# **Модели: Базовые инструменты**

## **Параметры, поддерживаемые полями всех типов**

1. Verbose\_name = человеческое название поля
2. Help\_text – дополнительный поясняющий текст, выводимый на экран
3. Default – значение по умолчанию
4. Unique – если True, то в текущее поле может быть занесено только уникальное в пределах таблицы значение
5. Umiquer\_for\_date –
6. Null – если True, то поле в таблице базы данных может хранить значение null
7. Blank – если True, то Django позволит занести в поле пустое значение, тем самым сделав поле необязательным к заполнению
8. Db\_index – если True, то по текущему полю в таблице будет создан индекс
9. Primary\_key – если True, то текущее поле станет ключевым

## **Классы полей моделей**

1. CharField – строковое поле, хранеящее строку ограниченной длины
2. TextField – текстовове поле, хранящее строку неограниченной длины
3. EmailField – адрес электронной почты в строковом виде
4. SlugField – слаг, т.е. строка, однозначно идентифицирующая запись и включаемая в состав интернет адреса (max\_length,)
5. BooleanField – логическое поле
6. IntegerField – знаковое целочисленное поле обычной длины (32 - разрядное)
7. SmallIntegerField – 16-разрядное число
8. BigIntegerFiled – 64- разрядное число
9. PositiveIntegerFIled – безнаковое целочисленное поле половинной длины (16-разрядов)

10)FloatField – вещественное число

11) DateField – значение даты в виде объекта типа date из модуля datetime

Поля:

Auto\_now – елси True, то при каждом сохранении записи в поле будет заноситься текущее значение даты

Auto\_now\_add – то же самое, что и auto\_now, но текущая дата заносится в поле только при создании записи и при последующих сохранениях не изменяется

## **Создание полей со списком**

Перечень значения длдя выбора заносится в парметр choices конструктора поля и может быть задан в виде:

1. Последовательности – списка или кортежа, каждый элемент которого представляет отдельное значение и записывается в виде последовательности из двух элементов (значения, по которому мы можем обращаться, и значение выводящееся пользователю)

Пример

Class Bb(models.Model):

KINDS = (

(‘b’, ‘Куплю’)

)

Kind = models.CharField(max\_length=1, choices = KINDS, default=’s’)

Значения могут объединяться в группы

KINDS = (

(‘Купля-Продажа’, (

(‘b’, ‘Куплю’),

(‘s’, ‘Продам’)

))

)

**Создание связей между моделями**

**Связь один со многими**

Для создания связи такого типа, во вторичной модели следует объявлить поле типа ForeignKey

ForeignKey(связываемая первичная модель, on\_delete-поведение при удалении записи, остальные параметры)

On\_delete может содержать

1. CASCADE – удаляет все связанные записи вторичной модели
2. PROTECT -предотвращает удаление записи первичной модели
3. SET\_NULL - заносит в поле внешнего включа всех связанных записей вторичной модели значение null
4. SET\_DEFAULT – заносит в поле внешнего ключа всех связанных записей вторичной модели заданное для него значение по умолчанию

Класс foreignKey поддерживает следующие дополнительные необязательные параметры

1. Limit\_choices\_to – позволяет вывести в раскрывающемся списке записей первичной модели, отображаемом в веб-форме, только записи, удовлетворяющие заданным критериям фильтрации

Limit\_choices\_to = {‘show’: True}

1. Related\_name – имя атрибута записи первичной модели, предназначенного для достпупа к связанным записям вторичной модели, в виде строки

First\_rubric = Rubric.objects.first()

Получаем доступ к связанным объявлениям через атрибут entries, указанный в параметре related\_name

Bbs = first\_rubric.entries.all()

## **Связь один с одним**

Такая связь создается в классе вторичной модели объявлением поля типа OneToOneField.

From Django.contrib.auth.models import User

class AdvUser(models.Model):

is\_activated = models.BooleanField(deffaykt=True)

user = models.OneToOneField(User, on\_delete=models.CASCADE)

## **Связь многие со многими**

Для создания такой связи нужно объявитьв одной из моделей поле внешнего ключа ManyToManyField

Class Spare(models.Model):

Name= models.CharFIeld(max\_length=30)

Class Machine(models.Model):

Name = models.CharField(max\_length=30)

Spares = models.ManyToManyField(Spare)

## **Параметры самой модели**

Параметры самой модели описываются различными атрибутами класса Meta, вложенного в класс модели и не являющегося производным ни от какого класса.

1. Verbose\_name – название сущности, зранящейся в модели, которое будет выводится на экран
2. Verbose\_name\_plural – название набора сущностей
3. Ordering – параметры сортировки записей модели по умолчанию

Ordering = (‘-published’, ‘title’ )

1. Get\_latest\_by – имя поля типа DateField которое будет взято в расчет при получении наиболее поздней или наиболее ранней записи с помощью метода latest

Get\_latest\_by = ‘published’

## **Интернет-адрес модели и его формирование**

Императивный способ заключается в непосредственном переопределении метода get\_absolute\_url(self)

def get\_absolute\_url(self):

return ‘bboard/%s/’ % self.pk

## **Методы модели**

1. \_\_str\_\_(self) – возвращает строкове представление записи модели.
2. Save(self, \*args, \*\*kwargs) – сохраняет запись. При определении этого метода обязательно следует вставить в нужное место кода вызова метода, унаследованного от базового класса

Пример

def save(self, \*args, \*\*kwargs):

super().save(\*args, \*\*kwargs)

В зависимости от выполнения ил иневыполнения какого-то условия, можно отменить сохранение записи, для чего достаточно просто не вызывать унаследованный метод save

def save(self, \*args, \*\*kwargs):

if self.is\_model\_correct():

super().save(\*args, \*\*kwargs)

1. delete(self, \*args, \*\*kwargs) – удаляет запись. Этот метод также преопределяется для добавления какой-либо логики, кокторая должна выполняться перед удаление или после него

def delete(self, \*args, \*\*kwargs):

super().delete(\*args, \*\*kwargs)

## **Валидация модели. Валидаторы**

Валидацией называется проверка на корректность данных, занесенных в поля модели. Валидацию можно реализовать непосредственно в модели или же в форме, которая используется для занесения в нее данных

## **Стандартные валидаторы Django**

1. MinLengthValidator – проверяет не меньше ли длина заносимой строки, чем минимум, заданный в первом параметре

Пример

From Django.core import validators

def get\_min\_length():

return min\_length

Title = models.CharField(max\_length=50, verbose\_name=’Товар’

Validators=[validators.MinLengthValidator(get\_min\_length)])

1. RegexValidator – проверяет значение на соответсвие заданному регулярному выражению
2. EmailValidator – проверяет на корректность зановимый в поле адрес электронной почты
3. MaxValueValidator – проверяет, не превышает ли заносимое число заданный в первом параметре максимум

## **Вывод собственных сообщений об ошибках**

Собственные сообщения об ошибках указываются в параметре error\_messages конструктора клааса поля

Title = models.CharField(max\_length=50,

Validators = [validators.RegexValidator()], error\_messages={“invalid”: ‘Неправильное названеие товара’})

## **Написание своих валидаторов**

Если нужный валидатор отсутсвеуте в стандартном наборе, мы можем написать его самостоятельно

From Django.core.exceptions import ValidationError

def validate\_even(val):

if val % 2 != 0:

raise ValidationError(f“Число {val} нечетное”)

price = models.FloatField(validators=[validate\_even])

## **Валидация модели**

Может возникнуть необходимость проверить на корректность не значение одного поля, а всю модель. Для этого достаточно переопределить в классе модели метод clean(self)

Пример

Class Bb(models.Model):

…

Def clean(self):

Errors = {}

If not self.content:

Errors[‘content’] = ValidationError(‘Укажите описание продаваемого товара)

If self.price and self.price < 0:

Errors[‘price’] = ValidationError(‘Укажите неотрицательнео значение цены’)

If errors:

Raise ValidationError(errors)

## **Фильтрация записей**

1. Filter – отбирает из текущего набора только записи, удовлетворяющие заданным условиям филтриации

#Отбираем только объявления с ценой не менее 10000руб

For b in Bb.objecs.filter(price\_\_gte=10000)

#с ценой менее 100000

For b in Bb.ojects.filter(price\_\_lt=10000)

1. Exclude – тоже самое, что filter, но наоборот, отбирает записи , не удовелтворяющие заданным условиям фильтарции

For b in Bb.objcets.exlude(price\_\_gte=10000)

Lt – значение меньше заданного

Lte- значение меньше или равно

Gt – значнеие больше заданного

Gte – значение больше ил равно

Стр(144)

## **Сравнение со значениями других полей**

Для этого предназначен класс с именем F, обхявленный в модуле Django.db.models

From Django.db.moedls import F

F = F(‘title’)

For b in Bb.objects.filter(content\_\_icontains=f)

Icontains – Заданное значение должно присутсвовать в значении, зранящемся в поле. Регистр символов не учитывается

## **Сложные условия фильтрации**

Класс Q из модуля Django.db.models позволяет создавать более сложные условия фильтрации.

From Django.db.models import Q

q = Q(rubric\_\_name=’Недвижимость’) | Q(rubric\_\_name=’Бытовая техника’)

for b in Bb.objects.filter(q)

## **Выборка уникальных записей**

Distinct – выводит только уникальные записи

## **Сортировка записей**

Order\_by(поля)

# **Контроллеры-функции**

## **Формирование ответа на основе шаблона**

Для загрузки нужного шаблона Django предоставляет две фнукции, объявленные в модуле Django.template.loader

1. Get\_template(путь к шаблону) – загружает шаблон, расположенный по указанному пути, и возвращает представляющий его экземпляр класса tenplate из модуля Django.template
2. Select\_template(последовательность путей шаблонов) – перебирает указанную последовательность пуетй шаблонов, пытается загрузить шаблон, расположенный по очередному пути, и возвращает первый шаблон, который удалосьб загрузить в виде экземпляра класса Template

Для получения обычной веб-страницы нужно выполнить рендеринг шаблона, вызвва один из следующих метоов

1. Render – выполняет рендеринг текущего шаблона на основе заданного контектса
2. Render\_to\_string(путь к шаблону, context=, request=)- фукнция из модуля Django.template.loader . Загружает шаблон с указанным путем и выполняет его рендерин с применением заданных контекста шаблона и запроса. Возвращает строку с html-кодом сформированнной стрнациы

## **Перенаправление**

Временное перенаправление – перенаправление пользователя на страницу со всеми обхявлениями, после добавления нового объявления. Временное перенаправление выполняется с помощь функции из Django.http

HttpResponseRedirect(reverse(“bboard:index”))

## **Обратное разрешение интернет-адресов**

Механизм обратного разрешения формирует интернет-адреса на основе объялвенных в списках именованных марщрутов

Для формирования адресов в коде контроллеров применяется функция reverse() или reverse\_lazy() из модуля Django.urls

Reverse(“bboard:index”)

Reverse(“bboard:by\_rubric”, args(current\_rubirc.pk))

## **Выдача сообщений об ошибках и обработка особых ситуация**

Для выдачи сообщейни об ошибках и уведомления клиентов об особых ситуациях Django предоставляет ряд классов. Все они являются производными от класса HttpResponse и объявлены в модуле Django.http

1. HttpResponseNotFound – запрашиваемас страница не существует(код 404)

Def detail(request, bb\_id):

Try:

Bb = Bb.objects.get(pk=bb\_id)

Except Bb.DoesNotExist:

Return HttpResponseNotFound(такое объявление не существует)

Return HttpResponse(,,,)

1. HttpResponseBadRequest – клиентский запрос неккоректно сформирован (код 400)
2. HttpResponseForbidden – доступ к запрошенной странцие запрешен (код статуса 403)

## **Отправка файлов**

Для отправки клиентам файлов применяется класс FileResopsne

From Django.http import FileResponse

Filename = r’c:/images/image.png’

Return FileResponse(open(filename), ‘rb’)

## **Отправка данных в формате JSON**

Для отправки данных в формате JSON применяется класс JsonResponse

Date = {‘title’: ‘Мотоцикл’}

Return JsonResponse(data)

## **Сокращения Django**

Сокращения – это функция, выполняющая сразу несколько действий.

1. render() – выполняет рендеринг шаблона и отпраку получившейся в результате страницы клиенту.
2. redirect() – выполняет перенаправление по заданной цели

return redirect(“bboard:by\_rubric”, rubric\_id=bbf.cleaned\_data[‘rubric’].pk)

1. get\_object\_or\_404() – ищет запись согласно заданным условиям поиска и возвращает ее в качестве результата. Если запись найти не удается возбуждает исключение Http404

bb = get\_object\_of\_404(Bb, pk=bb\_id)

1. get\_list\_or\_404() – применяет к записям заданные условия фильтрации и возвращает в качетсве результата полученный набор записей. Елси ни одной записи , удовлетворяющей условиям, не существует, то возбуждает исключение Http404.

# **Контроллеры-классы**

## **Базовые контроллеры-классы**

## **Контроллер View**

Данный класс хранит список [get, post, put, pathc, delete, head, options, trace], включающий все методы, поддерживаемые протоколом Http

Класс также содержит четрые метода, которые переоределяются в подклассах

1. setup(self, request, \*args, \*\*kwargs) – выполняется первым и инициализирует объект контроллера

## **Примесь ContextMixin: создание контекста шаблона**

Класс-примесь ContextMixin добавляет контроллеру-классу средства для формирования контекста шаблона

1. extra\_context – атрибут, задающий содержимое контекста шаблона. Его значение должно представлять собой словарь, элементы которого будут добавлены в контекст
2. get\_context\_date(self, \*\*kwargs) – метод, должен содлать и возвращать контекст шаблона. С параметром kwargs передается словарь, элементы которого должны быть добалвены в контекст шаблона

## **Примесь TemplateResponseMixin: рендеринг шаблона**

Данный класс-примесь добавляет наследующиему классу средства для рендеринга шаблона

1. template\_name – путь к шаблону
2. get\_template\_name(self) – метод, должен возвращать список путей к шаблонам
3. render\_to\_response(self, context, \*\*response\_kwargs) – возвращает экземпляр класса, представляющего ответ.

## **Контроллер TemplateView: все вместе**

Контроллер класс TemplateView наследует классы View, ContextMixin и TemplateREsponseMixin

Он автоматически выполняет рендеринг шаблона и отправку ответа при получении запроса по методу GET

Пример:

From Django.views.generic.base import TemplateView

From .models import Bb, Rubric

Class BbByRubricView(TemplateView):

Template\_name = ‘bboard/by\_rubric.html’

Def get\_context\_data(self, \*\*kwargs):

Context = super().get\_context\_data(\*\*kwargs)

Context[‘bbs’] = Bb.objects.filter(rubric=context[‘rubric\_id’])

Return context

## **Классы выводящие одну запись**

## **Примесь SingleObjectMixin: поиск записи**

Контроллеры-классы из модуля Django.views.generic.detial – более высокоуровневые, чем базовые. Они носят название обобщенных, поскольку выполняют типовые задачи и могут быть использованы в различных ситуациях

Класс-примест SingleObjectMixin наследующи от ContextMixin, выполняет сразу три действия

1. извлекает из полученного интернет адрес ключ или слаг записи
2. ищет заптсь в заданной модели по полученному ранее ключу или слагу
3. помещает найденную запист в контекс шабона

Примест поддерживает довольно много атрибутов и методов

1. model – задает модель
2. queryset – указывет либо диспетчер записей, либо набор записей в котором будет выполняться поиск записи
3. get\_queryset(self) – метод, должен возвращать набор записей, в котором будет выполняться поиск записи
4. context\_object\_name – атрибут, задает имя переменной контекста шаблолна, в которой будет сохранена найденная запись
5. get\_context\_object\_name(self, obj) – метод, должен возвращать имя переменной контекста шаблона, в которой будет сохранена найденная запись, в виде строки
6. get\_context\_data(self, \*\*kwargs) – переопределенный метод, создающий и возвращающий контекст шаблона

## **Контроллер DetailView: все вместе**

Контроллер Detail Veiw ищет запись по полученным значениям ключа или слага, заносит ее в атрибут object и выводит на экран страницу с содержимым этой записи

Пример

From Django.views.generic.detial import DetailView

From .models import Bb, Rubric

Class BbDetialView(DetailView):

Model = Bb

Def get\_context\_data(self, \*\*kwargs):

Context = super().get\_context\_data(\*\*kwargs)

Context[‘rubric’] = Rubric.objects.all()

Return context

## **Классы выводящие наборы записей**

Обобщенные классы из модуля Django.veiws.generic.list выводян на экран целый набор записей

## **Примесь MultipleObjectMixin: извлечение набора записей**

Поля:

1. model – модель
2. queryset – атрибут, указывает либо диспетчер записей, либо исходный набор записей, из которого бцдцт извекаться записи
3. ordering – атрибут, задающий параметры сортировки записей. (значение указывается в виде строки с именем поля, либо последовательности строк с именами полей)
4. paginate\_by – атрибут, задающий целочисленное количество записей в одной части пагинатора.
5. Page\_kwarg – атрибут задающий URL или GET параметр, через который будет передаваться номер выводимой части пагинатора.
6. Paginator\_class – атрибут, указывающий класс использеумого пагинатора (по умолчанию Paginator из модуля Django.core.paginator)
7. Context\_object\_name – атрибут, задает имя переменной контекста шаблона, в которой будет сохранен извелечнный набор записей
8. Get\_context\_data

## **Контроллер ListView: все вместе**

Он извлекает из модели набор записей, записывает его в атрибут object\_list и выводит на экран страницу со списком записей

Пример

From Django.views.generic.list import ListView

Form .models import Bb, Rubric

Class BbByRubricView(ListView):

Template\_name = ‘bboard/by\_rubric.html’

Context\_object\_name = ‘bbs’

Def get\_queryset(self):

Return Bb.objects.filter(rubric=self.kwargs[‘rubric\_id’])

Def get\_context\_data(self, \*\*kwargs):

Context = super().get\_context\_data(\*\*kwargs)

Context[‘rubrics’] = Rubric.objects.all()

Return context

# **Классы, работающие с формами**

## **Классы для вывода и валидации форм**

## **Примесь FormMixin: создание формы**

Примесь FormMixin создает форму, проверяет введенные в нее данные, выполняет перенаправление, если данные прошли проверку, или выводит форму повторно

Атрибуты и методы:

1. Form\_class – атрибут, хранит ссылку на класс используемой формы
2. Get\_form\_class – метод, должен возвращать ссылку на класс с используемой формы
3. Initial – атрибут, хранящий словарь с изначальными данными для занесения в только что созданную форму. Ключи элементов этого словаря должны соответсвовать полям формы, а значения элементов зададут значения полей.
4. Success\_url – атрибут, зранит интернет адрес дл яперенаправления, если введенные в форму данныые прошли проверку на корректность
5. Prefix – атрибут, задает строковый префикм для имени формы, который будет присутсвовать в создающем форму HTML коде.Префик стоит задавть только в том случае елс ипланируется поместить несколько однотипных форм в одном теге form.
6. Get\_form(self, form\_class=None) – метод, создающий и возвращающий объект формы
7. Get\_context\_data
8. Form\_valid(self, form) – метод должен выполнять обработку данных, введенных в переданную через параметр form форму, в том случае, если они прошли валидацию

## **Контроллер ProcessFormView: вывод и обработка формы**

Контроллер класс ProcessFormView выводит на экран страницу с формой, принимает введенные данные и проводит их валидацию.

Он переопределяет три метода

1. Get(self, request, \*args, \*\*kwargs) – выводит страницу с формой на экран
2. Post(self, request, \*args, \*\*kwargs) – получает введенные в форму данные и выполняет их валидацию. Если валидация прошла успешно, вызывает метод form\_valid(), в противном случае – метод form\_invalid

## **Контроллер-класс FromView: создание, вывод и обработка формы**

**Пример**

From Django.views.generic.edit import FormView

From Django.urls import reverse

From .models import Bb, Rubric

Class BbAddView(FormView):

Template\_name = ‘main/create.html’

Form\_class = BbForm

Initial = {‘prirce’: 0.0}

Def get\_context\_data(self, \*args, \*\*kwargs):

Context = super().get\_context\_data(\*args, \*\*kwargs)

Context[‘rubrics’] = Rubric.objects.all()

Return context

Def form\_valid(self, form):

Form.save()

Return super().form\_valid(form)

Def get\_form(self, form\_class=None):

Self.object = super().get\_form(form\_class)

Return self.object

Def get\_success\_url(self):

Return reverse(“bboard:by\_rubric”, kwargs={‘rubric\_id’: sself.object.cleaned\_data[‘rubric’].pk})

# **Классы для добавления, правки и удаления записей**

## **Примесь ModelFormMixin: создание формы, связанной с моделью**

Атрибуты и методы:

1. Model – атрибут, задает ссылку на класс модели, на основе которой будет создана форма
2. Fields – атрибут, указывает последовательность имен на основе полей модели, которые должны присутсвовать в форме
3. Get\_form\_class – метод, должен возвращать ссылку на класс используемой формы
4. Success\_url
5. Form\_valid

## **Контроллер CreateView: создание новой записи**

Данный класс выводит форму, проверяет введенные в нее данные и создает на их основе новую запись

## **Контроллер UpdateView: исправление записи**

Данный контроллер ищет запись по полученным из url парметра ключу или слагу, выводит странниуц с формой для ее правки, проверяет исправленны еданные

Поскольку класс UpdateView предварительно выполняет поиск записи, в нем необходимо указать модль, набор записей в атрибуте queryset

From Django.view.generic.edit import UpdateView

From .models import Bb

Class BbEditView(UpdateView):

Model = Bb

Form\_class = BbForm

Success\_url = ‘/’

## **Контроллер DeleteView: удаление записи с подтверждением**

Данный контроллер ищет запись по полученному из URL параметру ключу ил ислагу, выводит страницу с подтверждения, включающую дв себя форму с кнопкой удаления, и удаляет запись

Также в классе доступен атрибут object, в котором хранится удаляемая запись.

From Django.views.generic.edit import DeleteView

Form .models import Bb

Class BbDeteleView(DeleteView):

Model = Bb

Success\_url = ‘/’

Def get\_context\_data(self, \*args, \*\*kwargs):

## **Классы для вывода хронологических списков**

**Стр 217-227**

## **Контроллеры-классы смешанной функциональности**

Большая часть функциональности контроллеров-классов наследуется ими от классов-примесей. Наследуя классы от нужных примесей, можно создавать контроллеры смешанной функциональности.

Так мы можем объявить класс, производный от классов SingleObjectMixin и ListView. В результате получится контроллер, одновременно выводящий сведения о выбранной записи и набор связанных с ней записей.

From Django.views.generic.detail impoer SingleObjectMixin

From Django.views.generic.list import ListDetail

From .models import Bb, Rubric

Class BbByRUbricView(SingleObjectMixin, ListView):

Template\_name = ‘bboard/by\_rubric.html’

Pk\_url\_kwarg = ‘rubric\_id’

Def get(self, request, \*args, \*\*kwargs):

Self.object = self.get\_object(queryset=Rubric.objects.all())

Return super().get(request, \*args, \*\*kwargs)

Def get\_context\_data(self, \*\*kwargs):

Return context

Def get\_queryset(self):

Return self.object.bb\_set.all()

# **Шаблоны и статические файлы:**

## **Базовые инструменты**

## **Теги шаблонизватора**

Для цикла for:

Forloop.counter – номер текущей итерации цикла

{% now формат %} – выовдит текущие дату и время. Оформленное согласно заданному формату.

Filter … endfilter – применяет к содержимому указанные фильтры

{% filter фильтр %}

{% endfilter %}

## **Формы, связанные с моделями**

Форма в терминологии Django – это объект, выводящий на страницу веб форму для занесения данных и проверяющий введенные данные на корректность. Форма определяет набор полей, в которые будут выводиться отдельные значения, типы занносимых в них значений.

Форма, связанная с моделью, отличается от обычной формы тем, что представляет какую-либо запись модели – храннящуюся в базе данных или еще не существующую.

## **Создание форм, связзаанных с моделью**

## **Создание форм путем бытрого объявления**

Если же форма, связанная с моделью, используется часто, то целесообразно объявить ее явно

Класс формы, связанной с моделью, должен быть производным от класса ModelForm из модуля Django.forms. В этом классе объявленется вложенный класс Meta, в котором записывается набор атрибутов, имеющих те же имена, что параметры функции

From Django.forms import ModelForm, DecimalField

From Django.forms.widgets import Select

From .modelsl import Bb

Class BbForm(ModelForm):

Class Meta:

Model = Bb

Fields = (‘tittle’, ‘content’, ‘price’)

Labels = {‘title’: ‘Название товара’}

Help\_text = {‘rubric’: ‘Не забудь задать рубрику’}

Field\_classes = {‘price’: DecimalField}

Widgets = {‘rubric’: Select(attrs={‘size’:8})}

## **Создание форм путем полного объявления**

Опа описанных ранее способа создания форм позволяли задать для ее полей весьма ограниченный набор параметров. Если же требуется описать поля формы во всех деталях, придется прибегнуть к третьему, сложному способу объявления

При полном объявление формы в ее классе детально описываются параметры как отдельных полей, так и – во вложенном классе Meta – самой формы.

From Django import forms

From .models import Bb, Rubric

Class BbForm(forms.ModelForm):

Title = forms.CharField(label=’Название товара’)

Content = forms.CharField(label=’Описание’,

Widget=forms.widgets.Textarea())

Price = forms.DecimalField(label=’Цена’, decimal\_places=2)

Rubric = forms.ModelChoiceField(queryset=Rubric.objects.all(),

Label=’Рубрика’, help\_text=’Не забудь задать рубрику’,

Widget=forms.widgets.Select(attrs={‘size’:8}))

Class Meta:

Model = Bb

Fields = (‘title’, ’content’, ’price’, ‘rubric’)

## **Параметры поддерживаемые всеми типами полей**

1. Label – надпись для поля
2. Help\_text – дополнительный, поясняющий текст для текущего поля
3. Initial = начальное значени для поля формы
4. Required – если True, то в поле обязательно должно быть занесено значение
5. Widget – элемент управления представляющий текущее поле на веб странице.
6. Validators – валидаторы длля текущего поля
7. Error\_messages – сообзения об ошибках
8. Disabled – если True, то поле при выводе на экран станет недоступным

## **Классы полей форм**

1. CharField – строковое или текстовое поле. Дополнительные параметры(min\_length, max\_length, strip, empty\_value)
2. Emailfield – адрес электронной почты в строковом виде(min\_length, max\_lenght)
3. SlugField – слаг
4. BooleanField
5. IntegerField(min\_value, max\_value)
6. FloatField
7. DecimalField(вещественное число фиксированной точности, decimal\_places-количество цифр в дробной части числа)
8. DateField значение даты представленное в виде объекта типа date
9. DateTimeField – временная отметка в виде объекта тива datetime
10. ModelChoiceField – поле внешнего ключа вторичной модеи создающее свять один-со-многими или один-с-одним.(queryset-набор записей первичной модели на основе которого будет формироваться список, empty\_label)
11. ChoiceField – поле со списком, в которое можно занести только те значения, что приведены в списке
12. MultipleChoiceField – тоже самое что ChoiceFiled но позволяет выбрать в списке произвольное число пунктов

## **Задание элементов управления**

Любому полю формы можно сопоставить элемент управления, посредством которого в него будет заноситься значение, указав его в параметре widget поля.

## **Классы элементов управления**

Все классы элементов управления являются производными от класса widget из модуля Django.forms.widgets. Этот класс поддерживает параметр конструктора attrs, указывающий значения атрибутов HTML тега., который помещается в код генерируемой страницы и создает элемент управления.

Список элементов управления

1. TextInput – обычное поля ввода
2. NumberInput – поле для ввода числа
3. EmailInput – воле для ввода адреса электроеной почты
4. PasswordInput – поля для ввода пароля (поддерживает доп. Параметр render\_value, если присволить ему значение True, то после неудачной валидации и повторного вывода формы на экран в поле ввода пароля будет подставлен набранный ранее пароль)
5. HiddenInput – скрытое поле
6. DateInput – поле для ввода значения даты
7. Textarea – облать редактирования
8. CheckboxInput – флажок
9. Select – список, обычный или расрывающийся

## **Добавление записи посредством формы**

Для добавления записи следует создать кземпляр класса формы, поместить его в контекст шаблона и выполнить рендериг шаблона, представляющего страницу добалвения тем самым выведя форму на экран

Bbf = BbForm()

Bbf = BbForm(initial={price=1000})

## **Валидация формы**

Выполнить более сложную проверку или проверить значения сразу нескольких полей форы, можно в переопределенном методе clean(self)

## **Наборы форм связанные с моделями**

Если обычная форма, связанная с моделью, позволяет работать лишь с одной записью, то набор форм, свзяанный с моделью, дает возможнолсть сразу с несколькими записями. Внешне он представляет собой группу форм, в каждой из которых отображается содержимое одной записи. Помимо тоого, там могут быть выведены путсые формы для добавления записей и специальные средства переупорядочивания и удаления записей.

**Создание наборов форм, связанных с моделями**

# **Разграничение доступа: базовые инструменты**

## **Настройка подсистемы разграничения доступа**

На работу подсистемы влияют следующие параметры:

1. LOGIN\_URL – интернет адрес, на который будет выполнено перенаправление после попытки попасть на страницу, закрытую от неавторизованных посетителей
2. LOGIN\_REDIRECT\_URL – интернет адрес, на который будет выполнено перенаправление после успешного входа на сайт
3. LOGOUT\_REDIRECT\_URL – интернет-адрес на который будет выполнено перенаправление после успешного выхоода с сайта
4. PASSWORD\_RESET\_TIMEOUT\_DAYS – число дней, в течение которых будет действителен интернет-адрес сброса пароля, отправленный посетителю в электронном письме

## **Аутентификация и служебные процедуры**

Для выполнения аутентификации, т.е. входа на сайт, и различных служебных процедур Django предлагает ряд контроллеров, объявленных в модуле Django.contirb.auth.views

## **Контроллер LoginView: вход на сайт**

При получении запроса по HTTP методу GET он выводит на экран страницу входа с формой, в которуюю следует занести имя и пароль пользователя. При получении POST запроса, он ищет в списке пользователя с указанными именем и паролем. Если такой пользователь онбаружился выполняется перенаправление по интернет-адресу, взятому из GET или POST параметра next, или, если такой параметр отсутсвует, из параметра LOGIN\_REDIRECT\_URL настроек проекта.

Атрибуты:

1. Template\_name
2. Redirect\_field\_name – имя GET или POST-параметра, из которого будет извлекаться интернет адрес для перенаправлнеия после успешного входа
3. Redirect\_authenticated\_user – Если true, то пользователи уже выполнившие вход, при попытке попасть на страницу входа будут перенаправлены по интернет адресу, взятому из параметра next или параметру LOGIN\_REDIRECT\_URL настроек проекта
4. Authentication\_form – ссылка на класс формы входа

## **Коннтроллер LogoutView: выход с сайта**

Контроллер-класс LogoutView реализует выход с сайта при получении get запроса, после чего осущетсвляет перенаправление на интернет адрес, указанный в get-параметре next или, если такового нет, в атрибуте next\_page.

Атрибуты

1. Next\_page – тоже что и LOGOUT\_REDIRECT\_URL
2. Template\_name
3. Redirect\_field\_name – имя get параметра, из которого будет извлекаться интернет адрес для перенаправления после успешного выхода в виде строки
4. Extra\_context – дополнительное содержимое контекста шаблона. Его значение должно представлять собой словарь, элементы которого будут добавлены в контекст

## **Контроллер PasswordChangeView: смена пароля**

При получении GET запроса он выводит на экран страницу с формой, где нужно ввести старый пароль и дважды новый пароль. При получении POST запроса он сохраняет введенный новый пароль и перенаправляет пользователя на страницу с сообзением об успешной смене пароля

Атрибуты

1. Template\_name
2. Success\_utl
3. Extra\_context – дополнительное содержимое контекста. Его значение должно представлять собой словарь.
4. Form\_class – ссылка н акласс формы для ввода нового пароля

Контекст шаболн асодержит переменные

1. Form
2. Title

## **Контроллер PasswordChangeDoneView**

## **Уведомление об успешной смене пароля**

Выводит страницу с уведомлением об успешной смене пароля

Атрибуты

Template-name –

Extra\_context

## **Контроллер PasswordResetView: отправка письма для сброса пароля**

Данный контроллер инициирует процедуру сброса пароля. При поулчении get запроса он выводит страницу с формаой, в которую пользователю нужно занести свой адрес электронной почты. Послу поулчения POST запроса он проверит существование этого адреса в списке пользователей и если такой адрес есть, отпарвит по нему элекнтронное письмо с гиперссылкой на страницу собствннно сброса пароля

Атрибуты  
1) template\_name

1. Subject\_template\_name – путь к шаблону тепы электронного письма
2. Email\_template\_name – путь к шаблону тела электронногго письма в формате обыного текст
3. Success\_url
4. From\_email – адрес электронной почты отправителя, который будет вставлени в отправляемое письмо
5. Extra\_context
6. Extra\_email\_context – дополнительное содержимое контекста шаблона для электронного письма
7. Form\_class
8. Token\_generator – экземляр класса, выполняющего формирование электронного жетона безопасносит, который будет включен в интернет-адрес страницы сброса пароля

Контекст шаблона страницы содержит следующие переменные

1. Form – форма
2. Title
3. Protocol – обозначение протокола http или https
4. Domain – строка с комбинацией ip адреса и номера TCP порта, через который рабоате веб-сервер
5. Uid – закодированный ключ пользователя
6. Token – электронный жетон безопасности
7. Email – адрес электронной почты пользователя
8. User – текущий пользователь

## **Контроллер PasswordResetDoneView: уведомление об отправке письма для сброса пароля**

Выводит страницу с уведомление об успешной отправке электронного письма дл ясброаса пароля

Атрибуты

1. Template\_name
2. Extra\_context – дополнительное содержимое контекста

## **Контроллер PasswordResetConfirmView: собственно сброс пароля**

Класс выполняет сброс пароля. Он запускается при переходе по интернет адресу, отправленному в письме с сообзение о сбросе пароля. С URL параметром uidb64 он получает закодированный ключ пользователя,а с url параметром token электронный жетон безопасности, значения обоих параметров должны быть строковыми. Получив Get запрос он выводит страницу с формой для задания нового пароля, а после получения POST запроса производит смену пароля и выполняет перенаправлнеие на страницу с уведомлением об успешном сбросе пароля

Атрибуты

1. Template\_name –
2. Post\_reset\_login – елси True, то после успешного сброса пароля будет автоматически выполнен вход на сайт.
3. Success\_url
4. Extra\_context
5. Form\_class
6. Token\_generator – экземмпляр класса, выполняющего формирование электронного жетона безопасносит, который был включен в интернет адрес, вудещий на стриацнус сброса пароля
7. Reset\_url\_token – строковый фрагмент, который при вводе страницы с формой для задания новогоп пароля будет подставлени в интернет адрес вместо электронного жетона (это делаетс ядля безопасносит, чтобы никто не смог подсмотреть жетон и использовать его для атаки на сайт)

Контекст шаблона:

1. Form
2. Validlink – если True то интернет-адрес по которому пользователь попан на эту страницу, действителелн и еще н иразу не использовался, если False то этот интренет адрес скомпрометрирова
3. Title

## **Контроллер PasswordResetCompleteView: уведомление об успешном сбросе пароля**

Выводит страницу с уведомление об успешном сбросе пароля

Template\_name

Extra\_context

# **Получение сведений о пользователях**

## Авторизация

## Авторизация в контроллерах классах

Реализовать авторизацию в контроллерах-классах можно посредством классов-примесей, объявленных в модуле Django.contrib.auth.mixins. Они указываются в числе базоваых в объявлении производных контрооллеров-классовэ

Класс AccessMixin – базовый для остальных классво-примесей.Он поддерживает раяд атрибутов и методов, предназначенных для указания важных параметров авторизации.

1. Login\_url – атрибут, задает интернет-адрес или имя маршрута страницы входа
2. Permission\_denied\_message- атрибут, хранит строковое сообщение о возникшей ошибке
3. Redirect\_field\_name – атрибут, указывает имя get-параметра, передающего интернет-адрес страницы с ограниченным доступом, на которую пытался попасть посетитель
4. Reaise\_exception – атрибут. Если его значение равно True, то при попытке попасть на страницу гость или пользователь с недостаточными правами полуист сообщение об ошибке 403

Рассмотрим классы-примеси, производные от AccessMixin

1. LoginRequiredMixin – допускает к страницуе только пользователей, выполнившим вход
2. UserPassesTestMixin – допускает к странице только тез пользователей, кто выполнил вход и в чьем отношении переопределенных метод test\_func(self)
3. PermissionRequiredMixin – допускает к странице только пользователей, имеющих заданные права. Класс поддерживает допольнительные атрибуты и методы(
4. Permission\_required – атрибут, задает требуемые права, которые указываются в том же формате, который применяется в методах has\_perm())

# **Расширенные инструменты и дополнительные библиотеки**

## **Создание своих диспетчеров записей**

Диспетчер записей – это объект, предоставляющий доступ к набору записей, которые хранятся в модели.

## **Создание диспетчеров записей**

Диспетчеры записей наследуются от класса Manager. В них можно как переопределять имеющиеся методы, так и обхявлять новые

Переопределять имеет смысл только метод get\_queryset(self), который должен возвращать набор записей текущей модели в виде экземпляра класса Queryset. Обычно в теле переопределенного метода сначала вызывают тот же метод мазового класса, чтобы получить изначальный набор записей, устанавливают у него фильтрацию, сортироку, добавляют вычисляемые поля и возвращают в качестве результата

From dango.db import models

Class RubricManager(models.Manager):

Def get\_queryset(self):

Return super().get\_queryset().order\_by(‘order’, ‘name’)

Def order\_by\_bb\_count(self):

Return super().get\_queryset().annotate(cnt=models.Count(‘bb’).order\_by(‘-cnt’))

Использовать новый диспетчер записей в модели можно трояко

1. Class Rubric(models.Model):

Objects = RubricManager()

## **Создание диспетчеров обратной связи**

Аналогично можно создать свой диспетчер обратной связи, который выдает набор записей вторичной модели, связанный с текущей записью первичной модели.

From Django.db import models

Class BbManager(models.Manager):

Def get\_queryset(self):

Return super().get\_queryset().order\_by(‘price’)

# **Расширенные средства для вывода форм и наборов форм**

## **Указание CSS стилей у форм**

Для указаня css стилей, которые будут применены к отдельным элементам выводимой формы, класс форм поддерживают два атрибута

1. Required\_css\_class – имя стилевого класса, которым будут помечатьс я элементы управления, обязательные для заполнения
2. Error\_css\_class – имя стилевого класса, которым будут помечаться элементы управления с некорретными данными

# **Библиотека DjangoSimpleCapthca: поддержка CAPTCHA**

Если планируется дать пользователям-гостяп возможность добавлять какие либо данные в базу, не помешает как-то обезопасить форму, в которую вводятся эти данные от программ-роботов

В Django существует довольно много библиотек, реализующих в формах поддержку CAPTHCA. Одна из них Django Simple Captcha

**Установка Django Simple Captcha**

Pip install Django-simple-captcha

**Использование Django Simple Captcha**

В форме, в которой должна присутсвовать CAPTCHA, следует объявить поле типа CapthcaField из модуля captcha.fields.

**Настройки Django Simple Captcha**

Параметры библиотеки указываются в настройках проекта – в модуле settings.py

1. CPATCHA\_LENGTH – длина каптчи в символах текста

# **Обработка выгруженных файлов**

## **Настройка подсистемы обработки выгруженных файлов**

Настройки этой подсистемы записываются в модуле settings.py пакета конфигурации

1. MEDIA\_URL – префикс, добавляемый к интернет-адресу, выгруженного файла. Встретив в начале интренет-адреса этот префикс, Django поймет, что это выгруженный файл и его нужно передать для обработки подсистеме выгруженных файлов.
2. MEDIA\_ROOT – полный путь к папке, в которой будут храниться выгруженные файлы.

Это единственные обязательные для указания параметры подсистемы выгруженных файлов

MEDIA\_ROOT = os.path.join(BASE\_DIR, ‘media’ )

MEDIA\_URL = ‘/media/’

## **Указание маршрута для выгруженных файлов**

Практически всегода посетиители выгружают ффайлы на сайт для того, чтобы показать их другим посетителям. Следовательно на веб-страницах сайта позже будут выводиться сами эти файлы или указывающие на них гиперссылки.Для того чтобы посетители смогли их просмотреть или загрузить, нужно создать соответствующий маршрут.

Маршрут указывающий на выгруженнный файл, записывается в списке уровкня проекта, т.е. в модуле urls.py пакета конфикугарции. Для его указаня используется static() из модуля Django.conf.urls.static

From Django.conf.urls.static import static

From Django.conf import settings

Urlpatterns = […]

Urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

## **Хранение файлов в модлеях**

## **Типы полей модлеи, предназначенные для хранения файлов**

1. FileField – файл любоого типа.
2. ImageField – графический файл (max\_length, upload\_to,

Width\_field – имя поля модлеи, в которое будет записана ширина иизображения из выгруженного файла

Height\_field – тоже что и width\_field)

## **Поля форм, валидаторы и элементы управления служащие для указания файлов**

В полях этих типов можно использовать следующие валидаторы, обхявленные в модуле Django.core.validators

FileWxtensionValidator – класс проверяет входит ли расширение сохраняемого в поле файла в список допустимых

Он принимает следующие параметры

1. Allowed\_extensions – последовательность, содержащая допустимые расширения файлов.
2. Message – строка с сообщение об ошибке.
3. Code – код ошибки.

## **Для указания выгружаемых файлов применяются такие классы элементов управления из модуля Django.forms.widgets.**

FileInput – обычное поле для ввода файла

ClearableFielInput – поле для ввода фалйа с возможностью очистки.

Class ImgForm(forms.ModelForm):

Img = forms.ImageField(label=’Изображнеие’, validators=[validators.FileExtinsionValidator(allowed\_extensions=(‘gif’))], error\_messages={‘invalid\_extension’: “Этот формат не поддержвивается”})

Class Meta:

Model = Img

Fields = “\_\_all\_\_”

## **Обработка выгруженных файлов**

Обработка выгруженнгых файлов в контроллерах осуществялется так же как обработка люьых других данных полученных от посетителей. Есть лишь два момета, которые нужно иметь в виду:

1. При выводе формы на экран необходимо указать для нее метод кодирования данных myltipart/form-data, воспользовавшись атрибутом enttype тега form.
2. При повторном создании контруктора ее класса вторым позиционным параметром следует передать значение атрибута FILES объекта зпроса.

# **Разграничение доступа: расширенные инструменты и дополнительная библиотека**

## **Создание своей модели пользователя**

Для хранения списка пользователей в подсистеме разграничения доступа Django предусмотрена стандартная модель User, объявленная в модуле Django.contrib.aut.modles. Эта модель храни объем сведений о пользователе, вполне достаточный для многих случаев. Однако часто приходистя сохранять в составе сведений о пользователе дополнительные данные: номер телефона, интернет адрес сайта, признак, хочет ли пользователь получать по электронной почте уведомелняи о новых сообщениях и т.п.

Можно объявлить дополнительную моделть поместить в нее поля для хранения всех нужных данных и добавть поле, устанавливающее связь одни с одним со стандартной моделью пользователя.

Пример

From Django.db import models

From Django.contrib.auth.models import User

Class Profile(models.Model):

Phone = models.CharField(max\_length=20)

User = models.OneToOneField(User, on\_delete=models.CASCADE)

Другой подход заключается в написании своей собственной модели пользователя. Такую модель следует сделать производной от класса AbstractUser, который объявлени в модуле Django.contrib.auth.models, реальзует всю функциональность по хранению пользователей и представляет собой абстрактную модлеь – собственно, класс стандартной модели полльзователей User так же является производным от класса AbstractUser.

Пример

From Django.db import models

From Django.contrib.auth.models import AbstractUser

Class AdvUser(AbstractUser):

Phone = models.CharField(max\_length=20)

Новую модель пользователя следует указать в параметре auth\_USER\_MODEL настроек проекта

AUTHO\_USER\_MODEL = ‘testapp.models.AdvUser’

Если нужно лишь расширить или изменить функциональность модели пользователя, то можно создать на его основе прокси-модель, также не забыв занести ее в параметр AUTH\_USER\_MODEL

From Django.contrib.auth.models import User

Class AdvUser(User):

…

Class Meta:

Proxy = True

## **Создание своих прав пользователя**

Для любой модели можно создать дополнитлеьный набор произвольных прав. Для этого мы воспользуемся парметром permissions, задаваемым дл ясамой модели – во вложенном классе Meta. В качесвте его значения указывается список или кортеж каждый элемент которого описывает одно право и также представляет собой кортеж из двух элементов: обозначения, использумеого самим Django и наименования, предназначенного для вывода на экран

Class Comment(models.Model):

…

Class Meta:

Permissions = (

(‘hide\_comments’, ‘Можно скрывать комментарии’)

)

Обрабатываются эти права точно также, как и стандартные, в частности можно программно проверить, имеет ли текущий пользователь право скрывать комментарии

Def hide\_comment(requets):  
 if request.user.has\_perm(‘bboard.hide\_comments’):

Пользователь может скрывать комментарии

# **Посредники и обработчики контекста**

Посредник Django – это программный модуль, выполняющий предварительную обработку клиентского запроса перед передачей его контроллеру и окончательную обработку ответа, выданного контроллером, перед отправкой его клиенту. Список посредников, зарегестрированных в проекте, указывается в параметре MIDDLEWARE настроек проекта.

Посредники в Django можно использовать не только для обработки запросов и ответов, но и для добавления в контекст шаблона каких-либо значений. Ту же самую задачу выполняют и обработчики контекста, список которых указывается в дополнительном параметре context\_processors настроек шаблонизатора

## **Посредники**

Посредники – весьма мощный инструмент по обработке данных, пересылаемых по сети. Немалая часть функциональности Django реализована именно в посредниках.

## **Порядок выполнения посредников**

Посредники, зарегестрированные в проекте, при полученни запроса и формировании ответа выполняются дважды.

1. Первый раз – при получении запроса, перед передачей его контроллеру, в том порядке в котором записааны в списке параметра MiddleWare настроек проекта
2. Второй раз – после того, как контроллер сгенерирует отве, до отправки его клиенту. Если ответ представлени экземпляром класса TemplateREsponse, то посредники выполняются до непосредственного рендеринга шаблона.

## **Написание своих посредников**

## **Посредники-функции**

Посредники-функции прроще в написании, но предоставляют не очень много функциональных возможностей.Посредник функция должне принимать один параметр. С ним будет передан либо следующий в спеске посредник, либо контроллер

Посредник функция в качестве результатта доолжна возвращать функцию, в качестве единственного параметра принимающую запрос в виде экземпляра класса HttpRequesst.В этой внутренней функиции и будет выполняться предварительная обработка запроса и окончательная – ответа .

Шаблон, согласно которому пишутся посредники функции

Def my\_middleware(next):

# здесь можно выполнить какую либо инициализуацию

Def core\_middleware(request):

# здесь выполняется обработка клиентского запроса

Response = next(request)

# здесь выполняется обработка ответа

Return response

Return core\_middleware

## **Посредники-классы**

Посредники-классы предлагают больше функциональных возможностей, но писать их несколько сложнеее

Посредник класс должен объявлять по меньшей мере два метода:

1. Конструктор \_\_ini\_\_(self, next) – должени принять в параметре next либо следующий в списке посредник, либо контроллер и сохранить его
2. \_\_call\_\_(self, reques) – должен принимать в параметре request объект запроса и возвращать объект ответа. Тело этого метода пишется по тем же привилам что и тело внетренней фукнции у посредника фнукици

Шабло исходно кода, согласно которому пишутся посредники классы:

Class MyMiddleWare:

Def \_\_ini\_\_(self, next):

Self.next = next

# Здесь можно выполнить какую либо инициализацию

Def \_\_call\_\_(self, request):

#Здесь выполняется обработка клиентского запроса

Response = self.next(request)

#здесь выполняется обработка ответа

Return response

Дополнительно в посреднике-классе можно объявить следущие методы;

1. Process\_view(self, request, view\_func, view\_args, view\_kwargs) – выполняется непосредсвтенно перед выховом следующешо в списке посредника или контроллера
2. Process\_exception(self, request, exception) – вызывается при возюудлении исключения в теле контроллера.
3. Process\_template\_response(self, request, response) – вызывается уже после того как контроллер сгенерировал ответ, но перед рендерингом шаблона. Метод должен возвращать ответ в виде экземеляра класса TemplateResponse либо полученный с параметром response и измененный.

Метод может изменить имя шаблона, занеся его в атрибут template\_name ответа, или содержимое контекста шаблоан, доступного из атрибуда context\_data

Пример посредника RubricMiddleware, который добавляет в контекст шаблона список рубрик, взятый из модели Rubric

From .models import Rubric

Class RubricsMiddleware:

def \_\_init\_\_(self, get\_response):

Self.get\_response = get\_response

def \_\_call\_\_(self, request):

Return self.get\_response(request)

Def process\_template\_response(self, request, response):

Response.context\_data[‘rubrics’] = Rubric.objects.all()

Return rersponse

## **Обработчики контекста**

Обработчик контекста – это программный модуль, добавляющий в контекст шаблона какие либо дополнительные данные уже после формирования ответа контроллером.

Обработкии контекста удобно использовать если нужно просто добавить в контекст шаблона какие -либо данные. Обработчики контекста реализуются проще, чем посредники и работают в олюбом случае, независимо от того , представлен ответ экземляром класса TemplateRespones или HttpResponse

Обработчики контекста реализуются в виде обычной функции. Удинственный параметром она должна принимать запрос в виде экземпляра класса HttpRequest возвращать словарь с данными, которые нужно добавить в контекст шаблона

Пример, который добавляет в контекст шаблона список рубрик

From .models import Rubric

Def rubrics(request):

Return {‘rubrics’: Rubric.objects.all()}

Этот обработчик шаблона мы занесем в список параметра context\_processors

Стр 454