Создание Web-сервисов на Python

Московский физико-технический институт и Mail.Ru Group

Неделя 4. Часть 2

Содержание

1. HTML

1.1. Основы CSS

CSS выглядит следующим образом и имеет такой синтаксис. Вначале указывается некий селектор (что такое селектор, мы посмотрим немного позже), а в фигурных скобках указываются стили, которые необходимо применить к элементам, которые соответствуют этому селектору. Комментарии в CSS указываются как /* комментарии и закрыть */

```
.mid-play {
    padding:13px 0px 0px 13px;
}
p.inner-play a {
    color:#3c3c3c;
    text-decoration: underline;
}
.big-top {
    background-image: url(/img/pc/220_130_top.gif);
}
/* комментарии: селектор { имя_стиля1: значение1; } */
```

Рис. 1

Где могут быть заданы стили? Это стили, которые встроены в сам браузер; во внешнем файле, это с помощью тэга link внутри тэга head, который мы рассматривали в прошлом видео; в самом HTML — тэг style. И стили могут быть привязаны непосредственно к самому HTML элементу — к тэгу.

1.1.1. Классы и идентификатор

Идентификатор — это собственно id элемента. Он указывается с помощью атрибута id. Элемент с таким id может быть единственным на страницу.
Класс указывается в атрибуте class, их может быть несколько, и может быть несколько элементов с таким классом.

1.1.2. Селекторы

Давайте теперь рассмотрим селекторы и как их описывать. Селекторы — это некие правила, по которым тестируется, применять ли стили к тэгу или нет.
Универсальный селектор соответствует звездочке, то есть стили, которые указаны в этом селекторе, применить на все тэги.

```
* { margin: 0px; padding: 0px; border: 0px; }
```

Селектор по имени тэга. То есть все стили, которые указаны в этом селекторе, будут применены к тэгу p в нашем случае.

```
p { margin-top: 10px; }
```

Селектор по классу. Он начинается с точки, например, .btn — это будет означать, что данные стили будут применены ко всем элементам, в которых есть класс button.

```
.btn { border: solid 1px gray; }
```

И селектор по id. Это будет означать, что данные стили будут применены только к элементам, у которых id — userpic. Такой селектор начинается с решетки.

```
#userpic { padding: 10px }
```

Контекстные селекторы, или вложенные: div.article a будет означать элементы a, которые вложены в контейнер div с классом article. Дочерние, например, a > img будет означать, что стиль будет применен к всем тэгам img, которые находятся внутри a. Например, группировка — h1, h2 будет означать, что данный стиль нужно применить как к тэгам h1, так и к тэгам h2.
То же самое действует и для классов. То есть если через запятую указать селекторы по классам, то эти стили будут применены и к тому и к другому классу.
Также существуют псевдоклассы. Например, a:visited будет соответствовать ссылкам, по которым мы уже щелкнули; a:link — этот селектор будет соответствовать ссылкам, по которым еще не щелкали; div:hover будет соответствовать элементу div, на который мы навели мышкой; input:focus будет означать тот input, который является активным, то есть, например, мы на него нащелкали табом; li:first-child будет означать первый элемент в списке; li:last-child будет означать последний элемент в списке.
Псевдоэлементы — это виртуальные элементы сразу или непосредственно до элемента. Например, это необходимо, когда у нас есть элемент с id el, то есть el. И до него нам нужно вставить какую-то иконку или после него. Например, вы нажимаете оплату, и там будет иконка денег. В таком случае используются псевдоэлементы.
В случае, если два разных стиля конфликтуют между собой, применяется тот стиль, у которого большая специфичность у селектора. Давайте теперь рассмотрим, как же эта специфичность считается. Если в селекторе указан id, то это сразу специфичность 100. Если в селекторе указан класс, то это специфичность 10. Если в селекторе указан тэг, то это специфичность 1. И они все суммируются.

1.1.3. Методология БЭМ

БЭМ — это методология, как называть стили и как в целом верстать. Он описывает компонентный подход к веб-разработке. В его основе лежит принцип разделения интерфейса на независимые блоки.
БЭМ расшифровывается как Блок, Элемент, Модель.
Блок в методологии БЭМ — это функционально независимый компонент страницы, который может быть повторно использован. Блоком будет являться кнопка у вас на сайте, потому что эта кнопка может быть использована повсеместно на других страницах, и глупо привязывать ее к текущей. Название блока характеризует смысл (что это? — меню или кнопка), а не состояние (например, как она выглядит). В CSS по БЭМ не рекомендуется использовать селекторы по тэгам или по id, а рекомендуется использовать селекторы по классам.
Элемент в методологии БЭМ — это составляющая часть блока, которая не может быть использована в отрыве от него. Например, у вас внутри кнопки есть некая иконка. Эту иконку, скорее всего, в отрыве от этого блока нельзя будет использовать, поэтому это будет называться элементом. Название элемента характеризует его смысл (пункт меню), а не состояние (например, нажатый пункт меню или красный пункт меню). Структура полного имени элемента соответствует схеме: имя блока, которым является этот элемент, два андерскора, имя элемента. В имени блока не допускается нижнее подчеркивание, и в имени элемента тоже не допускается нижнее подчеркивание. Разделять слова стоит с помощью черточек.
Модификатор в методологии БЭМ — это сущность, которая определяет внешний вид, состояние или поведение блока. Например, название модификатора характеризует внешний вид (например, size_s, size_m — его размер) или его состояние (например, disabled, visited и так далее). Называется имя блока или имя элемента, один андерскор, модификатор.

Пример формы, сверстанной по БЭМ-у. Форма у нас будет являться блоком. И этот блок будет называться login-form. Дальше идет первый input. Это будет являться элементом этой формы, и стиль у него будет login-form__ username-input. Дальше у нас идет также элемент buttons, то бишь класс у него будет login-form__buttons. Внутри него также будут идти элементы. Например, элемент button login-form__submit. И в конце у нас будет еще один button, но можно увидеть, что у него есть класс login-form__ reset_disabled. Это будет означать, что мы применили модификатор к этой кнопке, и эта кнопка сейчас в данный момент неактивна.

```
<form class="login-form">
    <input type="text"
           class="text-input login-form__username-input"/>
    <div class="login-form__buttons">
        <button class="login-form__submit button">
            <span class="button__capture button__capture_red">CLICK ME</span>
        </button>
        <button class="login-form__reset login-form__reset_disabled button">
            RESET</button>
    </div>
</form>
```

Рис. 2

Создадим структуру проекта для шаблонов. Нам необходимо добавить папку templates в корень проекта и папку templates во внутрь нашего приложения. В корневой папке templates будут хра-

ниться шаблоны, которые относятся ко всему проекту, базовые шаблоны. А внутри приложений будут находиться шаблоны, которые относятся именно к этому приложению. Таким образом мы будем разделять проект на логические части. Для начала добавим папку templates и создадим в ней base.html.

Теперь нужно сказать Django о том, что необходимо искать в корневой папке templates шаблоны. Для этого нам необходимо пойти в cousera_blog, найти файл с settings.py и найти там параметр templates. Мы тут видим, что указан параметр APP_DIRS. Также нам нужно указать, чтобы она искал в корневой папке templates. Для этого нам нужно указать путь в DIRS, для этого мы напишем os.path.join(BASE_DIR, 'templates'). os.path.join соединяет путь, который находится в переменной BASE_DIR, 'templates', то есть корень нашего проекта плюс templates.

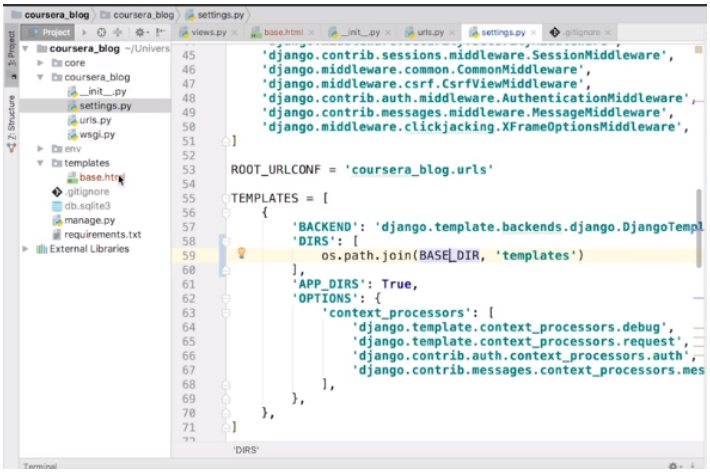

Рис. 3

Теперь давайте отредактируем base.html. Сделаем block title, block styles и block content. Зачем это нужно? Все наследники будут наследоваться от base.html. Нам необходимо будет только переопределить эти блоки.

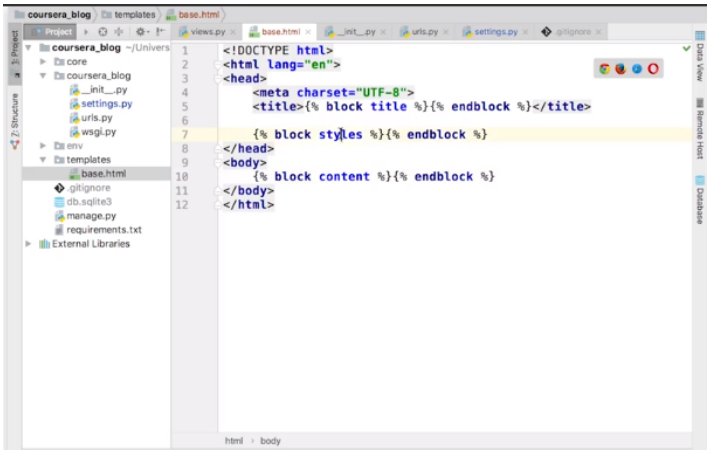

Рис. 4

Теперь давайте создадим шаблон внутри папки core. Это сделано для того, чтобы в дальнейшем, когда мы в коде указывали этот шаблон, нам было проще понять, к какому приложению он относится, и создадим в нем файл index.html.
Теперь мы просто берем и наследуемся с помощью тега extends от нашего базового шаблона, base.html и переопределяем только те блоки. Теперь нам нужно пойти в наш view и заменить наш шорткат рендер. У нас появилась страница контент.

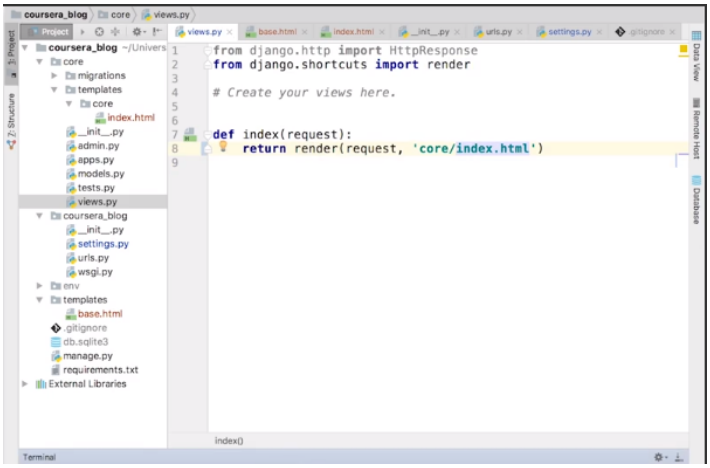

Рис. 5

Теперь давайте добавим стили. Так же как с шаблонами, необходимо добавить папку static, в ней будет храниться вся статика нашего проекта. В ней добавляем папку core, также для того, чтобы понимать какому приложению относится статика, добавляем в ней еще одну папку css с типом статики и в ней наконец добавим наш файл со стилями index.css и напишем там наш первый стиль. То есть этот селектор означает, что ко всем тегам див применить цвет красный. И давайте этот файл со стилями подключим. Открываем наш шаблон index.html и теперь нам необходимо переопределить блок со стилями. Путь нельзя так просто указать, его необходимо указать с помощью тега static и написать тут путь к нашей статике.

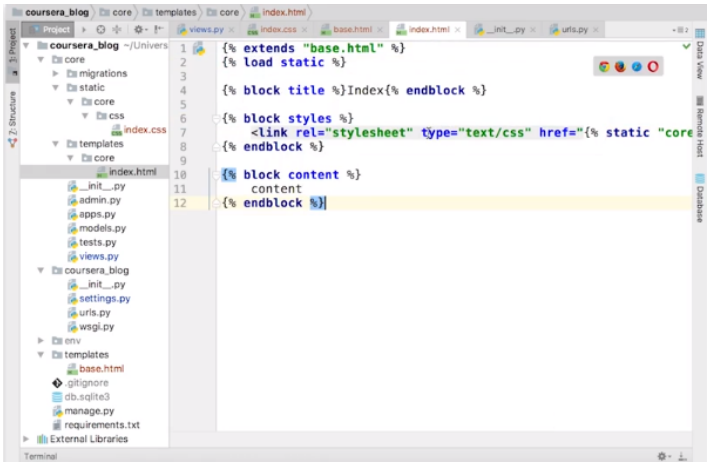

Рис. 6

Static необходимо подключить.
Если открыть панель разработчика, то мы увидим два запроса: запрос на индекс страницу, на html этой страницы и запрос на файл статики index.css. Если мы откроем и увидим, то путь будет /static/core/css/index.css. Это тег он берёт и подставляет префикс, по которому будут отдаваться статика нашим серверам. Если увидеть, то в логе запросов у нас будет два запроса, это запрос на страницу index.html и /static/core/css/index.css. Когда дев серверы получают запрос со статикой, он знает, что это запрос по которому необходимо дать статический файл.
Теперь перейдем к рассмотрению стилей. Начнем с свойств текста. Придем в наш файл с стилями index.css, уберем наш предыдущий селектор и там пишем селектор для класса test и добавим этот класс к нашему контенту.

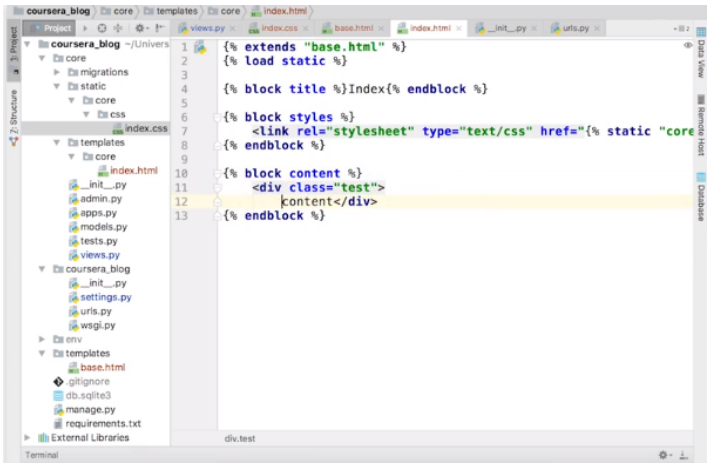

Рис. 7

Первое css-свойство, которое мы рассмотрим, это тип шрифта font-family. Для того, чтобы указать курсив, необходимо указать font-style: italic, и мы увидим, что шрифт станет курсивом. Для того, чтобы указать, насколько жирный шрифт, необходимо указать свойство font-weight. Для того, чтобы изменить размер шрифта, необходимо указать font-size и указать размер шрифта в пикселях.

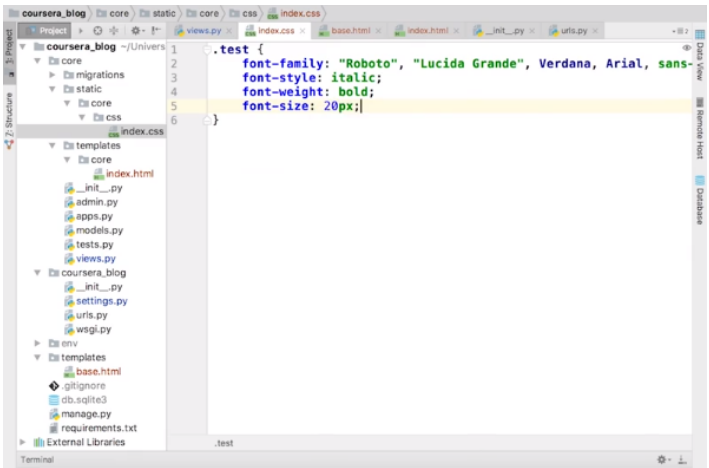

Рис. 8

Теперь рассмотрим простейшие свойства цвета. Свойство color указывает цвет внутри блока, а background-color – это цвет фона.
Теперь рассмотрим css свойства отступов. Ширина блока вообще создается из его содержимого, его паддингов, то есть отступов, грубо говоря, вовнутрь, его границы и его маржинов, отступов снаружи. Давайте это посмотрим. Для этого создадим второй блок - test2, применим к нему эти свойства. Напишем селектор, напишем, что у него высота 100 пикселей, паддинги у него 10 пикселей, граница, указывается с помощью свойства border, 5 пикселей, и отступы снаружи - марджины пусть будут тоже 10 пикселей.

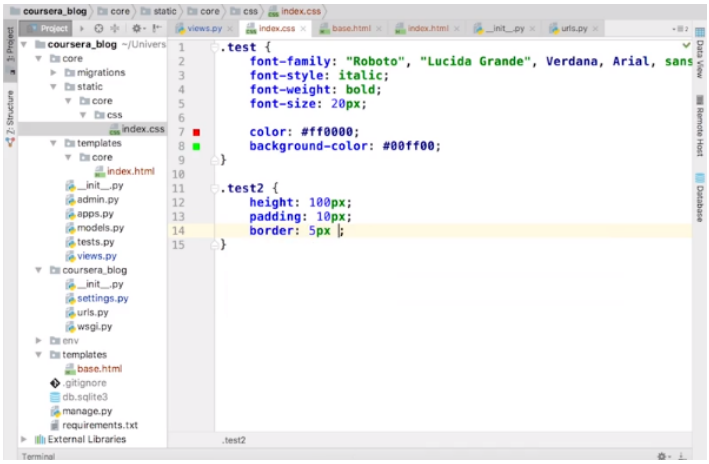

Рис. 9

Отступы снаружи, margins, занимают 10 пикселей, наша граница занимает у нас 5 пикселей, отступы вовнутрь занимают 10 пикселей и наша ширина, это весь наш экран, и высота ее 100 пикселей, которые мы указали. Т.е. ширина и высота нашего элемента складывается не только из его содержимого, но еще и из отступов, ширины границ и отступа снаружи.
Давайте рассмотрим режим отображения элементов. Перейдем в наши стили. Начнем с свойства и его значения display: none, это режим отображения такой, что элемент не видим и не занимает место. При display: block он себя будет вести так, как блочный элемент. Если сделать display: inline, то он будет вести себя как строчный элемент. Мы видим, что свойства высоты перестало работать и ширину, которое он занимает, ровно то, что занимает его отступы, граница и контент. Display: inline-block – это блочный элемент, который не разрывает строку, то есть, если мы напишем два блока подряд с режимом отображения на inline-block, то они будут располагаться на одной и той же странице, но они будут вести себя так же, как и блочный элемент. Т.е. они отображаются на строке.

1.2. Применение Twitter BootStrap

Bootstrap — это библиотека для верстки, то есть это набор некоторых готовых компонент с определенным стилем, что упрощает верстку сайта в стиле bootstrap. Давайте его подключим. Для этого откроем страницу документации официальной страницы bootstrap, нажмем download,

выберем Compiled CSS and JS и нажмем download. Скачается архив. Разархивируем этот архив и положим его в папку bootstrap вовнутрь нашего проекта в Core Static. Мы положили наш bootstrap в папку Static приложение Core. Также для Bootstrap необходим JQuery. Тоже его скачаем. Пойдем на официальную страницу JQuery, нажмем download, выберем Compressed Production JQuery 3.2.1. Скачанный файлик JQuery мы положим в папку JQuery вовнутрь нашего приложения. Теперь давайте подключим Bootstrap. Для этого мы пишем тег link, rel = stylesheet, type = text /css. И ссылку также сформируем с помощью тега static и укажем путь к css bootstrap. css bootstrap.min.css. И подключим static. И теперь необходимо подключить js, также создадим для него block scripts. И подключим наши скрипты. Они необходимы для работы bootstrap. Подключаем сначала JQuery и после этого подключаем bootstrap. Тег block scripts создан с такой же целью, как и stylesheet, чтобы при необходимости его можно было переопределить.

Теперь давайте рассмотрим основные составляющие bootstrap. Основное — это его сетка. Сначала необходимо указать контейнер. В bootstrap есть два типа контейнеров: это контейнер и контейнер-fluid.

Контейнер-тест, открываем и увидим, что test у нас не в самом левом краю, а где-то с некоторым отступом. Этот контейнер, он не занимает всю ширину страницы, а некоторую ее часть. И этот контейнер расположен посередине. Контейнер-fluid, он занимает всю ширину страницы.

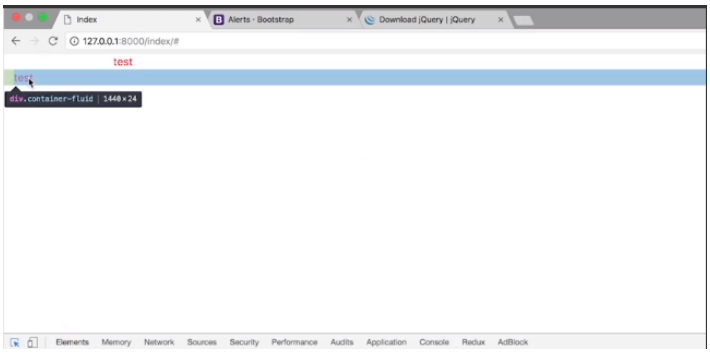

Рис. 10

Для того чтобы использовать сетку bootstrap, необходимо внутри контейнера указать блок row. Это будет строкой. И указать столбцы — col.

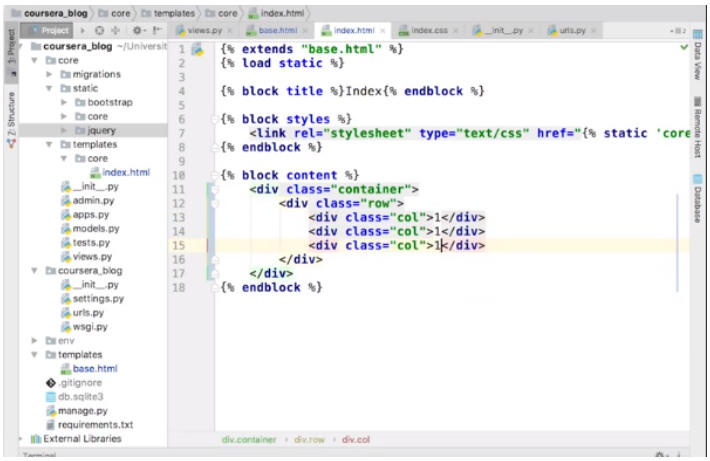

Рис. 11

В итоге внутри нашего контейнера расположены три столбца, 1, 1, 1, которые имеют одинаковый размер. Что же делать, если нам необходимо иметь разную ширину этих столбцов? Bootstrap все свое пространство делит на 12 частей. И можно указать модификатор с циферкой от 1 до 12, и сумма всех столбцов должна равняться 12.
Теперь давайте рассмотрим компоненты. Учиться мы будем на примере блога. Удалим наш контейнер, и давайте начинать верстать сайт. У всех сайтов есть шапка. Давайте пойдем в документацию bootstrap, найдем в компонентах bootstrap Navbar. Найдем подходящий нам, скопируем и посмотрим, что же получилось. Копируем Nav и вставляем в base html, потому что Navbar находится на всех страницах, и стоит это сделать именно в базовом шаблоне, чтобы каждый раз этот Nav не повторять. Давайте откроем нашу страницу и посмотрим, что же получилось.

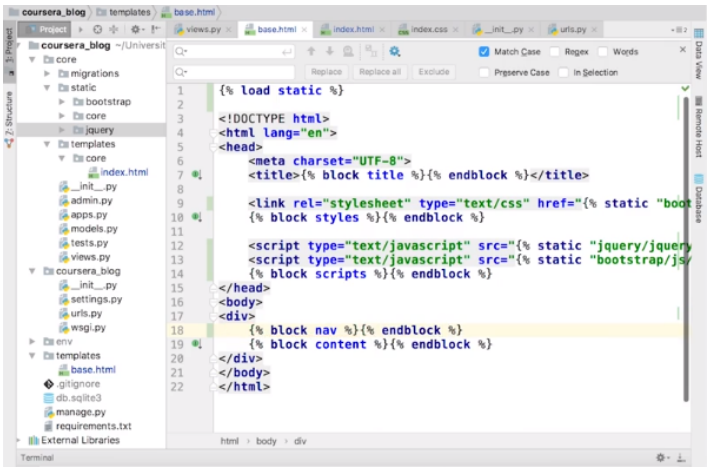

Рис. 12

Теперь давайте сверстаем список топиков. Так как список топиков — это достаточно распространенный элемент у нас, то мы его вынесем в отдельный компонент. Для этого создадим шаблон, topic list, и напишем blog div topic-list, и внутри него будут наши топики. Верстаем мы по БЭМу. Кнопку мы верстать не сами будем, мы пойдем в готовые компоненты нашего bootstrap, найдем

компонент Buttons и выберем кнопку, которая нам больше всего нравится. Посмотрим, что же получилось.

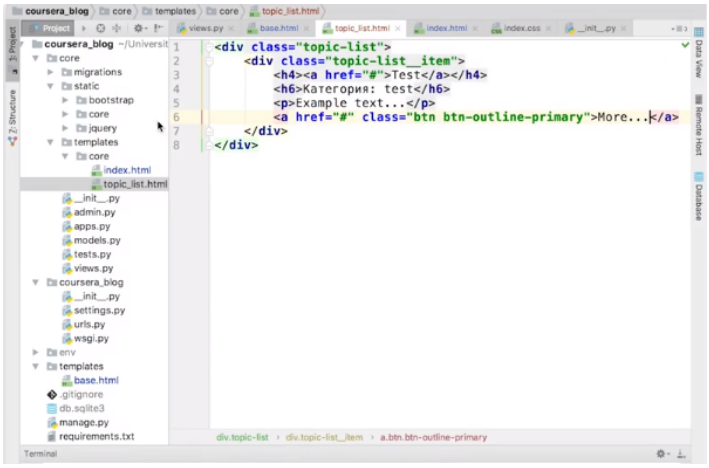

Рис. 13

Давайте еще добавим топиков. Добавили и увидели, что не хватает отступов от нашей шапки и не хватает отступов между нашими топиками. Давайте это исправим. Добавим стиль navigation, в котором укажем отступ снизу в 20 пикселей, и добавим этот стиль в блок navigation. Смотрим, что добавился отступ от Navbar.

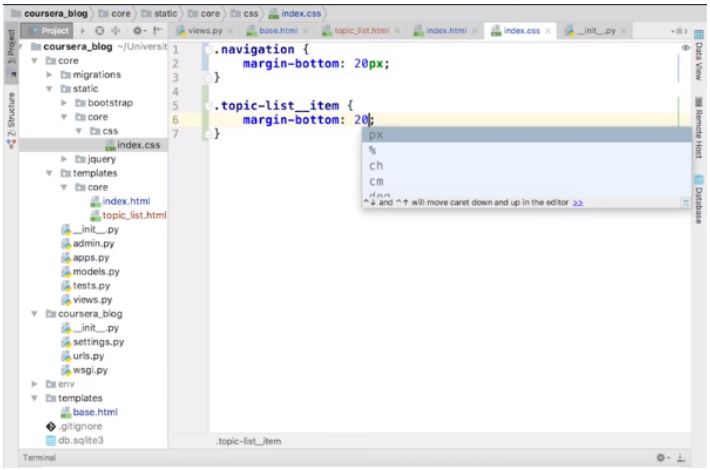

Рис. 14

Теперь давайте применим знание bootstrap сетки и добавим справа категории. Как мы знаем, для того чтобы использовать сетку, необходимо внутри контейнера сделать блок с классом row и в него добавить class col. Пусть наши топики будут занимать восемь ячеек, а категории четыре. И добавим внутри него наш topic list. Так как категории также могут использоваться повсеместно по проекту, то также его вынесем в отдельный компонент. Создадим шаблон category и сверстаем наши категории. Пусть у нас будет две категории — test1 и test2. Давайте добавим

некий background у категорий, чтобы они выделялись. Добавим отступы вовнутрь padding в 10 пикселей и посмотрим, что же получилось.

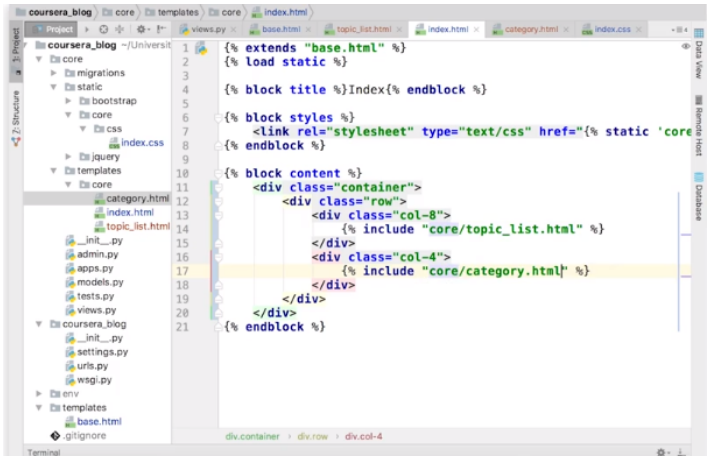

Рис. 15

Теперь давайте сверстаем детали топика. Мы туда будем проваливаться по кнопке More или по Test и будем попадать на детали топика. Для этого открываем views, добавляем еще один view, topic details. И создадим внутри core шаблон topic details. Укажем, что нужно рендерить именно этот шаблон.

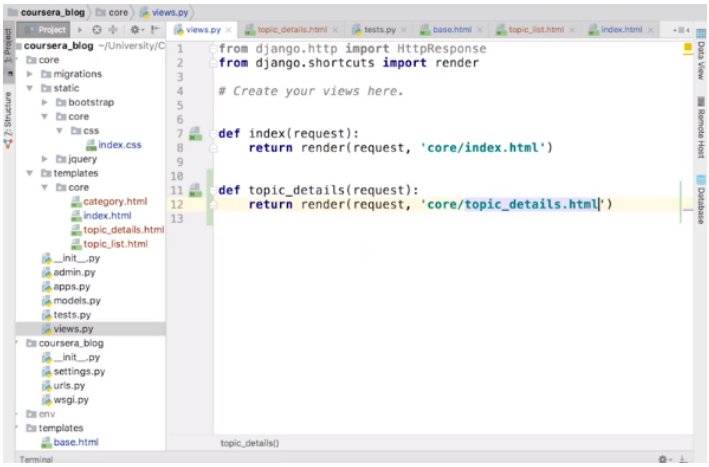

Рис. 16

Теперь в наших URL укажем этот view. Добавляем топик, view, и так как это детали топика, то его нужно открывать по некоторому идентификатору. Давайте добавим в URL наш идентификатор.

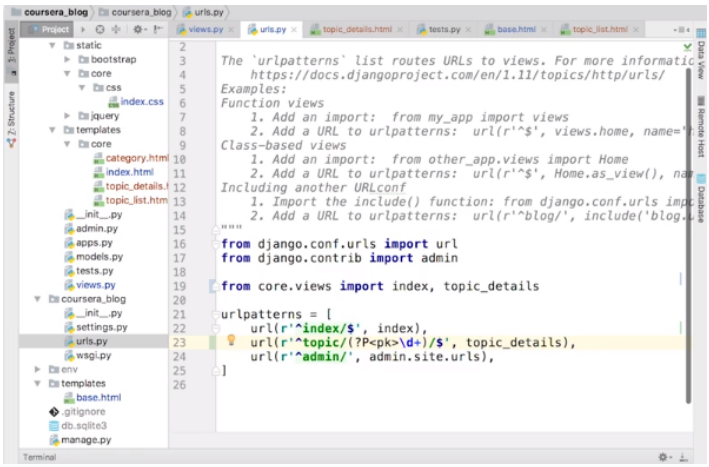

Рис. 17

И этот параметр pk будет передан в view. Для этого нужно указать параметр pk тут.

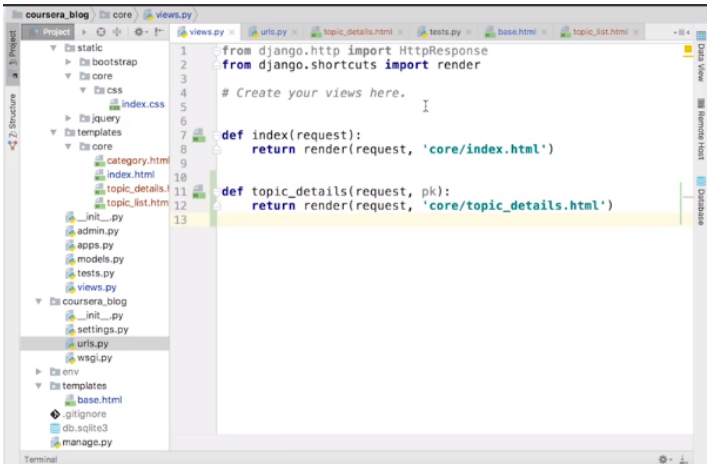

Рис. 18

Давайте напишем что-нибудь в div. Также вспомним, что у URL есть алиасы. Назовем этот URL алиасом topic details. Теперь откроем наш topic-list и постараемся сформировать URL к нашему топику. Указываем тут его alias topic_details и указываем его айдишник. Повторим это для топиков ниже. Таким образом мы этот URL не захардкодили в шаблоне. Мы использовали его alias. В случае, если нам необходимо будет просто поменять название, то мы меняем его в урле, нигде по коду нам это больше делать не нужно. Просто поменяется URL.

2. Django + BootStrap

2.1. Оживляем наш блог

Давайте динамически будем выводить топики из базы.
Создадим базу. Подключаемся к mysql с пользователем root и создаем базу под именем coursera_blog.
Теперь давайте создадим пользователя. Пользователя coursera_blog с паролем даже coursera_blog.
И дадим ему права на базу данных coursera_blog.
Теперь нам необходимо установить коннектор к mysql из Python. Для этого мы перейдем в файл requirements и добавим туда строчку mysql Client, версия 1.3.12. И после этого зайдем в консоль, напишем pip install –r requirements.
Теперь, для того чтобы подключиться из Python к базе данных, необходимо открыть coursera_blog settings.py и отредактировать database.
ENGINE меняем на django.db.backends.mysql. NAME меняем на coursera_blog. USER меняем тоже на coursera_blog. PASSWORD меняем на coursera_blog. HOST: 127.0.0.1, PORT: 3306 дефолтный для mysql. И OPTIONS.
Насколько мы помним, OPTIONS — это переменная, которая зависит от базы данных, и нам необходимо указать, с помощью какой кодировки мы будем общаться с базой. Теперь, для того чтобы убедиться, что все хорошо, необходимо накатить миграции с помощью команды python manage.py migrate. И теперь давайте подключимся к mysql с пользователем coursera_blog и паролем coursera_blog.
Посмотрим, что база у нас есть, с помощью команды show databases видим, что coursera_blog есть, напишем, что мы ее хотим использовать, и посмотрим, какие же таблицы у нас создались. Мы видим, что создались дефолтные джанговские таблицы из auth и из django. Мы видим, что первое — это название приложения, а второе — это название таблицы.
Теперь давайте зайдем в core, models.py и создадим наши модели. Будет модель Category, поле title, которое будет являться CharField с максимальным размером 255, и таблица Topic. Также с полем title тоже является CharField с максимальным размером 255, body, которое будет являться TextField, и ManyToMany — связь к категории. И укажем related_name=topics.
Создадим миграцию и ее применим. Выйдем из mysql. Для того чтобы создать миграцию, напишем python manage.py makemigrations core, посмотрим, что у нас создалось две модели, и зайдем вовнутрь migrations. Создалась миграция, нужно создать модель с названием Category, с полем id, оно автоматически добавляется, и title. Применим эту миграцию с помощью также команды python manage.py migrate. Миграция применилась, и давайте теперь зайдем также в mysql, используем базу coursera_blog и посмотрим, какие таблицы у нас сейчас создались.
Мы видим, что создались три таблицы: core_category, core_topic и core_topic_categories. core_topic_cate будет являться связкой ManyToMany между категориями и топиками. Теперь давайте откроем shell и заполним наши топики. Для этого напишем python manage.py shell. Откроется консоль, импортируем наши модели core.models, import Topic, Category, и для начала создадим два топика: topic1=Topic. objects.createtitle=’First Topic’, и body= ’body’. И topic2= Topic.objects.create title =’Second Topic’, body=’body2’.
Давайте создадим три категории. Category.objects.create (title= ’Category1’). И давайте добавим Topic1 в три категории, а Topic2 в две категории. Для этого напишем topic1.categories.add.(category, category2, category3). И topic2 добавим в категорию два и три.
Теперь давайте приступим к написанию наших обработчиков. Выйдем из консоли и напишем наш индекс-обработчик. Собственно, давайте напишем запрос с помощью нашей ORM. Импор-

тируем модель topic и категорию заодно. И применим аннотацию, посчитаем, сколько в каждом топике категорий. Собственно, пишем count и считаем, сколько категорий относится к каждому топику. Также получим все категории, чтобы их вывести справа. Category.objects.all.
Мы сделали два запроса, теперь их необходимо передать в наш контекст. Собственно, topics и categories. Теперь откроем наш шаблон. Core templates, topic_list. Удалим наш статический контент. Напишем цикл for topic in topics. У нас в цикле будет выводиться наш topic_list item. Почему topics? Потому что мы в контексте его так назвали. Теперь давайте выведем его title.
Давайте выведем его категории. Это также делается с помощью цикла. For category in topic.categories.all. И закроем цикл. Внутри цикла выведем category title. Category.title, поставим после него запятую.
Мы посчитали количество категорий в каждом топике, давайте выведем после слова "категория" количество. Насколько мы помним, имя формируется — это то, что мы указали то, что мы будем группировать и его агрегаты. То есть categories__count. Потом выведем в body. Topic.body. Заменим на topic.pk.
Теперь перейдем в наш шаблон категорий. Удалим статический контент и также напишем цикл, for category. Вместо test1 выведем категорию title. Теперь давайте откроем view, topic details и сделаем переход на topic тоже динамический. Открываем views. Во view topic details получаем топик с помощью Topic.objects.get и передаем pk=pk. Перехватываем исключения. И в случае, если он не существует, кидаем исключение. Передаем в контекст топик. Теперь открывает шаблон topic_details, там такой же контент, как и был в topic_list. Но немного с нюансами. Ссылку мы оставляем, topic.title категорию посчитаем по-другому, потому что у нас уже не список, а просто один элемент, и мы просто можем прямо в шаблоне написать топик categories.count. Это мы оставляем topic.body, ссылку мы удаляем. В итоге удаляем все лишнее, смотрим, что получилось. Открываем, видим, что мы получили наш топик по id2. Если тут написать id3, то если мы перейдем по топику first.topic, мы увидим firs topic, если по second.topic, то мы увидим second topic.
Теперь давайте добавим фильтры. Во-первых, сначала добавим alias'ы для индекса. И зайдем в base.html. Поправим ссылку в шапке на url index, news.
Теперь давайте оживим нашу форму поиска. Для этого в form мы передадим action url.index. В input передадим параметр name Q. Теперь откроем нашу view, в индекс добавим фильтров. Пока мы не обратимся к данным, он не выполнится. Поэтому мы можем сделать так: if q = request.GET.get (Q) if q is not NONE, topics = topics.filter. title icontains q. Насколько мы помним, icontains означает, что в title содержался q. Таким образом мы не будем делать два запроса, а он исполнится только тогда, когда мы обратимся к данным. Таким образом мы можем конструировать наш запрос.
Открываем, заходим, пишем, например, first. Мы написали first и у нас остался только первый топик. Если напишем second, останется только second. Фильтр по текстовому запросу работает.
Теперь давайте добавим фильтр по категории. Для этого нам необходимо открыть categories, и сформировать ссылку url.index, и добавить get parameter category равняется category.pk Открываем теперь нашу вьюху, добавляем еще одно условие. Category requests.GET.get category. If category is not None, topics.filter, и должна быть связка с categories с таким pk.