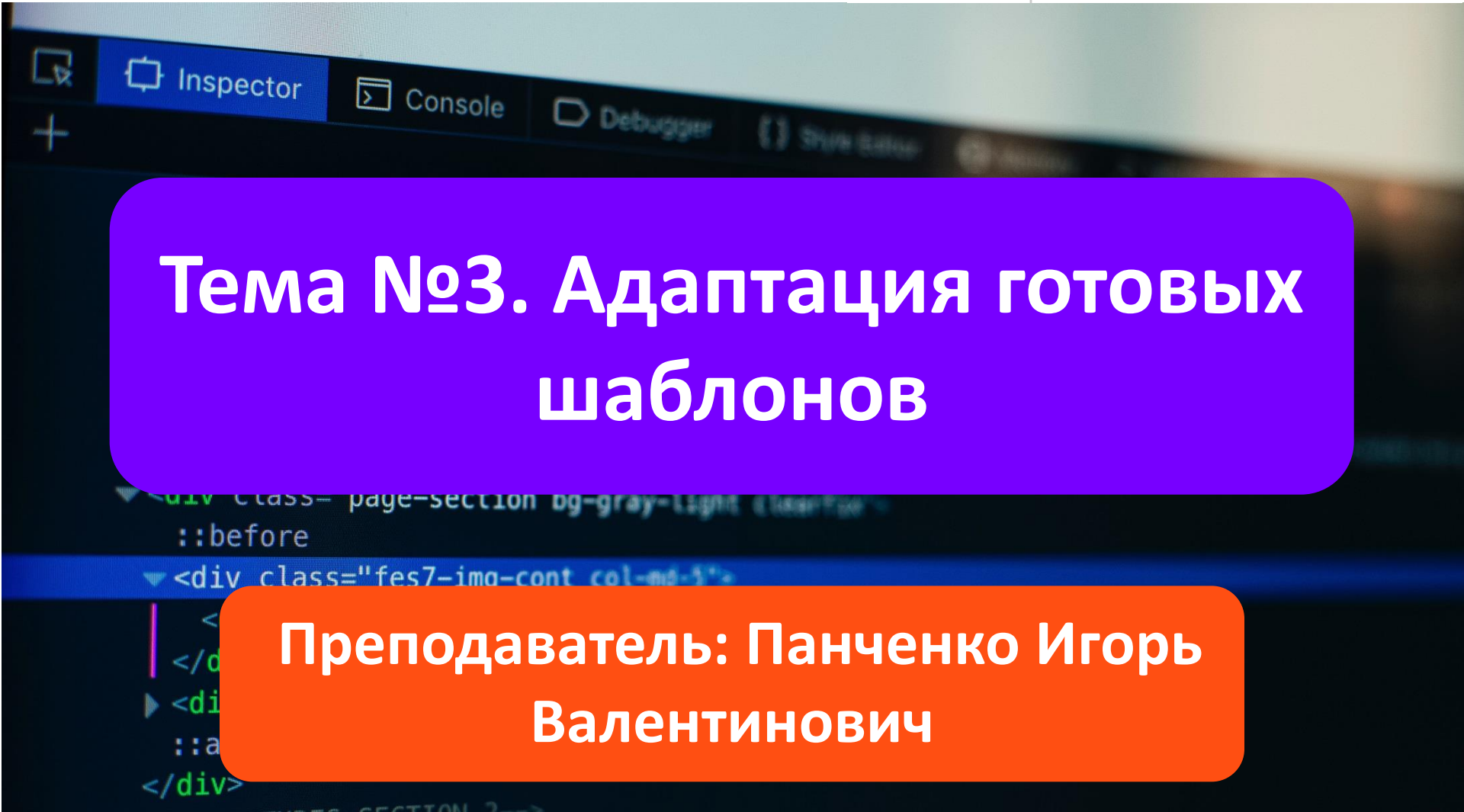
Python для продвинутых специалистов (Django)



Тема №3. Адаптация готовых шаблонов

Преподаватель: Панченко Игорь Валентинович

Готовый шаблон

Готовый шаблон

Зачастую в нашей работе требуется минимизировать затраты времени и итоговую стоимость проекта. Для достижения этих целей можно использовать готовые HTML+CSS шаблоны.

. Готовые шаблоны можно искать в интернете (google: ‘free css web templates’)

Free CSS | 3464 Free Website Templates, CSS Templates and Open Source Templates (free-css.com)

Поиск шаблона

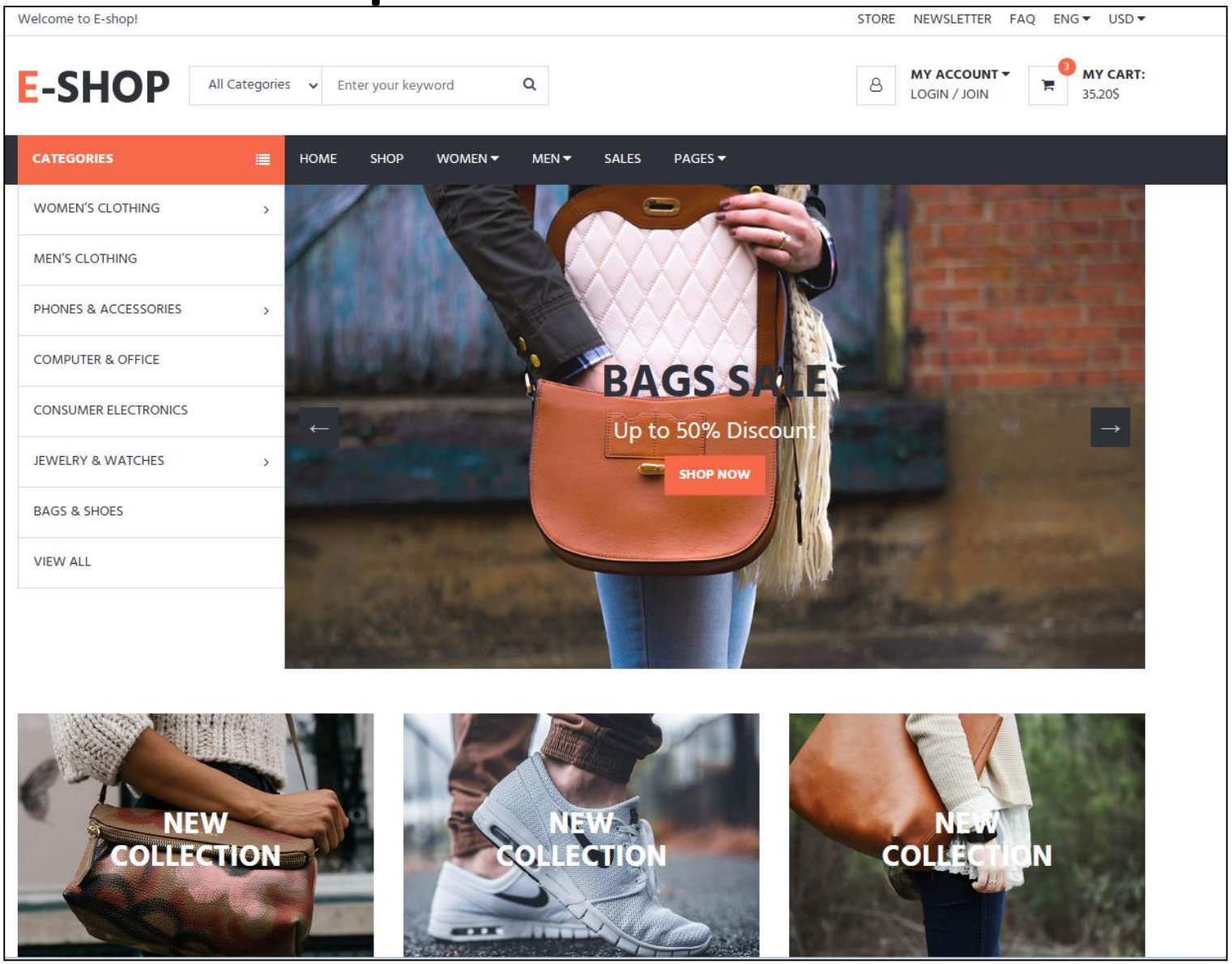
Шаблоны представляют собой набор CSS, HTML и JavaScript файлов, содержащих шаблоны типовых страниц, элементы оформления и скрипты.

Выберите шаблон, в котором есть все необходимые страницы: галерея, формы регистрации, контакты, и т.д.

Выбранный шаблон

Шаблон, выбранный автором вместе со всеми необходимыми файлами можно скачать тут: https://drive.google.com/drive/folders/1-tzxaaLBp1w649WPI9xjCckhXoLnnRqY?usp=share_link

Выбранный шаблон

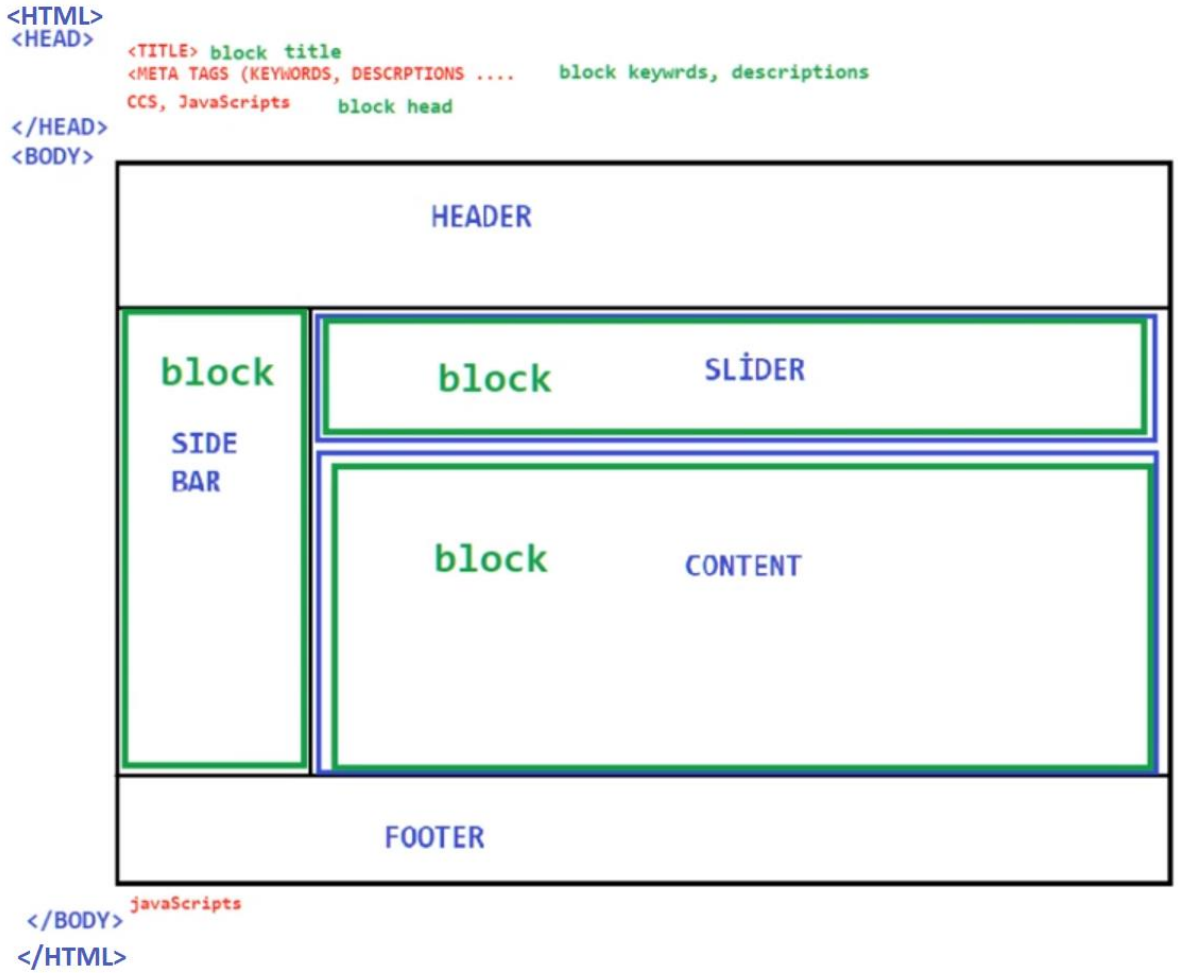


Структура базовой страницы

У нашего сайта будет такая структура →

Sidebar (Боковое меню), **Slider** (карусель) и основное поле **Content** будут оформлены в виде динамически изменяющихся **блоков**.

Header и Footer будут статичны (т.к. будут одинаковы на всех страницах)



Шаблонизатор Django

Язык шаблонизации

Шаблон – это текстовый документ в котором есть неизменные части (например – названия разделов сайта), и есть динамическое содержимое, отображающее информацию из БД, и адаптирующееся под уровень доступа к данным нашего пользователя.

Для реализации динамического вывода данных используется **язык шаблонизации**

Язык шаблонизации

Язык шаблонизации позволяет применять условия, циклы, вывод переменных прямо в HTML шаблонах. Шаблон страницы знает, как нужно отобразить данные, а сами данные не зависят от конкретного представления (View)

Django имеет два «встроенных» бэкенда (обработчика) шаблонизатора: DJangoTemplates и Jinja2

Настройка бэкенда шаблонизатора

В файле **settings.py** есть переменная Templates, В которой можно настраивать шаблонизаторы:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [...],
        },
    },
]
```

Возможности шаблонизатора

Оператор ветвления:

{% if “Условие” %}

{% else %}

{% endif %}

Оператор цикла

{% for «конструктор цикла» %}

{% endfor %}

Возможности шаблонизатора

Вывод значений из переменных контекста шаблона:

<h1> {{ title }} </h1>

 {{ descrirption }}

Контекст – это словарь, который передаётся в функцию render:

```
def index(request):
    context = {
        'title': 'Заголовок',
        'description': 'Описание',
                }
    return render(request,'main/content.html',context)
```

Шаблонные фильтры

При выводе значений вы можете применять шаблонные фильтры:

Они могут прибавлять значения, изменять регистр символов и т.д.

- **{{ value|add:”10”}}** к Value прибавится 10
- **{{ title|capfirst }}** то же, что capitalize() в python
- **{{ title|lower }}, {{ title|upper }}** – аналогично
- **{{ title|default:”Нет данных” }}** – записывает значение при отсутствии значения внутри переменной **title** и т.д.

Возможности шаблонизатора

Оператор ветвления:

{% if “Условие” %}

{% else %}

{% endif %}

Оператор цикла

{% for «конструктор цикла» %}

{% endfor %}

Механизм наследования/расширения

Базовый шаблон:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>{% block title %}
    {% endblock %}</title>
</head>
<body>
    {% block content %}
    {% endblock %}
</body>
</html>
```

Расширяемый шаблон:

```
{% extends "базовый_шаблон.html" %}

{% block title %}
Заголовок страницы
{% endblock %}

{% block content %}
<h1> Содержимое </h1>
<p> текст</p>
{% endblock %}
```

«Встраивание» - include

Встраиваемый шаблон: **sidebar.html**

```
<aside>
    <h1>Боковое меню</h1>
    <a>Ссылка 1</a>
    <a>Ссылка 2</a>
</aside>
```

Основной шаблон:

```
{% include 'sidebar.html' %}
<h1> HTML-Документ</h1>
```

Вы можете разработать боковое меню в отдельном файле и внедрять его во всех необходимых документах.

Использование статических файлов

Для коректного использования статических файлов (изображений, css-файлов и т.д.), нужно убедиться что в файле **settings.py** объявлены три константы:

- **STATIC_URL** – префикс URL-адреса для статических файлов (создаётся по умолчанию, необходимый минимум)
- **STATIC_ROOT** – путь к общей статической папке, формируемой при запуске команды **python manage.py collectstatic**
- **STATICFILES_DIRS** – список дополнительных (нестандартных) путей к статическим файлам, используемых для сбора и для режима отладки.

Использование статических файлов

Далее, необходимо создать каталог ProjectName/main**/static/main** .

В этом каталоге следует создать подкаталоги css и images, куда и следует поместить все необходимые статические файлы.

Подробная информация доступна в документации: How to manage static files (e.g. images, JavaScript, CSS) | Django documentation | Django (djangoproject.com)

Использование статических файлов

Далее – в основном файле Urls нужно прописать:

```
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    # ... the rest of your URLconf goes here ...
] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

И теперь мы можем использовать статические файлы:

```
{% load static %}
<img src="{% static 'my_app/example.jpg' %}" alt="My image">
```

Создание нового приложения

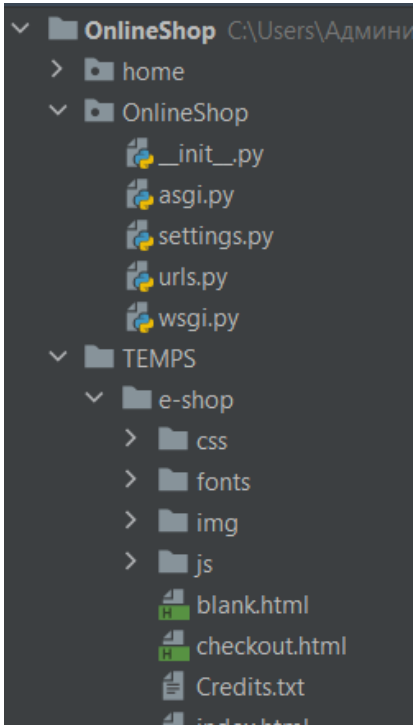
- Создайте новый проект:

django-admin startproject OnlineShop

- Создайте приложение home:

python manage.py startapp home

- Создайте внутри проекта папку TEMPS и перенесите папку с шаблоном



Создание базовой страницы

Для создания базовой страницы, нам необходимо перенести содержимое HTML файла шаблона index.html и назначить блоки с динамическим содержимым в соответствующих местах.

Создание базовой страницы

Создайте папку templates в приложении home, добавьте туда 2 файла: homebase.html, index.html

Для создания базовой страницы **(homebase.html)**, нам необходимо перенести содержимое HTML файла шаблона index.html и назначить блоки с динамическим содержимым в соответствующих местах.

Блоки - <head>

Нам потребуются блоки:

[1] title – для замены заголовка страниц

[2] keywords – для указания ключевых слов страницы

[2] description – для описания страницы

(Последние два – это метаданные, используемые поисковыми системами для знакомства со страницей)

[3] head – если потребуется программно добавить содержимое в блок head

Добавляем блоки <head>


```
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <!-- The above 3 meta tags *must* come first in the head; any other head content must come *after* these tags -->

  <title> {% block title %} Интернет-магазин {% endblock %}</title>
  <meta name = "keywords" content = "{% block keywords %} Home keywords {% endblock %}">
  <meta name = "description" content = "{% block description %} Home description {% endblock %}">
  <!-- Google font -->
  <link href="https://fonts.googleapis.com/css?family=Hind:400,700" rel="stylesheet">

  <!-- Bootstrap -->
  <link type="text/css" rel="stylesheet" href="css/bootstrap.min.css" />

  <!-- Slick -->
  <link type="text/css" rel="stylesheet" href="css/slick.css" />
  <link type="text/css" rel="stylesheet" href="css/slick-theme.css" />

  <!-- nouislider -->
  <link type="text/css" rel="stylesheet" href="css/nouislider.min.css" />

  <!-- Font Awesome Icon -->
  <link rel="stylesheet" href="css/font-awesome.min.css">

  <!-- Custom stlylesheet -->
  <link type="text/css" rel="stylesheet" href="css/style.css" />

  <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -->
  <!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
  <!--[if lt IE 9]>
      <script src="https://oss.maxcdn.com/html5shiv/3.7.3/html5shiv.min.js"></script>
      <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
    <![endif]-->
  {% block head %} {% enblock %}
</head>
```

Блоки - <body>

Нам потребуются блоки:

sidebar – для работы с боковым меню

slider – для работы с «каруселью»

body – для добавления содержимого в основном блоке страницы

foot– для работы с «подвалом»

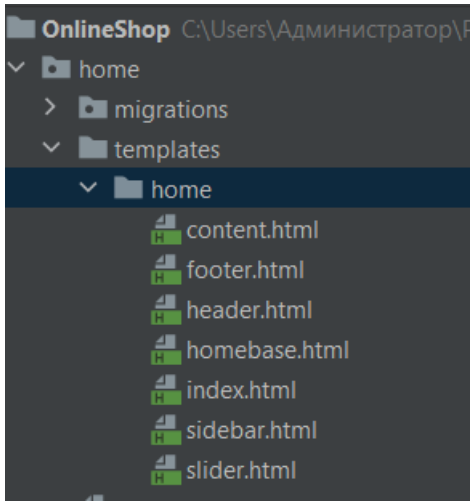
```
<body>
   {% include ‘header.html’ %}
   {% block sidebar %} {% endblock %}
   {% block slider %} {% endblock %}
   {% block body %} {% endblock %}

   {% block foot %} {% endblock %}
</body>
```

Header.html

В шаблоне очень много строк. Для простоты работы, разделим содержимое шаблона на отдельные файлы. Создайте файлы с названиями, соответствующими блокам:

- header.html
- sidebar.html
- slider.html
- content.html
- footer.html



Header.html

В шаблоне комментариями обозначены отдельные фрагменты страницы, перенесите их в соответствующие файлы:

<!-- HEADER --> - в header.html

<!-- NAVIGATION --> - sidebar панель

<!-- HOME --> - slider (Карусель)

<!-- section -->- в файл content.html

<!-- /FOOTER --> и **<!-- jQuery Plugins -->** в footer.html

extendes

Расширьте страницу **index.html** от homebase.html и переопределите содержимое отдельных блоков страницы:

```
{% extends ' home/homebase.html' %}
{% load static %}
{% block title %} Главная страница {% endblock %}
{% block description %} Описание {% endblock %}
{% block keywords %} Ключевые слова {% endblock %}

{% block sidebar %} {% include ‘home/sidebar.html' %} {% endblock %}
{% block slider %} {% include ' home/slider.html' %} {% endblock %}
{% block content %} {% include ' home/content.html' %} {% endblock %}
{% block foot %} {% include ' home/footer.html' %} {% endblock %}
```

urls

В **основном** файле **urls.py** нужно обработать соответствующие ссылки:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('', include('home.urls')),
    path('home/',include('home.urls')),
    path('admin/', admin.site.urls),
]
```

urls

В **home/urls.py:**

```
from django.urls import path

from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

В приложении home

```
from django.urls import path

from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

home/urls.py

```
from django.shortcuts import render

def index(request):
    return render(request,‘home/index.html')
```

me/views.py

Запуск сервера

Зарегистрируйте в Settings.py → **INSTALLED_APPS** приложение home.

Укажите путь к шаблонам:

TEMPLATES: 'DIRS': [os.path.join(BASE_DIR, 'templates')],

Запустите сервер и оцените результат

Запуск сервера

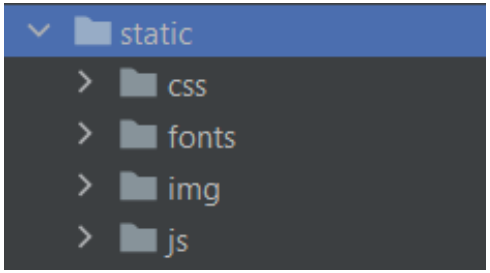
Если не возникло ошибок из-за невнимательности, то вы увидели нечто подобное →

Проблема

Сравните HTML – код шаблона и того, что отрисовалось после запуска сервера. HTML – код абсолютно идентичный.

Мы видим такую страницу, так как мы не добавили статические файлы к нашему проекту. Нам нужно перенести их в папку **static** внутри приложения home:

home/static/



Добавляем static files

Подробную информацию можно найти тут: [How to manage static files (e.g. images, JavaScript, CSS) | Django documentation | Django (djangoproject.com)]

Использовать статические файлы нужно так:

```
{% load static %}
<img src="{% static 'my_app/example.jpg' %}" alt="My image">
```

Добавляем static files

В **homebase.html** допишите **{% load static %}**

```
<meta name = "keywords" content =
<meta name = "description" content
{% load static %}
```

Нужно подключить статические файлы:

```
<link type="text/css" rel="stylesheet" href="{% static 'css/slick.css' %}" />
```

Или так:

```
<link type="text/css" rel="stylesheet" href="{% static '' %}css/slick-theme.css" />
```

И т.д.

Добавляем static files

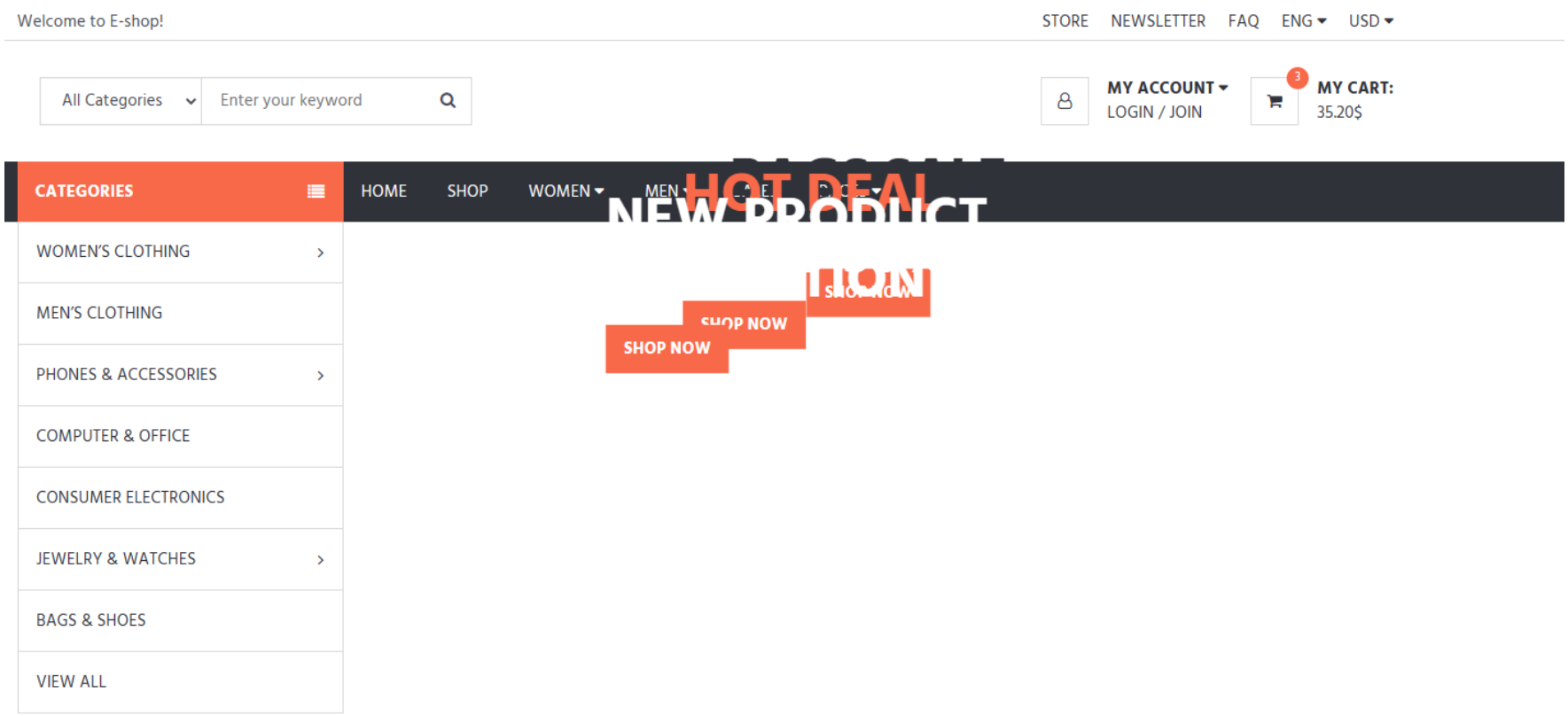
Если не загружаются элементы каких-то HTML шаблонах – пропишите в каждом из них в блоке head под тегами Meta Команду:

{% load static %}

Если какой-то из элементов не отображается до сих пор, откройте исходный код страницы в браузере и прокликайте все ссылки. Так вы сможете найти элемент, вызывающий ошибку, или не отображающийся корректно

Проверка Static

Обновите страницу и проверьте работу



Проверка Static

Как видите – работает не всё. Добавьте
{% static ‘’ %} к JS Элементам в footer.html

Также, нам необходимо добавить пути к изображениям во всех тегах img:

<img src="./img/logo.png" alt="">

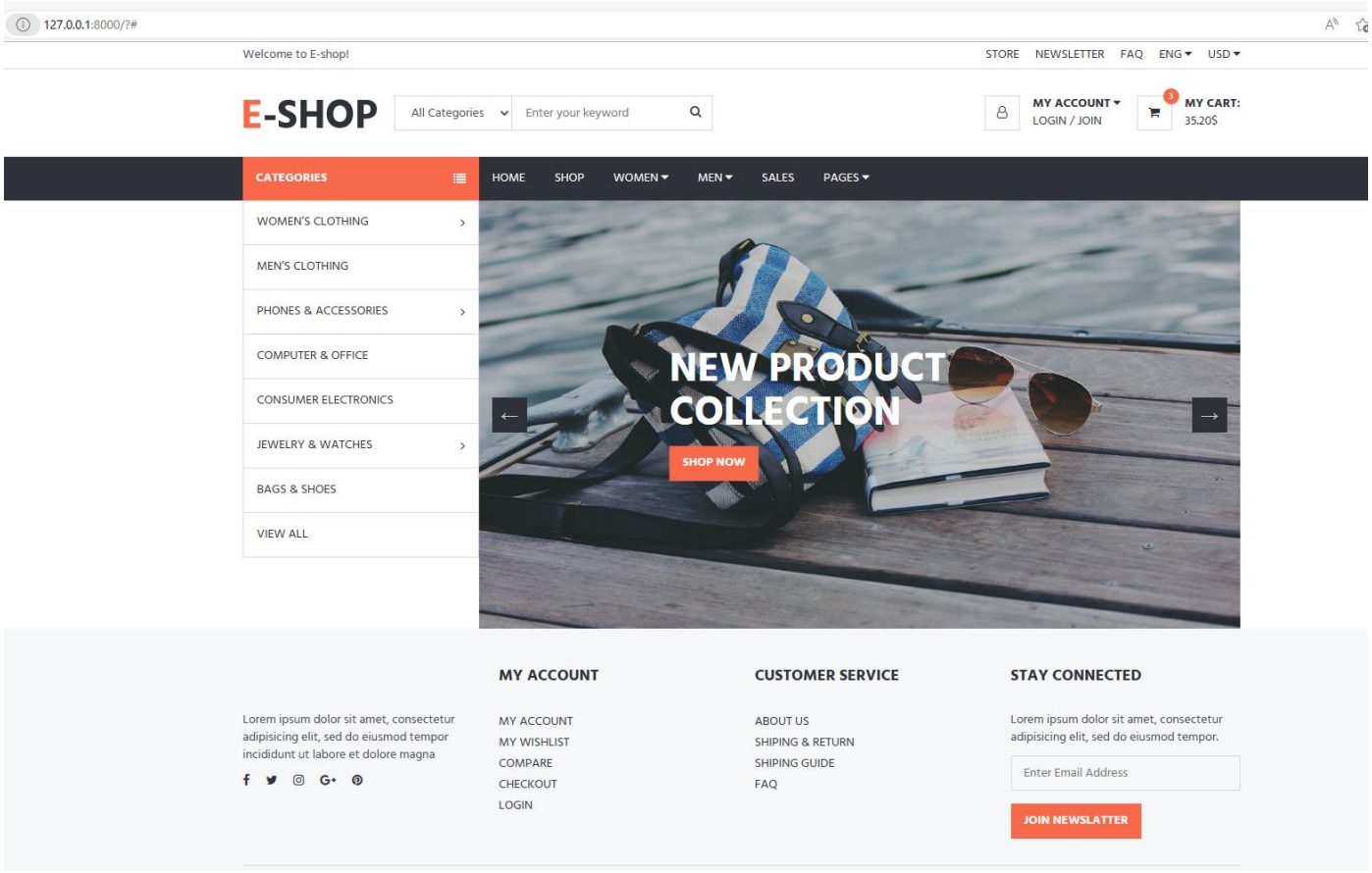
Заменить на:

<img src="{% static '' %}./img/logo.png" alt="">

Для быстрой замены можно воспользоваться комбинацией клавиш Ctrl+R

Проверка Static

После замены всё должно работать отлично



Возможные проблемы:

- Не работают картинки – проверяйте все теги img, пути к изображениям
- Не работают JS Элементы (карусель, вложенные меню и т.д.) – проверьте правильность путей в файле footer.html и {% include footer %} в файле index.html

Итоги

Подобным образом происходит встраивание всех прочих шаблонов в Django.

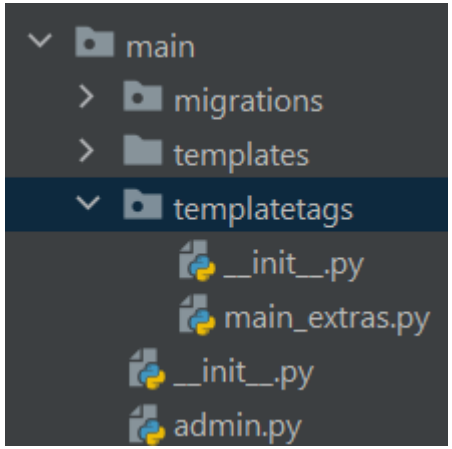
Если вам не подошёл этот шаблон – найдите другой доступный шаблон и повторите все процедуры: разбиение на отдельные элементы страницы, создание общего шаблона, настройки, ссылки

Пользовательские теги

Если вам не хватает возможностей встроенных тегов и фильтров – вы можете определить свои. Документация по созданию тегов есть по этой ссылке.

Для практики – создадим свой фильтр и тег.

В приложении **main** создайте директорию **templatetags**, внутри которой создайте 2 файла: **__init__.py** и **main_extras.py**



main_extras.py

Такой фильтр составляет буквы строки лесенкой. Для регистрации тега используется декоратор **@register.filter**

Для указания того, что данный фильтр применяется к строкам, используется декоратор **@stringfilter**

```python
from django import template
from django.template.defaultfilters import stringfilter
register = template.Library()

@register.filter(name='ladder')
@stringfilter
def ladder(value):
    """Составляет символы ЛеСеНкОй"""
    result = ''
    for i in range(len(value)):
        if i % 2 == 0:
            result += value[i].upper()
        else:
            result += value[i].lower()
    return result
```

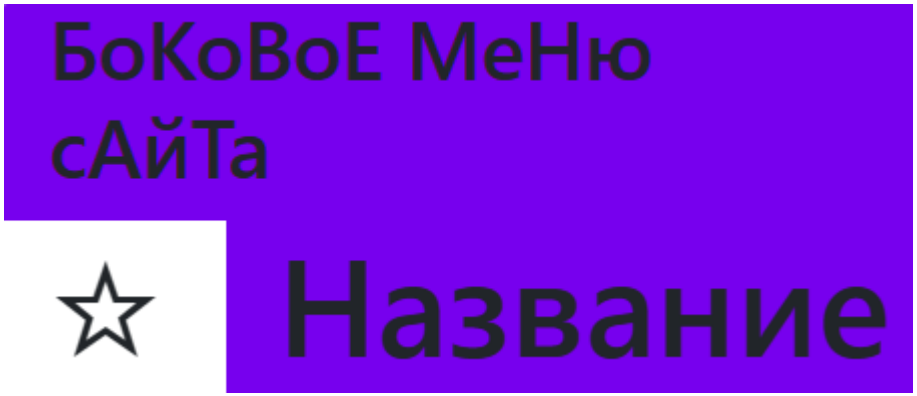
Ladder demo:

Загружаем пользовательские теги, применяем их к строке.

```
<!DOCTYPE html>
{% load main_extras %}

    <h4>{{ 'Боковое меню сайта'|ladder }}</h4>
```

Результат:

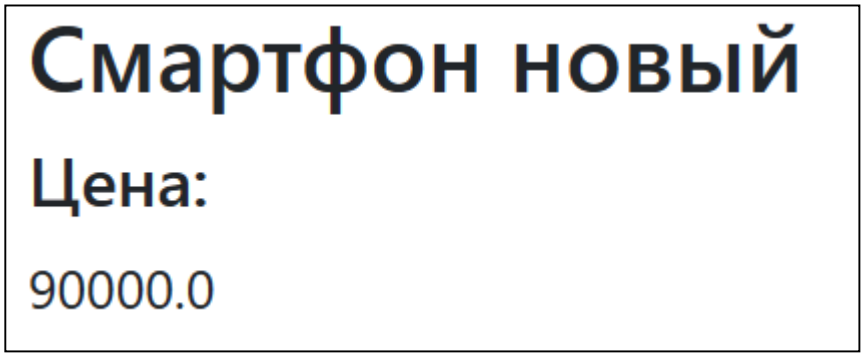


Пример тега, принимающего аргументы

Данный фильтр пересчитывает цену на товар, в зависимости от указанного значения скидки:

```
@register.filter(name='discount')
def discount(value, percent):
    return value*(100-int(percent))/100
```

```
<h3> Смартфон новый </h3>
<h5>Цена:</h5> <p>{{ 100000|discount:"10" }}</p>
```



Пользовательские теги

How to create custom template tags and filters | Django documentation | Django (djangoproject.com)

#16. Пользовательские теги шаблонов. Декораторы simple_tag и inclusion_tag | Уроки по Django 4 (proproprogs.ru)

Ссылки на материалы

- Создание первой веб-страницы | WebReference
- Селекторы CSS - Изучение веб-разработки | MDN (mozilla.org)
- Комбинаторы - Изучение веб-разработки | MDN (mozilla.org)
- Что такое Bootstrap и зачем он нужен? - ИТ Шеф (itchief.ru)