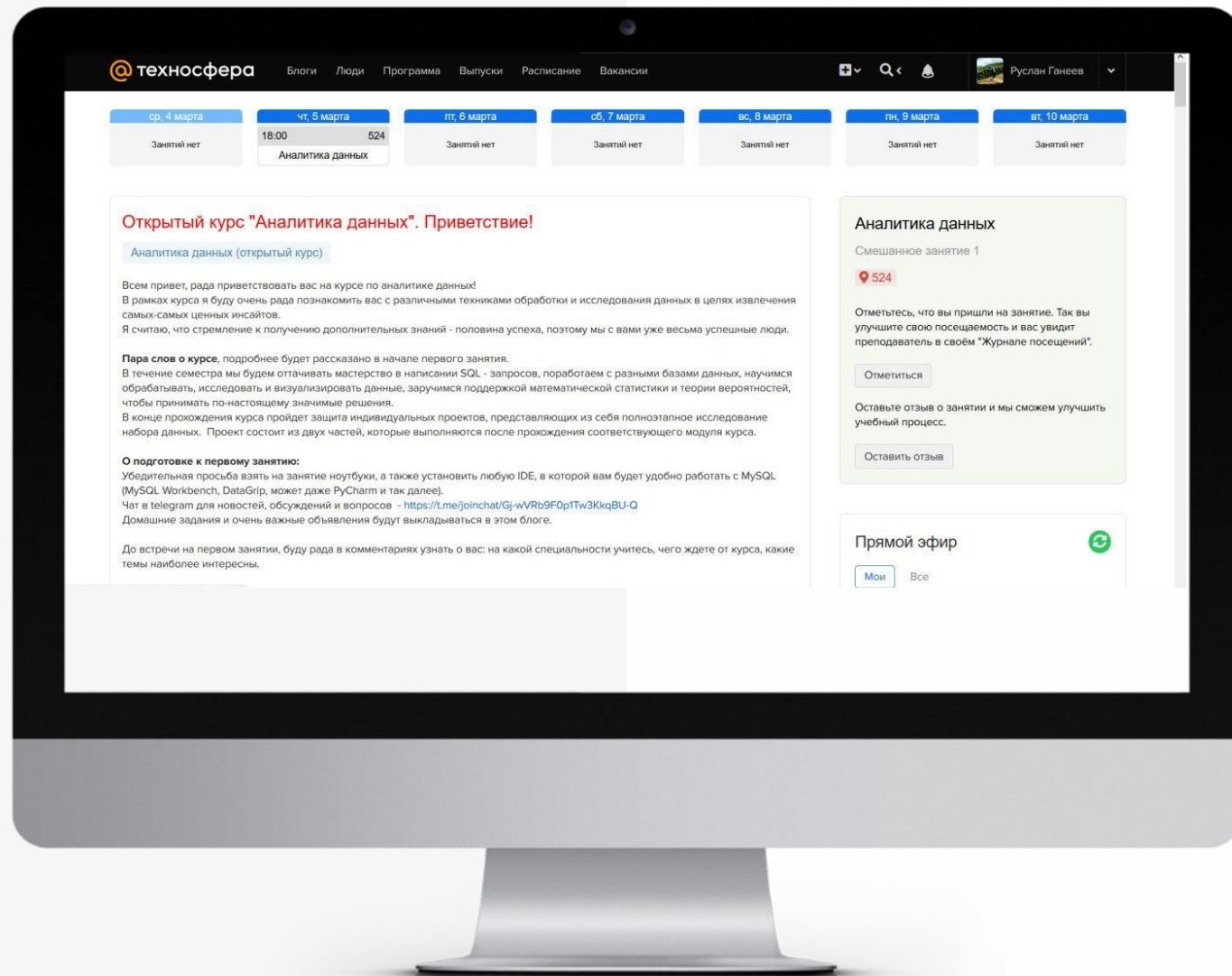
Антон Кухтичев





Содержание занятия

1. MVC
2. Django
3. API
4. Django Rest Framework



Напоминание отметиться на портале

Иначе плохо всё будет.



MVC



Основные типы запросов

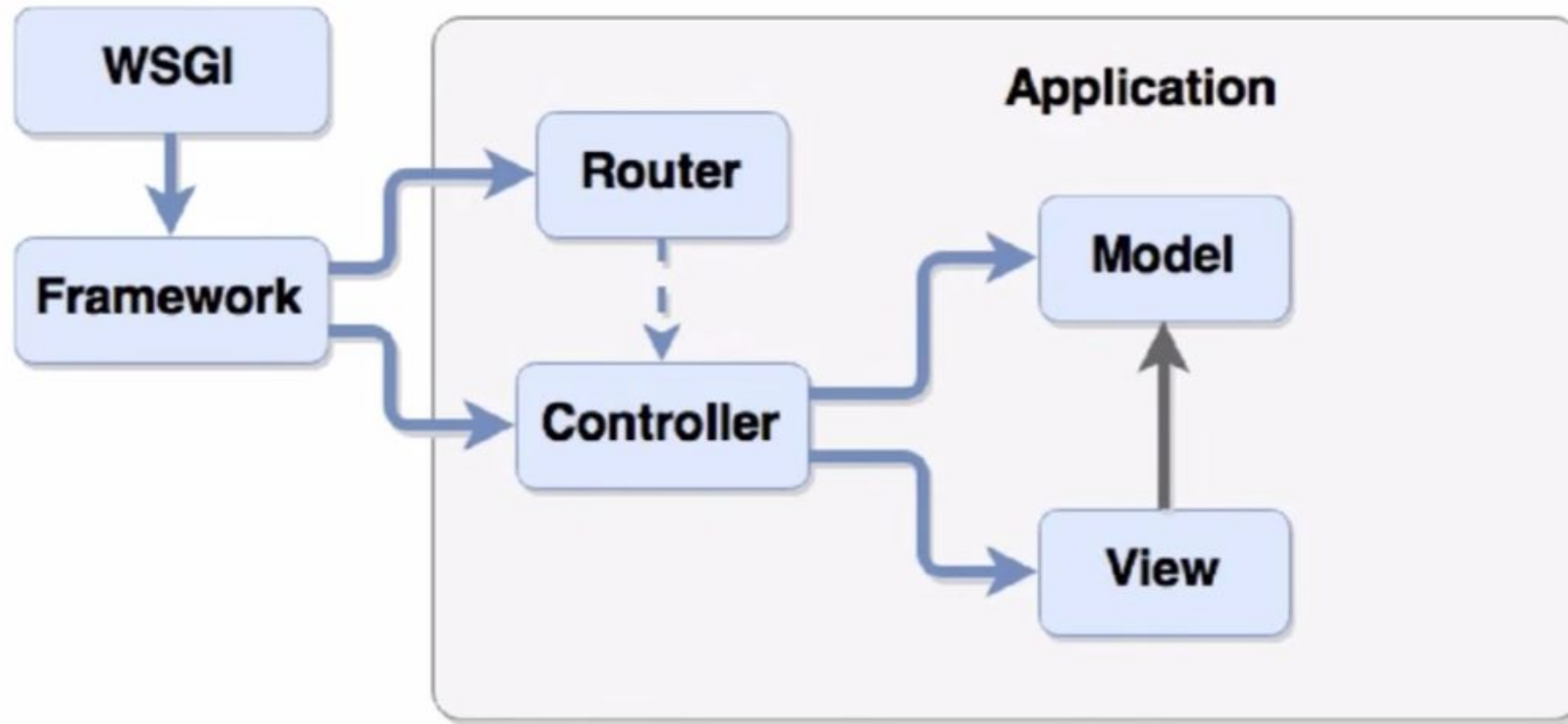
- Запросы статических документов;
- Запросы динамических документов;
- Отправка данных форм;
- AJAX-запросы;
- Запросы к API сайта;
- Персистентные соединения.



Основные задачи

- Маршрутизация URL;
- Парсинг заголовков и параметров запроса;
- Хранение состояния (сессии) пользователя;
- Выполнение бизнес-логики;
- Работа с базами данными;
- Генерация HTML-страницы или JSON-ответа.

Model-View-Controller





Роли компонентов MVC

- Router — выбор конкретного controller по URL;
- Model — реализация бизнес-логики приложения;
- Controller — работа с HTTP, связь controller и view;
- View — генерация HTML или другого представления.


Фреймворк

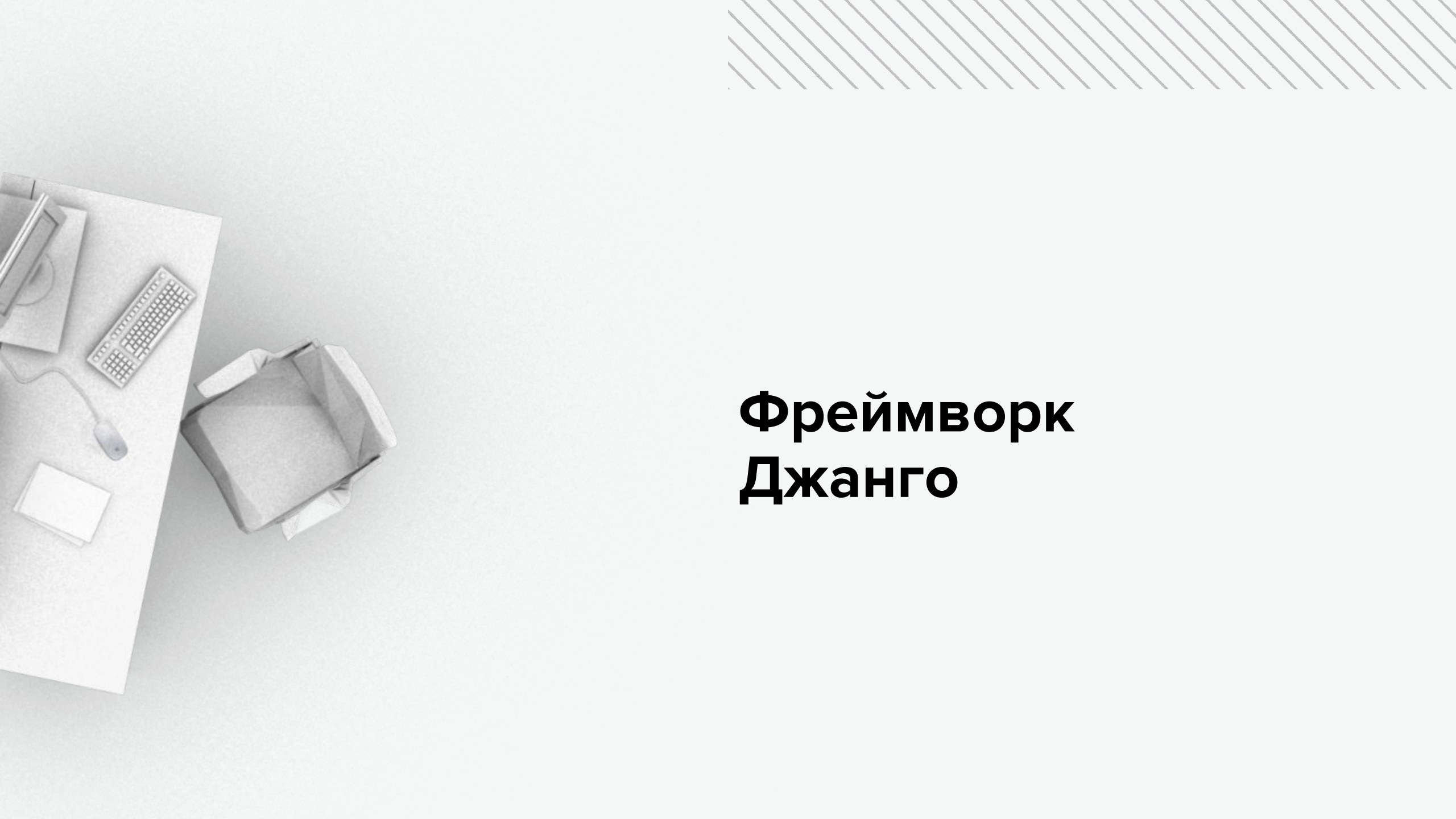
django





Плюсы фреймворков

- 
- Готовая архитектура;
 - Повторное использование кода;
 - Экономия ресурсов;
 - Участие в Open Source;
 - Проще найти программистов;
 - Проще обучать программистов.



Фреймворк Джанго



Установка



```
(venv) pip install Django==2.0
```

```
(venv) pip freeze > requirements.txt
```



Соглашение о именовании

MVC

Model

Router

Controller

View

Django

Model

urls.py

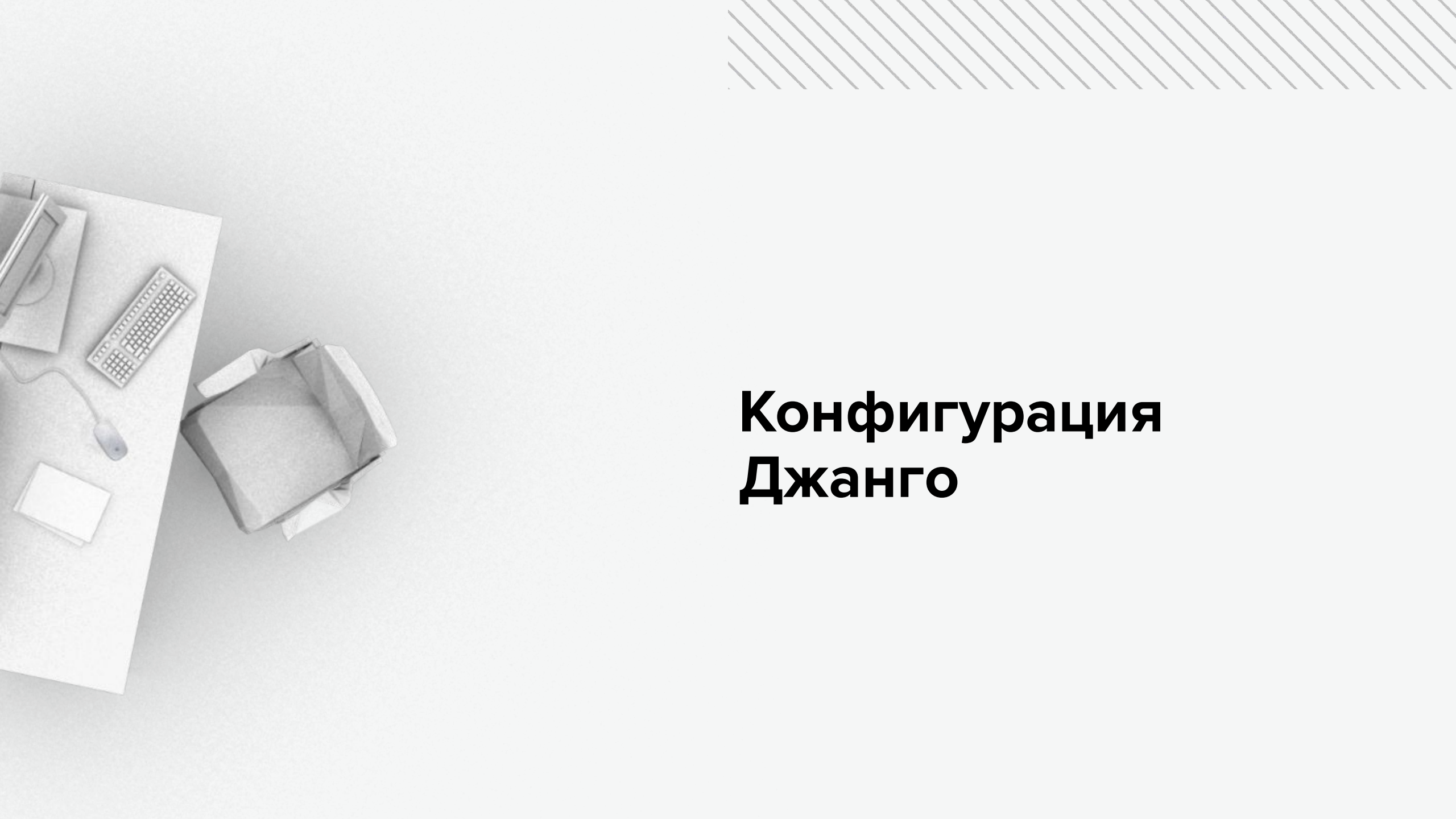
views

templates

Структура проекта

`django-admin startproject project` — создание проекта.

```
project
├── crm
│   ├── apps.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── manage.py
└── project
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```



Конфигурация Джанго

Конфиг — просто python-модуль

```
# project/project/settings.py
ROOT_URLCONF = 'project.urls'
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3')
    }
}
TEMPLATE_DIR = os.path.join(BASE_DIR, 'templates')
```


Пути в конфиге

Проблемы:

- Проект может быть развернут в любой директории;
- Несколько копий проекта на одном сервере.

Решения:

- Абсолютные пути в каждом конфиге
- Переменные окружения, `${PROJECT_PATH}`
- Относительные пути;

local_settings.py

```
# в конце project/settings.py
try:
    from .local_settings import *
except ImportError:
    pass
```



Маршрутизация в проекте

- Django начинает поиск с файла `ROOT_URLCONF`;
- Загрузив файл, Django использует переменную `urlpatterns`;
- Проходит по всем паттернам до первого совпадения;
- Если не найдено — 404.

Маршрутизация в проекте

```
# project/project/urls.py
from django.urls import path, re_path
from django.contrib import admin

urlpatterns = [
    path('', index, name='index'),
    re_path(r'^$', 'chats.views.index',
            name='index'),
    path('chats/', include('chats.urls')),
    path('admin/', admin.site.urls),
]
```

Маршрутизация в приложении

```
# project/chats/urls.py
from chats.views import chat_list
urlpatterns = [
    path('', chat_list, name='chat_list'),
    path('category/<int:pk>/',
         'chat_category',
         name='chat_category'),
    path('<chat_id>/', 'chat_detail', name='chat_detail'),
]
```

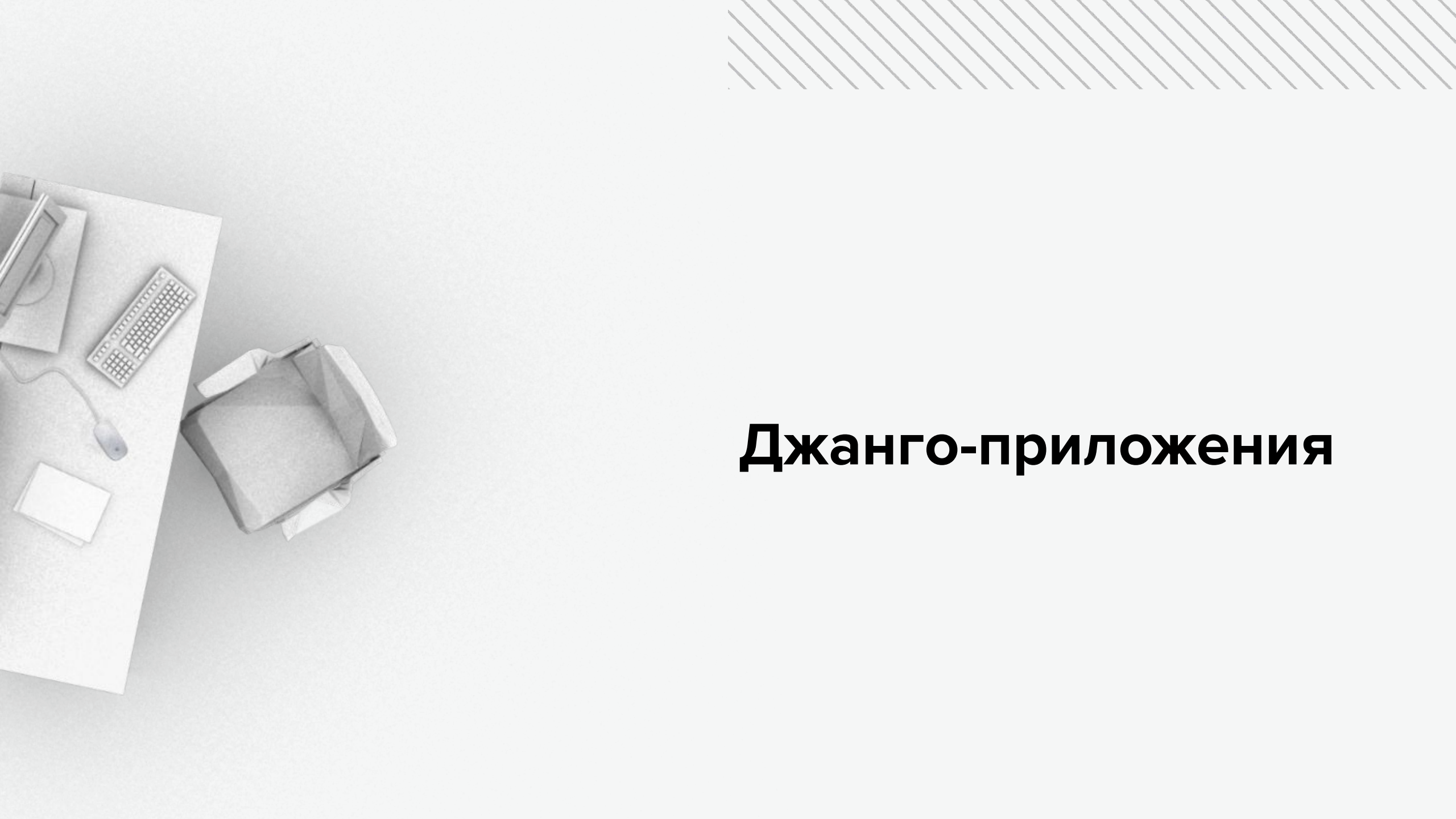


Особенности маршрутизации в Django

- Слеш (/) в начале роутеров не указывается;
- Можно указывать как имя, так и саму view-функцию;
- Роуты описываются с помощью регулярных выражений;
- Можно и нужно разносить роуты по приложениям;
- Можно и нужно создавать именованные роуты;
- Одно действие — один роут — один контроллер;

Reverse routing

```
from django.core.urlresolvers import reverse
reverse('index')
reverse('blog_category', args=(10, ))
reverse('blog_detail', kwargs={'pk': 2})
# в шаблонах
{% url 'blog_category' category_id %}
```



Джанго-приложения

Джанго-приложения

Приложения — способ распространения кода в Django-инфраструктуре. В случае, если вы не планируете публиковать ваш код, приложения - это просто способ логической организации кода.

`./manage.py startapp crm` — создание нового приложения с именем `crm`.
Нужно вызывать из директории проекта.



Джанго-приложения

Приложения — способ распространения кода в Django-инфраструктуре. В случае, если вы не планируете публиковать ваш код, приложения - это просто способ логической организации кода.

`./manage.py startapp crm` — создание нового приложения с именем `crm`.
Нужно вызывать из директории проекта.



Джанго-контроллеры

Контроллеры в Django — это обычные функции, которые:

- Принимают объект `django.http.HttpRequest` первым параметром;
- Возвращает объект `django.http.HttpResponse`.

Джанго-контроллеры

```
# project.blog.views
# запрос вида /blog/?blog_id=2
def blog_detail(request):
    try:
        blog_id = request.GET.get('blog_id')
        blog = Blog.objects.get(id=blog_id)
    except Blog.DoesNotExist:
        raise Http404
    return HttpResponse(blog.description,
                        content_type='text/plain')
```

Захват параметров из URL

```
# blog/urls.py
```

```
url(r'^category/(\d+)/$', 'category_view')
```

```
url(r'^(?P<pk>\d+)/$', 'post_detail')
```

Захват параметров из URL (2)

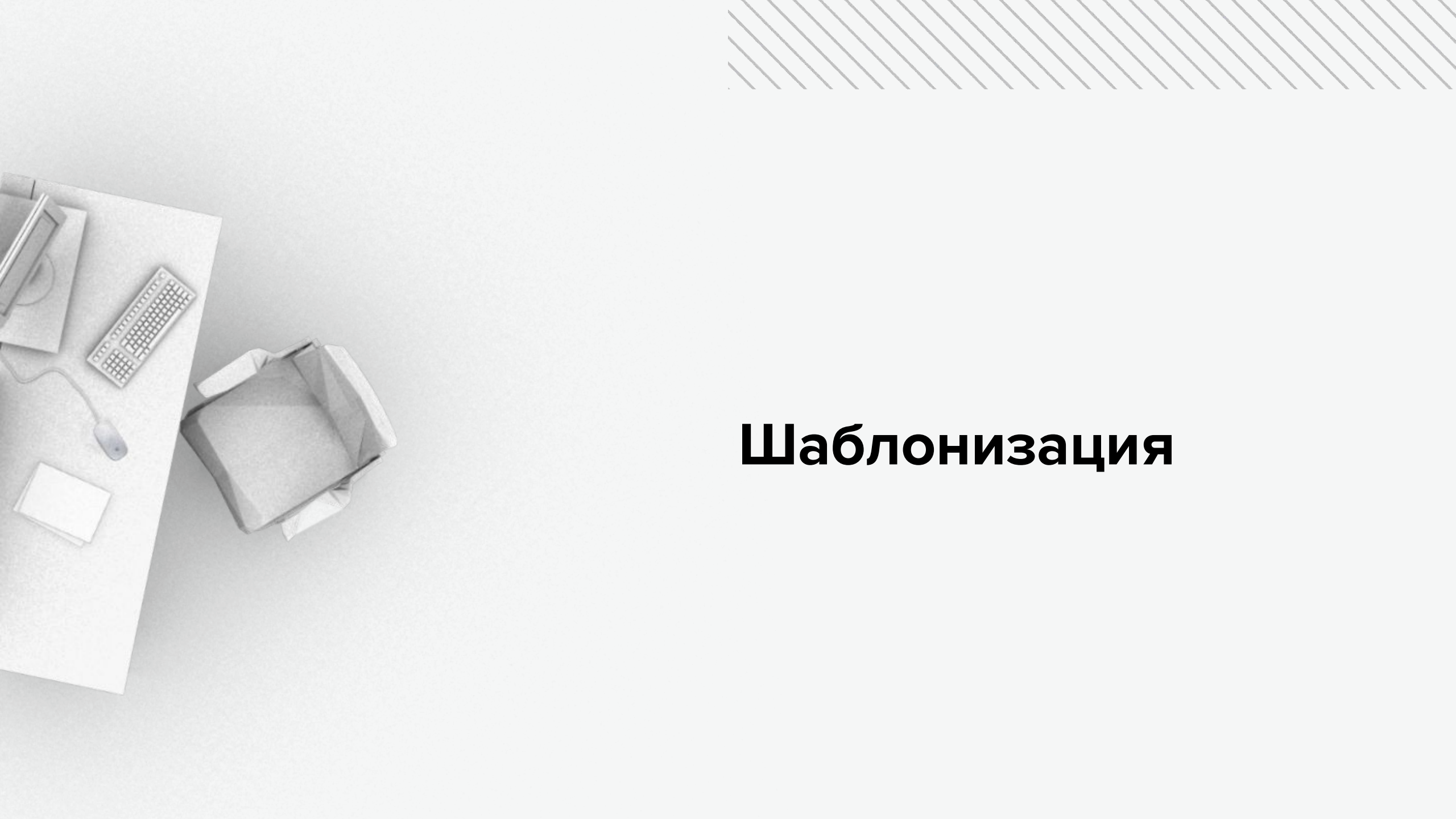
```
# blog/views.py
def category_view(request, pk=None):
    # вывести все посты

def post_details(request, pk):
    # вывести страницу поста

def category_view(request, *args, **kwargs):
    pk = args[0]
    pk = kwargs['pk']
```

Джанго-контроллеры

- `request.method` — метод запроса
- `request.GET` — словарь с GET параметрами
- `request.POST` — словарь с POST параметрами
- `request.COOKIES` — словарь с Cookies
- `request.FILES` — загруженные файлы
- `request.META` — CGI-like переменные
- `request.session` — словарь-сессия (*)
- `request.user` — текущий пользователь (*)



Шаблонизация

Неправильный подход

```
def header():  
    return "<html><head>...</head><body>"  
  
def footer():  
    return "</body></html>"  
  
def page1(data):  
    return header() + \  
        '<h1>' + data['title'] + '</h1>' + \  
        '<p>' + data['text'] + '</p1>' + \  
        footer()
```



Правильный подход

Необходимо отделить данные (**контекст**) от представления (**шаблона**). Для этого используются **шаблонизаторы**.

- Разделение работы web-мастера и программиста;
- Повторное использование HTML-кода;
- Более чистый python код.

Синтаксис шаблонов

```
<!-- templates/blog/post_details.html -->
<html>
  <head>...</head>
  <body>
    <h1>{{ post.title }}</h1>
    <p>{{ post.text }}</p>
    {% for comment in comments %}
      {% include "blog/comment.html" %}
    {% endfor %}
  </body>
</html>
```

Вызов шаблонизатора

```
# project/blog/views.py
from django.shortcuts import render, render_to_response

return render_to_response('blog/post_details.html', {
    'post': post,
    'comments': comments,
})

return render(request, 'blog/post_details.html', {
    'post': post,
    'comments': comments
})
```

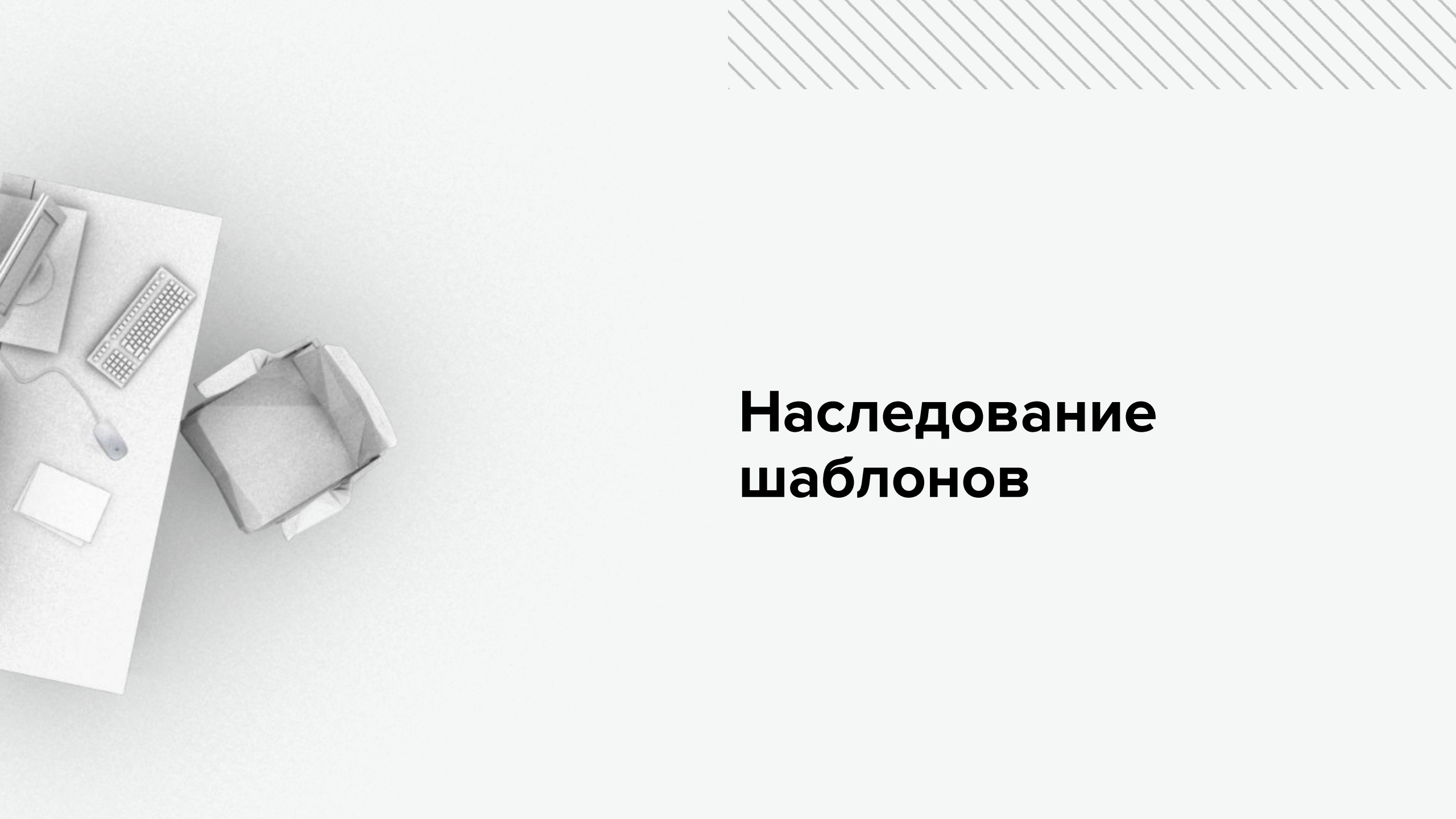
Возможности шаблонизатора

- `{% for item in list %}{% endfor %}` — итерация по списку
- `{% if var %}{% endif %}` — условное отображение
- `{% include "tpl.html" %}` — включение подшаблона
- `{{ var }}` — вывод переменной
- `{{ var | truncatechars:9 }}` — применение фильтров
- `{# comment #}`, `{% comment %}{% endcomment %}` — комментарии



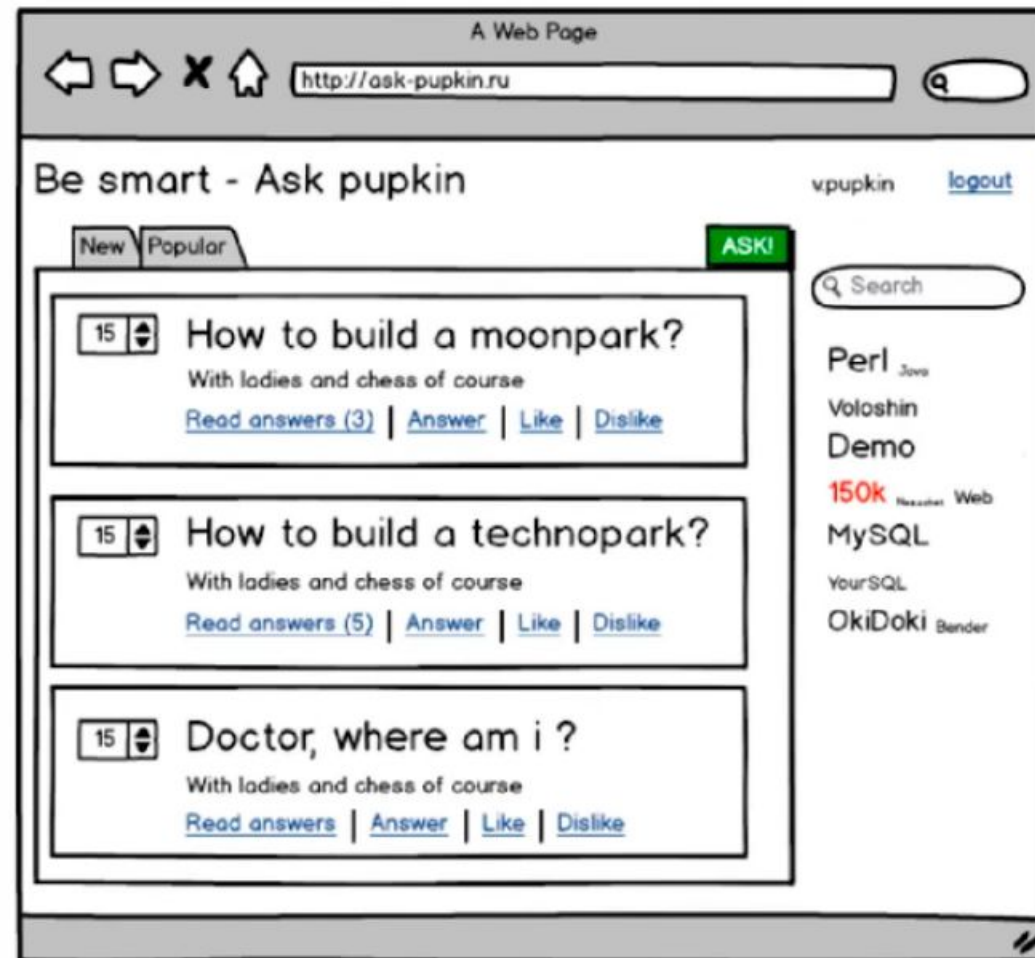
Особенности шаблонизатора

- Шаблоны автоматически экранируют HTML сущности;
- Шаблонизатор можно расширять своими фильтрами и тэгами.



Наследование шаблонов

Наследование шаблонов

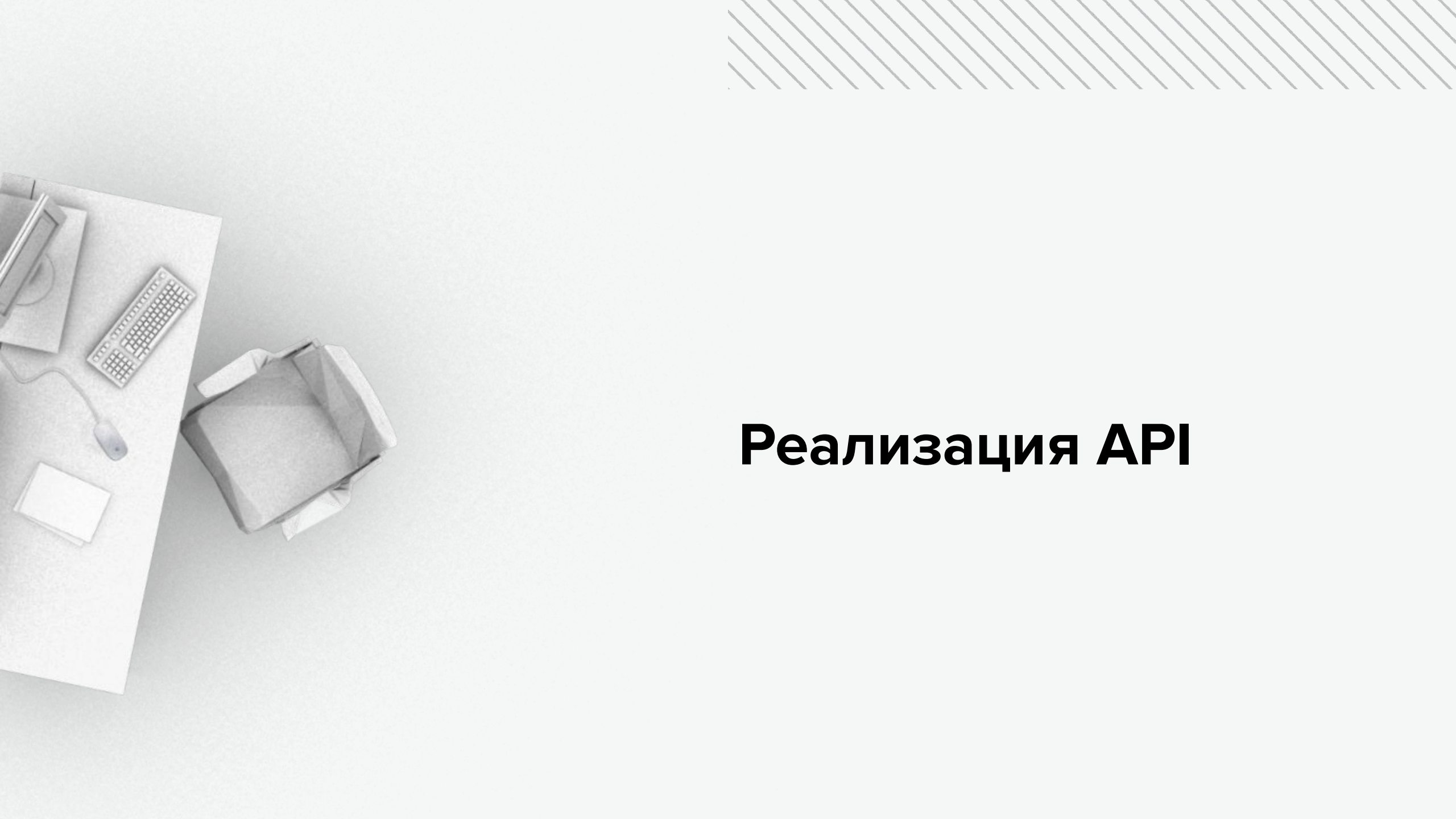


Базовый шаблон base.html

```
<!DOCTYPE HTML>
<html>
<head>
  <title>{% block title %}Q&A{% endblock %}</title>
  {% block extrahead %}{% endblock %}
</head>
<body>
  <h1>Вопросы и ответы</h1>
  {% block content %}{% endblock %}
</body>
</html>
```

Шаблон главной страницы


```
{% extends "base.html" %}
{% block title %}
    {{ block.super }} - главная
{% endblock %}
{% block content %}
    {% for obj in posts %}
        <div class="question">
            <a href="{{ obj.build_url }}">{{ obj }} </a>
        </div>
    {% endfor %}
{% endblock %}
```



Реализация API




Application programming interface (API)



Описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.



Виды совместимости приложений

- 
- Обратная совместимость — более новый код способен читать данные, записанные более старым;
 - Прямая совместимость — более старый код способен читать данные, записанные более новым.




REST (REpresentational State Transfer)

REST API подразумевает под собой простые правила:

- Каждый URL является ресурсом;
- При обращении к ресурсу методом GET возвращается описание этого ресурса;
- Метод POST добавляет новый ресурс;
- Метод PUT изменяет ресурс;
- Метод DELETE удаляет ресурс.



JSON-RPC



JSON-RPC (JavaScript Object Notation Remote Procedure Call — JSON-вызов удалённых процедур) — протокол удалённого вызова процедур, использующий JSON для кодирования сообщений.

JSON-RPC

Формат входного запроса:

- `method` — строка с именем вызываемого метода;
- `params` — массив объектов, которые должны быть переданы методу, как параметры;
- `id` — значение любого типа, которое используется для установки соответствия между запросом и ответом.

JSON-RPC

Формат ответа:

- `result` — данные, которые вернул метод. Если произошла ошибка во время выполнения метода, это свойство должно быть установлено в `null`;
- `error` — код ошибки, если произошла ошибка во время выполнения метода, иначе `null`;
- `id` — то же значение, что и в запросе, к которому относится данный ответ.

JSON-RPC

Пример запроса:

```
{ "method": "echo", "params": ["Hello JSON-RPC"], "id": 1 }
```

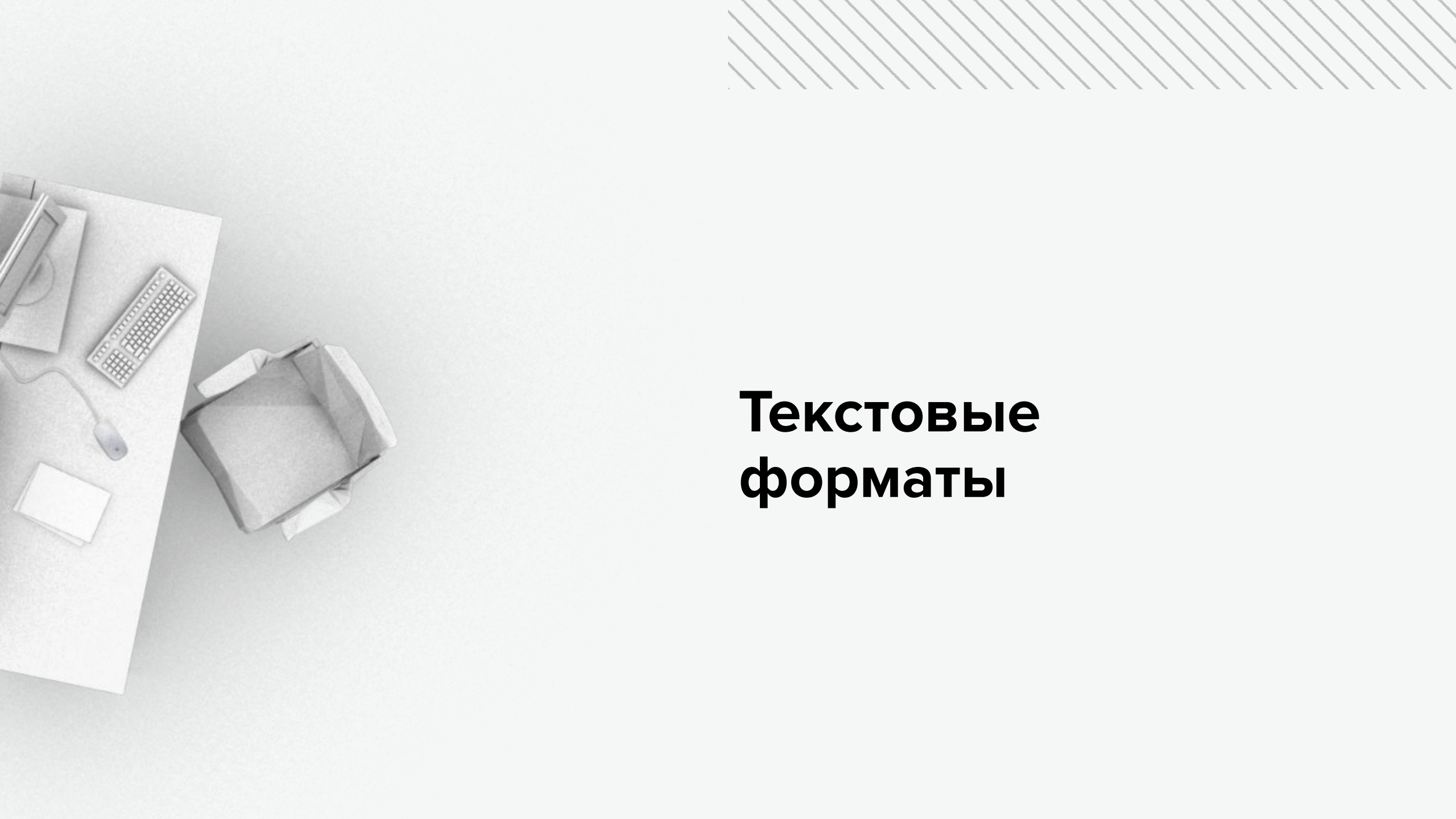
Пример ответа:

```
{ "result": "Hello JSON-RPC", "error": null, "id": 1 }
```



Форматы передачи данных

- Текстовые форматы (JSON, XML, CSV);
- Бинарный формат (Apache Thrift, Protocol Buffers);



Текстовые форматы

Формат CSV

- Каждая строка файла — это одна строка таблицы.
- Разделителем значений колонок является символ запятой (,)
- Однако на практике часто используются другие разделители.

1997,Ford,E350,"ac, abs, moon",3000.00

1999,Chevy,"Venture «Extended Edition»","",4900.00

1996,Jeep,Grand Cherokee,"MUST SELL! air, moon roof,
loaded",4799.00

Формат XML

XML (eXtensible Markup Language) – язык разметки, позволяющий стандартизировать вид файлов-данных, используемых компьютерными программами, в виде текста, понятного человеку.

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```



Формат XML

- Синтаксис XML избыточен;
- XML не содержит встроенной в язык поддержки типов данных;
- + Есть схема;
- + Человекочитаемый.

Формат JSON

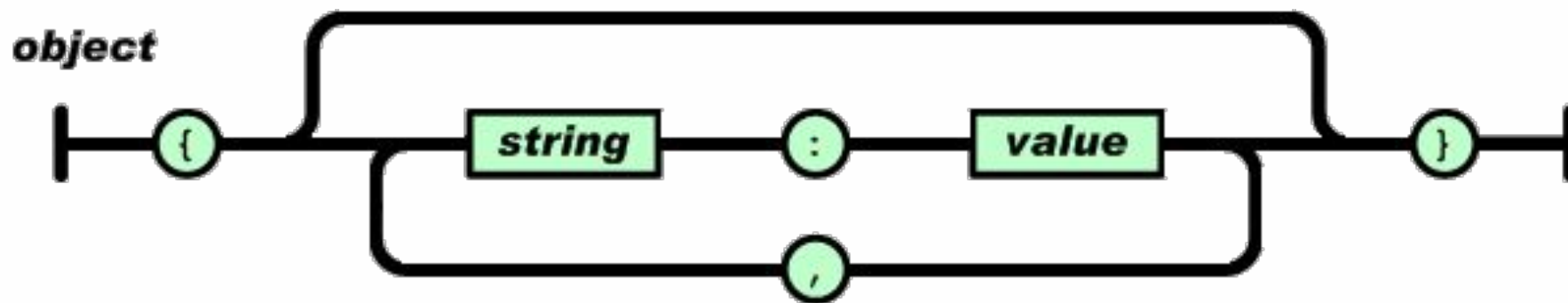
JSON (JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript.

```
{  
  "first_name": "Иван",  
  "last_name": "Иванов",  
  "phone_numbers": [  
    "812 123-1234",  
    "916 123-4567"  
  ]  
}
```


Формат JSON

JSON основан на двух структурах данных:

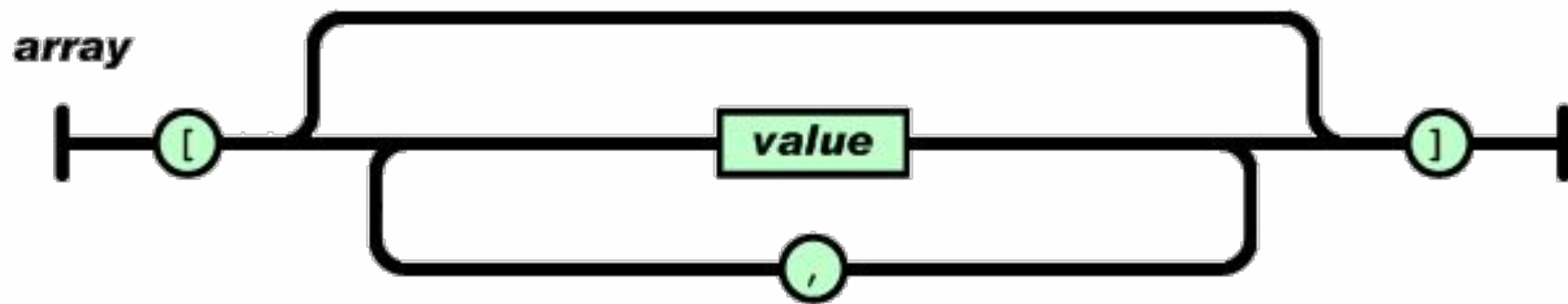
1. Коллекция пар ключ/значение. В разных языках, эта концепция реализована как объект, запись, структура, словарь, хэш, именованный список или ассоциативный массив;



Формат JSON

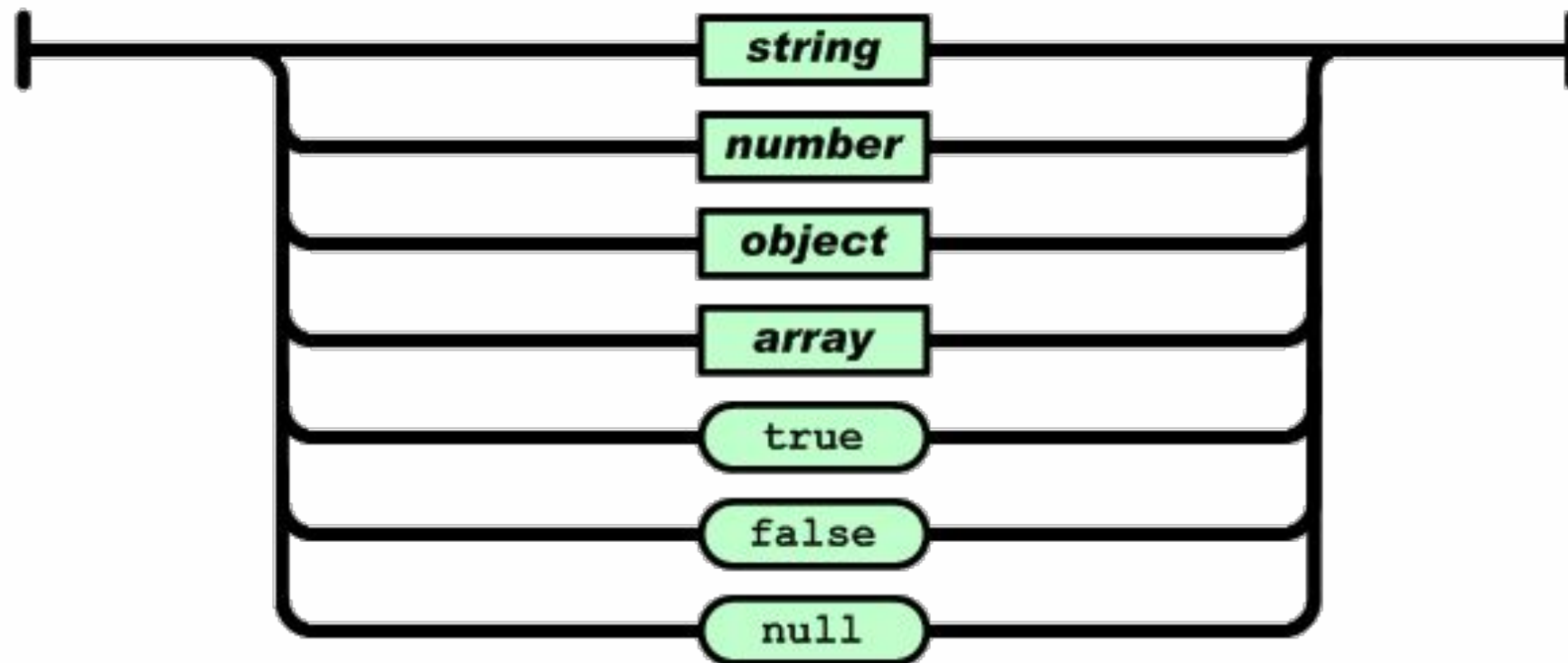
JSON основан на двух структурах данных:

2. Упорядоченный список значений. В большинстве языков это реализовано как массив, вектор, список или последовательность.




Формат JSON

value





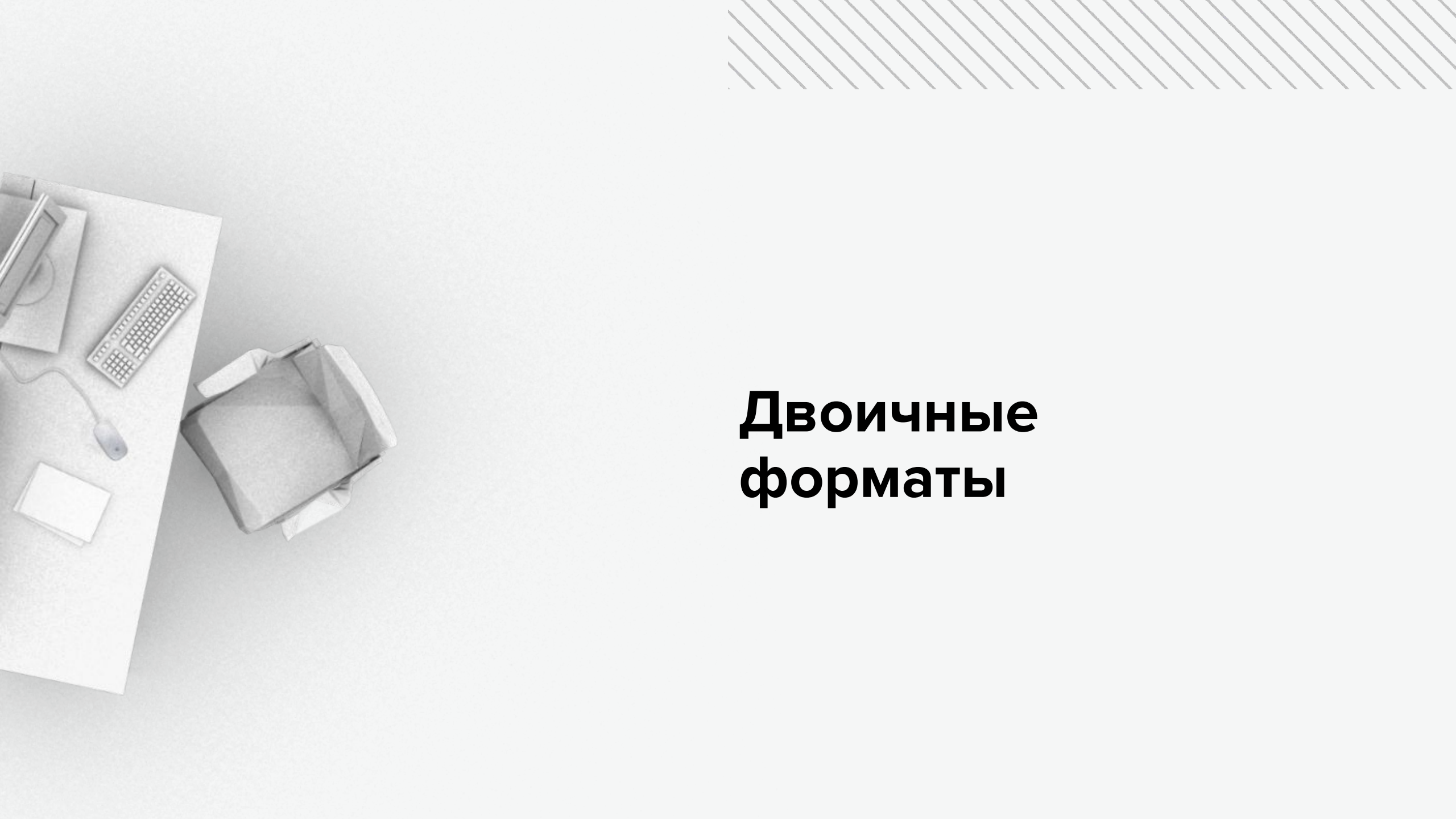
Преимущества JSON

- 
- Легко читается человеком;
 - Компактный;
 - Для работы с JSON есть множество библиотек;
 - Больше структурной информации в документе.



Преимущества JSON

- JSON - это формат данных - он содержит только свойства, а не методы;
- JSON требует двойных кавычек, которые будут использоваться вокруг строк и имен свойств;
- Вы можете проверить JSON с помощью приложения, такого как `jsonlint`;
- JSON может фактически принимать форму любого типа данных, который действителен для включения внутри JSON, а не только массивов или объектов.



Двоичные форматы



Преимущества двоичного кодирования

- Они могут быть намного компактнее различных вариантов “двоичного JSON”, поскольку позволяют не включать названия полей в закодированные данные;
- Схема - важный вид документа, вы всегда можете быть уверены в её актуальности;
- Пользователем языков программирования со статической типизацией окажется полезная возможность генерировать код на основе схемы, позволяющая проверять типы во время компиляции.

Protocol buffers

Protocol Buffers — протокол сериализации (передачи) структурированных данных, предложенный Google как эффективная бинарная альтернатива текстовому формату XML. Проще, компактнее и быстрее, чем XML.

```
message Person {  
    string user_name = 1;  
    int64 favorite_number = 2;  
    repeated string interests = 3;  
}
```

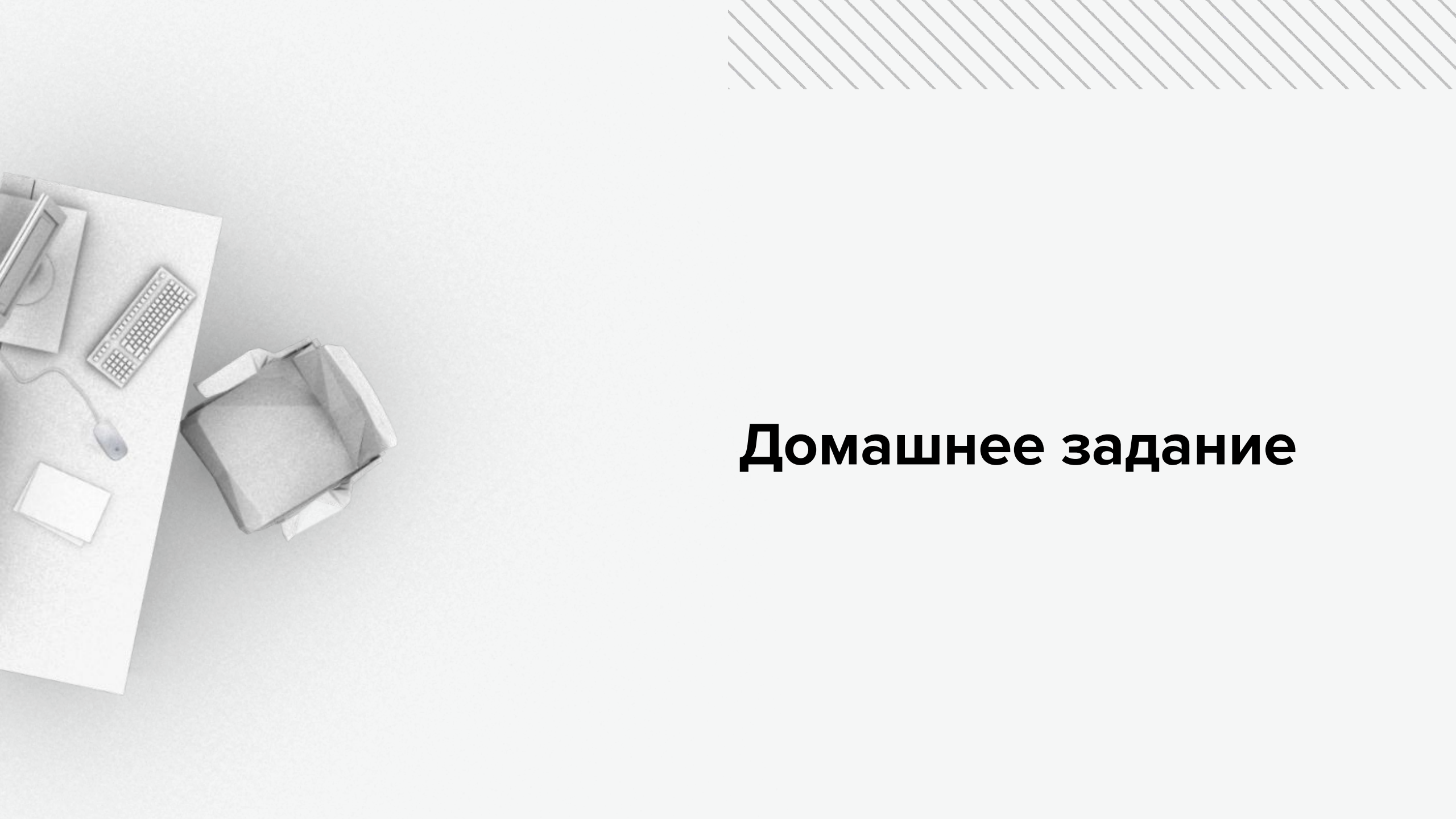



Django Rest Framework

Django Rest Framework

```
# Устанавливаем DRF
pip install djangorestframework

# Добавляем приложение в settings.py
INSTALLED_APPS = [
    ...
    'rest_framework',
]
```



Домашнее задание

Домашнее задание #6

- Создать Django-приложение;
- Создать класс-модель;
- Создать страницу для добавления/отображения/изменения/удаления записей;

Домашнее задание по уроку #6

Домашнее задание №6

#069



8

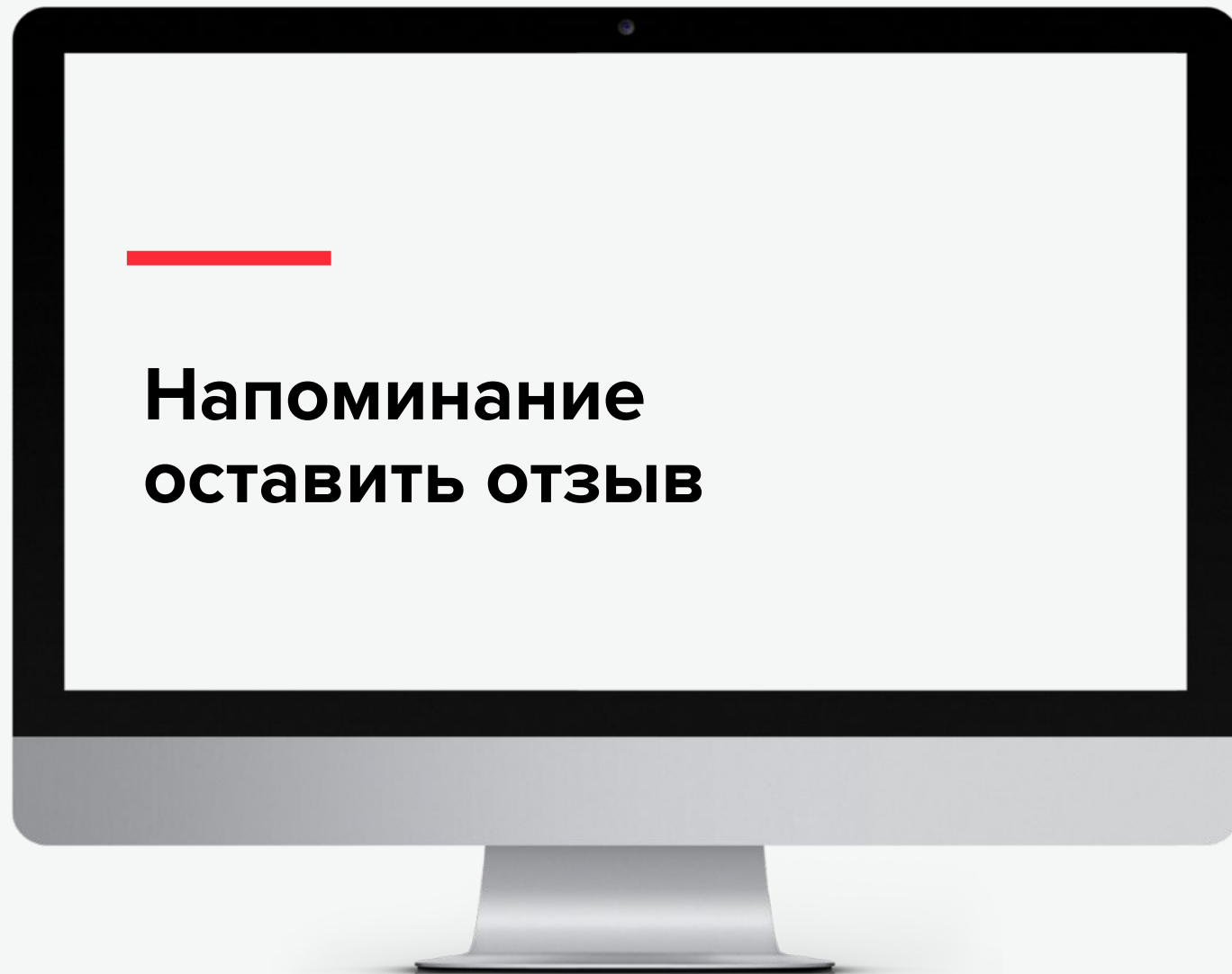
Баллов
за задание

Сроков нет, но вы держитесь

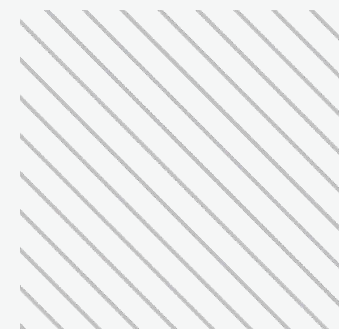
Срок
сдачи

- Высоконагруженные приложения.
Программирование
масштабирование поддержка |
Клеппман Мартин
- Google Protocol Buffers

Для саморазвития (опционально)
Чтобы не набирать двумя
пальчиками



**Напоминание
ОСТАВИТЬ ОТЗЫВ**



**СПАСИБО
ЗА ВНИМАНИЕ**

