django-shopkit Documentation

Release 0.1

Mathijs de Bruin

June 09, 2015

Contents

1 Webshops for perfectionists with deadlines **1**

2 Project status **3**

3 Compatibility **5**

4 Dependencies **7**

5 Contents **9**
- 5.1 Getting started . . . 9
- 5.2 TODO List . . . 10
- 5.3 Writing extensions . . . 13
- 5.4 Components . . . 13

6 Indices and tables **49**

Python Module Index **51**

CHAPTER 1

Webshops for perfectionists with deadlines

Similar to the way that Django is a web application framework, django-shopkit is a webshop application framework. It is, essentially, a toolkit for building customized webshop applications, for 'perfectionists with deadlines'.

CHAPTER 2

Project status

The current codebase of *django-shopkit* is currently used to run at least two shops in a production environment, where it performs just fine. However, there is an apparent lack of documentation, which we hope to fix during the upcoming months while unrolling subsequent webshop implementations.

If you are interested in using *django-shopkit* for building your own webshop application, please contact us and we'll see how we can work together in helping you understand shopkit's internals while laying out a documentation trail in the meanwhile.

CHAPTER 3

Compatibility

Django-shopkit is compatible with Django 1.4 and 1.5.

CHAPTER 4

Dependencies

The only hard dependency of *django-shopkit* is Django 1.4 or 1.5. When available, *django-shopkit* will use *django-mptt* (< v0.6) for nested categories and *sorl-thumbnail* (< v12) for images.

CHAPTER 5

Contents

5.1 Getting started

Warning: This getting started guide as well as the *basic_webshop* demonstration project are well outdated and probably very much broken by now.
Any supplements to the documentation are much welcomed!

1. Install the app basic_webshop in your environment using PIP (better make sure you're working in a virtual environment):

```
pip install -e git://github.com/dokterbob/basic-webshop.git#egg=basic-webshop
```

2. Enable the *basic_webshop* application in *INSTALLED_APPS* in *settings.py*:

```
INSTALLED_APPS = (
    ...
    'shopkit.currency.simple',
    'basic_webshop',
    ...
)
```

3. Import the webshop settings from *basic_webshop* in *settings.py*:

```
from basic_webshop.django_settings import *
```

   Or add the settings manually:

```
SHOPKIT_CUSTOMER_MODEL = 'basic_webshop.Customer'
SHOPKIT_PRODUCT_MODEL = 'basic_webshop.Product'
SHOPKIT_CART_MODEL = 'basic_webshop.Cart'
SHOPKIT_CARTITEM_MODEL = 'basic_webshop.CartItem'
SHOPKIT_ORDER_MODEL = 'basic_webshop.Order'
SHOPKIT_ORDERITEM_MODEL = 'basic_webshop.OrderItem'
SHOPKIT_CATEGORY_MODEL = 'basic_webshop.Category'
```

4. Now include the webshop URL's in *urls.py*:

```
urlpatterns = patterns('',
    ...
    (r'^shop/', include('basic_webshop.urls')),
    ...
)
```

5. Update the database by running *syncdb*:

```
./manage.py syncdb
```

6. You now have a working basic webshop, start developing in the *src* directory in your environment. Make your own branch with:

```
git checkout -b mywebshop
```

   And edit away! Whenever you want to update your own project with changes in the *basic_webshop* project, just do:

```
git pull origin master
git merge master
```

5.2 TODO List

These are tasks specified in the source and the documentation which are marked as TODO items and hence are to be seen as work in progress.

Todo

Add methods for listing all available products (using the *in_shop* manager) for a given brand.

(The original entry is located in docstring of shopkit.brands.models.BrandBase, line 6.)

Todo

Cache this. It is a slow operation which requires as many queries as the category tree is deep.

(The original entry is located in docstring of shopkit.category.basemodels.NestedCategoryBase.get_parent_list, line 10.)

Todo

We want a setting allowing us to limit the nestedness of categories. For 'navigational' reasons, a number of 3 should be a reasonable default.

(The original entry is located in docstring of shopkit.category, line 11.)

Todo

We should consider adding a manager to `OrderBase` which can filter on the completed states.

(The original entry is located in docstring of shopkit.core.basemodels.AbstractCustomerBase.get_confirmed_orders, line 3.)

Todo

Make this lazy object: we should only perform the actual database query when this object is requested from within template.

(The original entry is located in docstring of shopkit.core.context_processors.cart, line 4.)

Todo

Use aggregation here.

(The original entry is located in docstring of shopkit.core.models.CartBase.get_total_items, line 3.)

Todo

Use aggregation here.

(The original entry is located in docstring of shopkit.core.models.OrderBase.get_total_items, line 3.)

Todo

Graceously handle errors instead of form_invalid noting that render_to_response was not found.

(The original entry is located in docstring of shopkit.core.views.CartAddBase, line 15.)

Todo

Decide whether or not to make the default success url a configuration value or not.

(The original entry is located in docstring of shopkit.core.views.CartAddBase.get_success_url, line 4.)

Todo

Make an API hook allowing us to check whether a product available for adding it to a cart.

(The original entry is located in docstring of shopkit.core.views.CartAddFormMixin.get_context_data, line 7.)

Todo

Use a setting to define the way in which prices are formatted site-wide. This way we have a simple mechanism for formatting prices everywhere on the site while leaving everything loosely coupled.

(The original entry is located in docstring of shopkit.currency, line 6.)

Todo

Provide a listing/overview of the types of DiscountMixin's available, how they should be used and... whether they have been tested or not.

(The original entry is located in docstring of shopkit.discounts.advanced.models.discount_models, line 3.)

Todo

Unittest this function.

(The original entry is located in docstring of shopkit.discounts.advanced.models.discount_models.CouponDiscountMixin.generate_c
line 4.)

Todo

Test this code.

(The original entry is located in docstring of shopkit.discounts.advanced.models.discount_models.DateRangeDiscountMixin.get_val
line 5.)

Todo

Test this! There are likely to be bugs...

(The original entry is located in docstring of shopkit.discounts.advanced.models.discount_models.ManyCategoryDiscountMixin,
line 3.)

Todo

Test this! There are likely to be bugs...

(The original entry is located in docstring of shopkit.discounts.advanced.models.discount_models.ProductDiscountMixin,
line 3.)

Todo

Provide a listing/overview of the types of DiscountMixin's available, how they should be used and... whether they have been tested or not.

(The original entry is located in docstring of shopkit.discounts.advanced.models.order_models, line 4.)

Todo

Figure out what to do when multiple discounts are valid. Really, some though should be put into this. Maybe something like a *combine_with* boolean or other customizable behaviour.

(The original entry is located in docstring of shopkit.discounts, line 3.)

Todo

Write the *is_featured* manager - and test it.

(The original entry is located in docstring of shopkit.featured.models.FeaturedProductMixin, line 5.)

Todo

Make sure the *is_featured* manager for this base model uses the *featured_order* attribute.

(The original entry is located in docstring of shopkit.featured.models.OrderedFeaturedProductMixin, line 3.)

Todo

Add a setting for returning stub images when no default image currently exists.

(The original entry is located in docstring of shopkit.images.admin.ImagesProductAdminMixin.default_image, line 4.)

Todo

This code is probably a bit too low-level.

(The original entry is located in docstring of shopkit.shipping.advanced.models.order_models.CheapestShippingMixin.get_shipping_
line 3.)

Todo

Write documentation here.

(The original entry is located in docstring of shopkit.stock.advanced, line 1.)

Todo

Write more descriptive documentation here about the stock management API's.

(The original entry is located in docstring of shopkit.stock, line 3.)

Todo

Decide whether this bugger belongs into :module:shopkit.core or whether it is just fine at it's place right here.

- Pro: We'll have a generic API for determining the stock state of items.
- Con: It's bad to have too much code in the core, it is better if modules within *django-shopkit* have the least possible knowledge about one another.

(The original entry is located in docstring of shopkit.stock.models.StockedCartItemBase, line 5.)

5.3 Writing extensions

5.3.1 Extension practises

In order to provide for extensionability, two different mechanism can be applied:

1. Whenever it makes sense to have only one extension module activated at the same time, we should opt for a combination of subclassing abstract base classes and explicitly refererring to these classes from *settings.py*. This mechanism is to be used for things like products, categories, orders and shopping carts. In some situations we might want to default to a builtin implementation of the abstract base class.

2. For other functionality, such as taxes, shipment, payment or stock management we want to allow for a mechanism similar to the way the `Django admin` is extended, using a combination of subclassing and explicit registration of plugin modules.

These approaches have the added advantage that it does not matter where in the module tree extensions are located. They might are will often reside in different packages or applications, which may not pose a problem in any way.

5.4 Components

It is of the *utmost* importance to keep core and extension functionality separated.

The criterium for this should be in the dependency tree of the respective modules: core modules should *never* depend on extension modules but are free to depend on one another. Similarly: extensions modules can freely depend on any core modules but should, generally, not depend on one another.

Furthermore: the core API should provide for hooks to allow for any of the extension modules. Hooks should be added only when an actual need for them exist and not beforehand, as we should keep the code, API and documentation as simple as possible.

Contents:

5.4.1 Core

The core of django-shopkit framework contains the components considered to be essential in the basis of any kind of webshop.

As the models and other components contained in this module are meant to be extended, they exposed in the form of abstract base classes. In an actual webshop application, they should be subclassed and, where applicable, a reference to them should be made in *settings.py* so the other components can find them.

Contents:

Admin

shopkit.core.admin

class `shopkit.core.admin.`**`PricedItemAdminMixin`**
Bases: `object`

Admin mixin for priced items.

Base models

shopkit.core.basemodels

class shopkit.core.basemodels.**AbstractCustomerBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract base class for customers of the shop.

get_all_orders()
Get all orders by the customer

get_confirmed_orders()
Get all completed orders for this customer

Todo

We should consider adding a manager to `OrderBase` which can filter on the completed states.

---

get_latest_order()
Return the lastest confirmed order

class shopkit.core.basemodels.**AbstractPricedItemBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract base class for items with a price. This only contains a *get_price* dummy function yielding a NotImplementedError. An actual *price* field is contained in the *PricedItemBase* class.

This is because we might want to get our prices somewhere else, ie. using some kind of algorithm, web API or database somewhere.

get_price(***kwargs*)
Get price for the current product.

This method _should_ be implemented in a subclass.

class shopkit.core.basemodels.**ActiveItemBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract base class for items which can be activated or deactivated.

class shopkit.core.basemodels.**ActiveItemInShopBase**(**args*, ***kwargs*)
Bases: *shopkit.core.basemodels.ActiveItemBase*

This is a subclass of *ActiveItemBase* with an *ActiveItemManager* called *in_shop* returning only items with *active=True*.

The main purpose of this class is allowing for items to be enabled or disabled in the shop's backend.

class shopkit.core.basemodels.**DatedItemBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Item for which the add and modification date are automatically tracked.

class shopkit.core.basemodels.**NamedItemBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract base class for items with a name.

class shopkit.core.basemodels.**NumberedOrderBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Base class for *Order* with invoice and order numbers.

confirm()
Make sure we set an invoice number upon order confirmation.

generate_invoice_number()
Generates an invoice number for the current order. Should be overridden in subclasses.

generate_order_number()
Generates an order number for the current order. Should be overridden in subclasses.

save(**args*, ***kwargs*)
Generate an order number upon saving the order.

class shopkit.core.basemodels.**OrderedInlineItemBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

This base class does what, actually, order_with_respect_to should do but (for now) doesn't implement very well: ordering of objects with a fk-relation to some class.

As we do not know what the class with the respective relation is, it is important to note that something like the following is added:

```
class MyOrderedInline(OrderedInlineItemBase):

    <related> = models.ForeignKey(RelatedModel)

    class Meta(OrderedInlineItemBase.Meta):
        unique_together = ('sort_order', '<related>')

    def get_related_ordering(self):
        return self.__class__.objects.filter(<related>=self.<related>)



... Or we could simply wait for the Django developers to fix
`order_with_respect_to` once and for all. (Work in progress...
See `Ticket #13 <http://code.djangoproject.com/ticket/13>`.)
```

static get_next_ordering(*related*)
Get the next ordering based upon the QuerySet <django.db.models.QuerySet.QuerySet with related items.

get_related_ordering()
Get a QuerySet <django.db.models.QuerySet.QuerySet with related items to be considered for calculating the next *sort_order*.

As we do not know in this base class what the related field(s) are, this raises a NotImplementedError. It should be subclassed with something like:

```
return self.objects.filter(<related>=self.<related>)
```

save()
If no *sort_order* has been specified, make sure we calculate the it based on the highest available current *sort_order*.

class shopkit.core.basemodels.**OrderedItemBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract base class for items that have explicit ordering.

clean()
If no *sort_order* has been specified, make sure we calculate the it based on the highest available current *sort_order*.

class shopkit.core.basemodels.**PublishDateItemBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Item with a publish date.

class shopkit.core.basemodels.**QuantizedItemBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract base class for items with a quantity field.

Models

shopkit.core.models

class shopkit.core.models.**AddressBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Base class for address models.

This base class should be used when defining addresses.

class shopkit.core.models.**CartBase**(**args*, ***kwargs*)
Bases: *shopkit.core.basemodels.AbstractPricedItemBase*

Abstract base class for shopping carts.

add_item(*product*, *quantity=1*, ***kwargs*)
Adds the specified product in the specified quantity to the current shopping Cart. This effectively creates a CartItem for the Product-Cart combination or updates it when a CartItem already exists.

When *kwargs* are specified, these are passed along to *get_item* and signify properties of the *CartItem*.

Returns added *CartItem*

classmethod from_request(*request*)
Get an existing *Cart* object from the session or return a blank one.

Returns *Cart* object corresponding with this request

get_item(*product*, *create=True*, ***kwargs*)
Either instantiates and returns a CartItem for the Cart-Product combination or fetches it from the database. The creation is lazy: the resulting CartItem is not automatically saved.

Parameters

- **create** – Whether or not to create a new object if no object was found.
- **kwargs** – If *kwargs* are specified, these signify filters or instantiation parameters for getting or creating the item.

get_items()
Gets items from the cart with a quantity > 0.

get_order_line()
Get a string representation of this *OrderItem* for use in list views.

get_price(***kwargs*)
Wraps the *get_total_price* function.

get_total_items()
Gets the total quantity of products in the shopping cart.

Todo

Use aggregation here.

---

get_total_price(***kwargs*)
Gets the total price for all items in the cart.

remove_item(*product*, ***kwargs*)
Remove item from cart.

Returns True if the item was deleted succesfully, False if the item could not be found.

to_request(*request*)
Store a reference to the current *Cart* object in the session.

class shopkit.core.models.**CartItemBase**(**args*, ***kwargs*)
Bases: *shopkit.core.basemodels.AbstractPricedItemBase*, *shopkit.core.basemodels.QuantizedItemBase*

Abstract base class for shopping cart items.

get_order_line()
Natural (unicode) representation of this cart item in an order overview.

get_parent()
Get the relevant Cart. Used to have a generic API for Carts and Orders.

get_piece_price(***kwargs*)
Gets the price per piece for a given quantity of items.

get_price(***kwargs*)
Wraps *get_total_price()*.

get_total_price(***kwargs*)
Gets the tatal price for the items in the cart.

class shopkit.core.models.**CustomerAddressBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Base class for addresses with a relation to a customer, for which the *addressee* field is automatically set when saving.

save(***kwargs*)
Default the addressee to the full name of the user if none has been specified explicitly.

class shopkit.core.models.**CustomerCartBase**(**args*, ***kwargs*)
Bases: *shopkit.core.models.CartBase*

Abstract base class for shopping carts related to a Customer.

classmethod from_request(*request*)
Get cart from request and associate with customer, if related to the authenticated user.

class shopkit.core.models.**CustomerOrderBase**(**args*, ***kwargs*)
Bases: *shopkit.core.models.OrderBase*

Abstract base class for orders with Customer management.

classmethod from_cart(*cart*)
Make sure we copy the customer from the Cart, if available.

class shopkit.core.models.**CustomerPaymentBase**(**args*, ***kwargs*)
Bases: *shopkit.core.models.PaymentBase*

Abstract base class for payment related to a Customer.

class shopkit.core.models.**OrderBase**(**args*, ***kwargs*)
Bases: *shopkit.core.basemodels.AbstractPricedItemBase*, *shopkit.core.basemodels.DatedItemBase*

Abstract base class for orders.

confirm()
Method which performs actions to be taken upon order confirmation.

By default, this method writes a log message and calls the *register_confirmation* method on all order items. It also deletes to shopping cart this order was created from.

Subclasses can use this to perform actions such as updating the stock or registering the use of a discount. When overriding, make sure this method calls its supermethods.

When subclassing this method, please make sure you implement proper safety checks in the overrides of the *prepare_confirm()* method as this method should *not* raise errors under normal circumstances as this could lead to potential data/state inconsistencies.

In general, it makes sense to connect this method to a change in order state such that it is called automatically. For example:

..todo:: Write a code example here.

classmethod **from_cart**(*cart*)
Instantiate an order based on the basis of a shopping cart, copying all the items.

get_items()
Get all order items (with a quantity greater than 0).

get_price(***kwargs*)
Wraps the *get_total_price* function.

get_total_items()
Gets the total quantity of products in the shopping cart.

Todo

Use aggregation here.

---

get_total_price(***kwargs*)
Gets the total price for all items in the order.

prepare_confirm()
Run necessary checks in order to confirm whether an order can be safely confirmed. By default this method only checks whether or not the order has already been confirmed, but could be potentially overridden by methods checking the item's stock etcetera.

Raises AlreadyConfirmedException

save(**args*, ***kwargs*)
Make sure we log a state change where applicable.

class shopkit.core.models.**OrderItemBase**(**args*, ***kwargs*)
Bases: *shopkit.core.basemodels.AbstractPricedItemBase*, *shopkit.core.basemodels.QuantizedItemBase*

Abstract base class for order items. An *OrderItem* should, ideally, copy all specific properties from the shopping cart as an order should not change at all when the objects they relate to change.

confirm()
Register confirmation of the current *OrderItem*. This can be overridden in subclasses to perform functionality such as stock keeping or discount usage administration. By default it merely emits a debug message.

When overriding, be sure to call the superclass.

classmethod **from_cartitem**(*cartitem*, *order*)
Create and populate an order item from a shopping cart item. The result is *not* automatically saved.

When the *CartItem* model has extra properties, such as variations, these should be copied over to the *OrderItem* in overrides of this function as follows:

```
class OrderItem(...):
    @classmethod
    def from_cartitem(cls, cartitem, order):
        orderitem = super(OrderItem, cls).from_cartitem(
            cartitem, order
        )

        orderitem.<someproperty> = cartitem.<someproperty>

        return orderitem
```

get_parent()
Get the relevant Order. Used to have a generic API for Carts and Orders.

get_piece_price(***kwargs*)
Gets the price per piece for a given quantity of items.

get_price(***kwargs*)
Wraps *get_total_price()*.

get_total_price(***kwargs*)
Gets the tatal price for the items in the cart.

class shopkit.core.models.**OrderStateChangeBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract base class for logging order state changes.

classmethod get_latest(*order*)
Get the latest state change for a particular order, or *None* if no *StateChange* is available.

class shopkit.core.models.**PaymentBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract base class for payments.

class shopkit.core.models.**ProductBase**(**args*, ***kwargs*)
Bases: *shopkit.core.basemodels.AbstractPricedItemBase*

Abstract base class for products in the webshop.

The *in_shop* property should be a Manager containing all the *Product* objects which should be enabled in the shop's frontend.

class shopkit.core.models.**UserCustomerBase**(**args*, ***kwargs*)
Bases: *shopkit.core.basemodels.AbstractCustomerBase*, django.contrib.auth.models.User

Abstract base class for Customers which can also be Django users.

Managers

shopkit.core.managers

class shopkit.core.managers.**ActiveItemManager**
Bases: django.db.models.manager.Manager

Manager returning only activated items, ideally for subclasses of AbstractActiveItemBase.

get_query_set()
Filter the original queryset so it returns only items with *active=True*.

Views

shopkit.core.views

class shopkit.core.views.**CartAddBase**(***kwargs*)
Bases: django.views.generic.base.TemplateResponseMixin, *shopkit.core.views.CartAddFormMixin*, django.views.generic.edit.BaseFormView

View for processing POST requests adding items to the shopping cart. Process flow is as follows:

1. User is on a product detail page.
2. User clicks 'Add to cart' and (optionally) selects a quantity. This initiates a POST request to the current view.
3. The current view fetches the cart, checks for the current product in there.
4. (a)If it does, it adds the given quantity to CartItem which has been found.

   (b)If it does not, a new CartItem should be created and added to the users Cart.

5.Redirect to the cart view.

Todo

Graceously handle errors instead of form_invalid noting that render_to_response was not found.

form_valid(*form*)
Form data was valid: add a CartItem to the Cart or increase the number.

..todo:: Refactor this!

get_form_class()
Simply wrap the get_cart_form_class from CartMixin.

get_success_url()
The URL to return to after the form was processed succesfully. This function should be overridden.

Todo

Decide whether or not to make the default success url a configuration value or not.

class shopkit.core.views.**CartAddFormMixin**
Bases: object

Mixin providing a basic form class for adding items to the shopping cart. It will be added to the context as *cartaddform*.

get_cart_form_class()
Simply return the form for adding Items to a Cart.

get_context_data(***kwargs*)
Add a cart add form under the name *cartaddform* to the context, if and only if an object is available and is a product.

If this is not the case, we should fail silently (perhaps) logging a debug message.

Todo

Make an API hook allowing us to check whether a product available for adding it to a cart.

class shopkit.core.views.**InShopViewMixin**
Bases: object

Mixin using the *in_shop* manager rather than the default *objects*, so that it only uses objects which are actually enabled in the frontend of the shop.

get_queryset()
Return *in_shop.all()* for the *model*.

Forms

shopkit.core.forms

class shopkit.core.forms.**CartItemAddForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=u':'*, *empty_permitted=False*)
Bases: django.forms.forms.Form

Form for adding CartItems to a Cart.

shopkit.core.forms.**get_product_choices**()
Get available products for shopping cart. This has to be wrapped in a SimpleLazyObject, otherwise Sphinx will complain in the worst ways.

Settings

shopkit.core.settings

To set the values below from *settings.py*, prepend their names with *SHOPKIT_*. For example:

```
SHOPKIT_PRODUCT_MODEL = 'myapp.MyProduct'
```

Utils

shopkit.core.utils

shopkit.core.utils.**get_model_from_string**(*model*)
Takes a string in the form of *appname.Model*, (ie. *basic_webshop.CartItem*) and returns the model class for it.

Contents:

Fields

shopkit.core.utils.fields

class shopkit.core.utils.fields.**MinMaxDecimalField**(***kwargs*)
Bases: django.db.models.fields.DecimalField

DecimalField subclass which allows specifying a minimum and maximum value. Takes two extra optional parameters, to be specified as a Decimal or string:

- *max_value*
- *min_value*

class shopkit.core.utils.fields.**PercentageField**(***kwargs*)
Bases: *shopkit.core.utils.fields.MinMaxDecimalField*

Subclass of *DecimalField* with sensible defaults for percentage discounts:

- *max_value=100*
- *min_value=0*
- *decimal_places=0*
- *max_digits=3*

Admin

shopkit.core.utils.admin

class shopkit.core.utils.admin.**LimitedAdminInlineMixin**
Bases: object

InlineAdmin mixin limiting the selection of related items according to criteria which can depend on the current parent object being edited.

A typical use case would be selecting a subset of related items from other inlines, ie. images, to have some relation to other inlines.

Use as follows:

```
    class MyInline(LimitedAdminInlineMixin, admin.TabularInline):
        def get_filters(self, obj):
            return (('<field_name>', dict(<filters>)),)
```

get_filters(*obj*)
Return filters for the specified fields. Filters should be in the following format:

```
(('field_name', {'categories': obj}), ...)
```

For this to work, we should either override *get_filters* in a subclass or define a *filters* property with the same syntax as this one.

get_formset(*request*, *obj=None*, ***kwargs*)
Make sure we can only select variations that relate to the current item.

static limit_inline_choices(*formset*, *field*, *empty=False*, ***filters*)
This function fetches the queryset with available choices for a given *field* and filters it based on the criteria specified in filters, unless *empty=True*. In this case, no choices will be made available.

Listeners

shopkit.core.utils.listeners

class shopkit.core.utils.listeners.**Listener**(***kwargs*)
Bases: object

Class-based listeners, based on Django's class-based generic views. Yay!

Usage:

```
class MySillyListener(Listener):
    def dispatch(self, sender, **kwargs):
        # DO SOMETHING
        pass

funkysignal.connect(MySillyListener.as_view(), weak=False)
```

classmethod as_listener(***initkwargs*)
Main entry point for a sender-listener process.

Context processors

shopkit.core.context_processors

shopkit.core.context_processors.**cart**(*request*)
Request context processor adding the shopping cart to the current context as *cart*.

Todo

Make this lazy object: we should only perform the actual database query when this object is requested from within template.

Tests

shopkit.core.tests

class shopkit.core.tests.**CoreTestMixin**
Bases: object

Base class for testing core webshop functionality. This class should not directly be used, rather it should be subclassed similar to the way that included model base classes should be subclassed.

make_product()
Abstract function for creating a test product. As the actual properties of Products depend on the classes actually implementing it, this function must be overridden in subclasses.

setUp()
This function gets the model classes from *settings.py* and makes them available as *self.cusomter_class*, *self.product_class* etcetera.

test_basic_product()
Test if we can create and save a simple product.

test_cart()
Create a shopping cart with several products, quantities and prices.

test_cartitem_from_product()
Create a *CartItem* from a *Product*.

test_create_usercustomer()
Create a *UserCustomer*.

test_order()
Create an order on the basis of a shopping cart and a customer object.

test_orderitem_from_cartitem()
Create an *OrderItem* from a *CartItem*.

test_orderstate_change_tracking()
Change the state of an order, see if the state change gets logged.

Exceptions

shopkit.core.exceptions

exception shopkit.core.exceptions.**AlreadyConfirmedException**(*order*)
Bases: exceptions.Exception

Exception raised when confirmation is attempted for an order which has already been confirmed.

exception shopkit.core.exceptions.**ShopKitExceptionBase**
Bases: exceptions.Exception

Base class for exception in django-shopkit.

Signals

shopkit.core.signals

Listeners

shopkit.core.listeners

class shopkit.core.listeners.**EmailingListener**(***kwargs*)
Bases: *shopkit.core.utils.listeners.Listener*

Listener which sends out emails.

create_message(*context*)
Create an email message.

get_body_template_names()
Returns a list of template names to be used for the request. Must return a list. May not be called if render_to_response is overridden.

get_context_data()
Context for the message template rendered. Defaults to sender, the current site object and kwargs.

get_recipients()
Get recipients for the message.

get_sender()
Sender of the message, defaults to *None* which imples *DEFAULT_FROM_EMAIL*.

get_subject_template_names()
Returns a list of template names to be used for the request. Must return a list. May not be called if render_to_response is overridden.

handler(*sender*, ***kwargs*)
Store sender and kwargs attributes on self.

class shopkit.core.listeners.**StateChangeListener**(***kwargs*)
Bases: *shopkit.core.utils.listeners.Listener*

Listener base class for order status changes.

Example:

```
OrderPaidListener(StatusChangeListener):
    state = order_states.ORDER_STATE_PAID

    def handler(self, sender, **kwargs):
        # <do something>
```

dispatch(*sender*, ***kwargs*)
The dispatch method is equivlant to the similarly named method in Django's class based views: it checks whether or not this signal should be handled at all (whether or not it matches the specified) state change) and then calls the *handle()* method.

handler(*sender*, ***kwargs*)
The handler performs some actual action upon handling a signal. This must be overridden in subclasses defining *actual* listeners.

class shopkit.core.listeners.**StateChangeLogger**(***kwargs*)
Bases: *shopkit.core.listeners.StateChangeListener*

Debugging listener for *order_state_change*, logging each and every state change.

handler(*sender*, ***kwargs*)
Handle the signal by writing out a debug log message.

class shopkit.core.listeners.**TranslatedEmailingListener**(***kwargs*)
Bases: *shopkit.core.listeners.EmailingListener*

Email sending listener which switched locale before processing.

get_language(*sender*, ***kwargs*)
Return the language we should switch to.

handler(*sender*, ***kwargs*)
Handle the signal, wrapping the emailing handler from the base class but changing locale on beforehand, switching back to the original afterwards.

5.4.2 Price

shopkit.price By default, this extension contains base classes for two different types of pricing:

- Simple pricing gives you an abstract base class that simply adds a price field to the *ProductBase* class.
- Advanced pricing, allowing several prices to be specified per product, depending on factors such as the date or the amount of articles. The current structure for this code is very preliminary and mostly demonstrative.

Contents:

Models

class shopkit.price.models.**PricedItemBase**(**args*, ***kwargs*)
Bases: *shopkit.core.basemodels.AbstractPricedItemBase*

Abstract base class for priced models with a price field. This base class simply has a price field for storing the price of the item.

get_price(***kwargs*)
Returns the price property of the current product.

Simple

shopkit.price.simple As of now this is only a stub module using the base class for priced items found in the core. However, functionality might be added later which is specific to products, so please use this extension instead of the core base classes.

Contents:

Models

Advanced

shopkit.price.advanced The model structure in this extension is very preliminary. Ideally, one would want all ones prices to reside in a single table.

One way to approach this would be using a private function *_get_valid* for *PriceBase* subclasses and then implementing a *get_valid* in *PriceBase* which calls the *_get_valid* functions for direct parent classes that inherit from *PriceBase*. This could then be collapsed into a single QuerySet using Q objects. But perhaps this is too complicated. Any comments welcomed.

Models

shopkit.price.advanced.models

class shopkit.price.advanced.models.**DateRangedPriceMixin**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Base class for a price that is only valid within a given date range.

classmethod get_valid_prices(*date=None*, **args*, ***kwargs*)
Return valid prices for a specified date, taking the current date if no date is specified.

class shopkit.price.advanced.models.**PriceBase**(**args*, ***kwargs*)
Bases: *shopkit.price.models.PricedItemBase*

Abstract base class for price models, exposing a method to get the cheapest price under given conditions.

Be sure to add the proper *unique_together* contraints to subclasses implementing an actual price model.

classmethod get_cheapest(***kwargs*)
Get the cheapest available price under given conditions.

classmethod get_valid_prices(***kwargs*)
Get valid prices (as a QuerySet), given certain constraints. By default, this returns all prices available. Where applicable, subclasses might filter this result by:

- Product
- Date
- Quantity

class shopkit.price.advanced.models.**ProductPriceMixin**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Represents prices available for a specific product product.

classmethod get_valid_prices(*product*, **args*, ***kwargs*)
Return valid prices for a specified product

class shopkit.price.advanced.models.**QuantifiedPriceMixin**(**args*, ***kwargs*)
Bases: *shopkit.core.basemodels.QuantizedItemBase*

Base class for a price that is only valid above a certain quantity.

classmethod get_valid_prices(*quantity=1*, **args*, ***kwargs*)
Get valid prices for a given quantity of items. If no quantity is given, 1 is assumed.

Settings

shopkit.price.advanced.settings

Admin

shopkit.price.advanced.admin

class shopkit.price.advanced.admin.**PriceInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

Inline price admin for prices belonging to products.

formset
alias of PriceInlineFormSet

Forms

shopkit.price.advanced.forms

class shopkit.price.advanced.forms.**PriceInlineFormSet**(*data=None*, *files=None*, *instance=None*, *save_as_new=False*, *prefix=None*, *queryset=None*, ***kwargs*)
Bases: django.forms.models.BaseInlineFormSet

Formset which makes sure that at least one price is filled in.

clean()
Raise a ValidationError if no Price forms are filled in.

Tests

shopkit.price.advanced.tests

class shopkit.price.advanced.tests.**AdvancedPriceTestMixin**
Bases: object

Base class for testing advanced prices.

setUp()
This makes the *Price* class from the *SHOPKIT_PRICE_MODEL* available as *self.price_class* for unittests to make use of.

5.4.3 Shipping

Generic code for calculating shipping costs and methods.

Advanced

shopkit.shipping.advanced Advanced shipping allows for multiple shipping methods and algorithms.

Models

shopkit.shipping.advanced.models

class shopkit.shipping.advanced.models.shipping_models.**ItemShippingMethodMixin**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

Mixin for shipping methods which process individual items and not whole orders.

classmethod get_cheapest(***kwargs*)

Return the cheapest shipping method or *None*.

If *item_cost* is not specified, an attempt will be made to call *get_cheapest* on the superclass. If this method does not exist in the superclass, *None* is returned.

classmethod get_valid_methods(*item_methods=None*, ***kwargs*)

We want to be able to discriminate between methods valid for the whole item and those valid for item items.

Parameters **item_methods** – When *True*, only items for which *item_cost* has been specified are valid. When *False*, only items which have no *item_cost* specified are let through.

class shopkit.shipping.advanced.models.shipping_models.**MinimumItemAmountShippingMixin**(**args* ***kwa*

Bases: django.db.models.base.Model

Shipping mixin for methods valid only from a specified order amount.

classmethod get_valid_methods(*item_price=None*, ***kwargs*)

Return shipping methods for which the item price is above a minimum price or ones for which no minimal item price has been specified.

Parameters **item_price** – Price for the current *OrderItem*, used to determine valid shipping methods.

class shopkit.shipping.advanced.models.shipping_models.**MinimumOrderAmountShippingMixin**(**arg* ***k*

Bases: django.db.models.base.Model

Shipping mixin for methods valid only from a specified order amount.

classmethod get_valid_methods(*order_price=None*, ***kwargs*)

Return shipping methods for which the order price is above the minimal order price or ones for which no minimal order price has been specified.

Parameters **order_price** – Price for the current order, used to determine valid shipping methods.

class shopkit.shipping.advanced.models.shipping_models.**OrderShippingMethodMixin**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

Mixin for shipping methods which process whole orders and not individual items.

classmethod get_cheapest(***kwargs*)

Return the cheapest order shipping method if *order_methods* is specified. Return whatever it is the superclass returns otherwise.

classmethod get_valid_methods(*order_methods=None*, ***kwargs*)
We want to be able to discriminate between methods valid for the whole order and those valid for order items.

Parameters **order_methods** – When *True*, only items for which *order_cost* has been specified are valid. When *False*, only items which have no *order_cost* specified are let through.

class shopkit.shipping.advanced.models.shipping_models.**ShippingMethodBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Base class for shipping methods.

get_cost(***kwargs*)
Get the total shipping costs resulting from this *ShippingMethod*. This method should be implemented by subclasses of *:class:ShippingMethodBase*.

classmethod get_valid_methods(***kwargs*)
Get all valid shipping methods for a given *kwargs*. By default, all methods are valid.

is_valid(***kwargs*)
Check to see whether an individual method is valid under the given circumstances.

class shopkit.shipping.advanced.models.order_models.**AutomaticShippingMixin**
Bases: object

Mixin class for shippable items for which the choice of method is automatic.

get_shipping_method(***kwargs*)
Return the shipping method used for the current item. This method should be overridden in subclasses. It should return *None* if shipping is not applicable for this item and hence, the shipping costs should be 0.

class shopkit.shipping.advanced.models.order_models.**CheapestShippingMixin**
Bases: *shopkit.shipping.advanced.models.order_models.AutomaticShippingMixin*

Shippable item which defaults to using the

get_shipping_method(***kwargs*)
Return the cheapest shipping method or an order or item.

Todo

This code is probably a bit too low-level.

---

class shopkit.shipping.advanced.models.order_models.**PersistentShippedItemBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Mixin class for *Order*'s and *OrderItem*'s for which the shipping method is stored persistently upon calling the *update_shipping* method.

update_shipping()
Call *update_shipping* on the superclass and get the shipping method, store the resulting *ShippingMethod* on the *shipping_method* property.

class shopkit.shipping.advanced.models.order_models.**ShippedCartItemMixin**(**args*, ***kwargs*)
Bases: shopkit.shipping.advanced.models.order_models.CalculatedShippingItemMixin, *shopkit.shipping.advanced.models.order_models.CheapestShippingMixin*, *shopkit.shipping.basemodels.ShippedCartItemBase*

Base class for shopping cart items which are shippable.

class shopkit.shipping.advanced.models.order_models.**ShippedCartMixin**(**args*, ***kwargs*)
Bases: shopkit.shipping.advanced.models.order_models.CalculatedShippingOrderMixin,

shopkit.shipping.advanced.models.order_models.CheapestShippingMixin, *shopkit.shipping.basemodels.ShippedCartBase*

Base class for shopping carts with shippable items.

class shopkit.shipping.advanced.models.order_models.**ShippedOrderItemMixin**(**args*, ***kwargs*)

Bases: *shopkit.shipping.advanced.models.order_models.PersistentShippedItemBase*, shopkit.shipping.advanced.models.order_models.CalculatedShippingItemMixin, *shopkit.shipping.advanced.models.order_models.CheapestShippingMixin*, *shopkit.shipping.basemodels.ShippedOrderItemBase*

Base class for orderitems which can have individual shipping costs applied to them.

class shopkit.shipping.advanced.models.order_models.**ShippedOrderMixin**(**args*, ***kwargs*)

Bases: *shopkit.shipping.advanced.models.order_models.PersistentShippedItemBase*, *shopkit.shipping.basemodels.ShippedOrderBase*, shopkit.shipping.advanced.models.order_
shopkit.shipping.advanced.models.order_models.CheapestShippingMixin

Base class for orders with a shipping_method.

Settings

Admin

Settings

shopkit.shipping.settings

Models

shopkit.shipping.models

class shopkit.shipping.models.**ShippableCustomerMixin**

Bases: object

Customer Mixin class for shops in which orders make use of a shipping address.

get_recent_shipping()

Return the most recent shipping address

Base models

shopkit.shipping.basemodels

class shopkit.shipping.basemodels.**ShippedCartBase**(**args*, ***kwargs*)

Bases: *shopkit.shipping.basemodels.ShippedItemBase*

Mixin class for shopping carts with shipping costs associated with them.

get_order_shipping_costs(***kwargs*)

Get the shipping costs for this order. Must be implemented in subclasses.

get_total_shipping_costs(***kwargs*)

Get the total shipping cost for this *Cart*, summing up the shipping costs for the whole order and those for individual items (where applicable).

class shopkit.shipping.basemodels.**ShippedCartItemBase**(**args*, ***kwargs*)

Bases: *shopkit.shipping.basemodels.ShippedItemBase*

Mixin class for *CartItemz*'s with a function *get_shipping_costs()*.

class shopkit.shipping.basemodels.**ShippedItemBase**(**args*, ***kwargs*)
Bases: *shopkit.core.basemodels.AbstractPricedItemBase*

Base class for shippable items.

get_price(***kwargs*)
Get the price with shipping costs applied.

get_price_without_shipping(***kwargs*)
Get the price without shipping costs.

get_shipping_costs(***kwargs*)
Return the most sensible shipping cost associated with this item. By default, it returns the total shipping cost as yielded by *get_total_shipping_costs*.

get_total_shipping_costs(***kwargs*)
Return the total shipping applicable for this item. Must be implemented in subclasses.

class shopkit.shipping.basemodels.**ShippedOrderBase**(**args*, ***kwargs*)
Bases: *shopkit.shipping.basemodels.ShippedItemBase*

Mixin class for orders with shipping costs associated with them.

get_order_shipping_costs(***kwargs*)
Get the shipping costs for this order.

get_total_shipping_costs(***kwargs*)
Get the total shipping cost for this *Cart*, summing up the shipping costs for the whole order and those for individual items (where applicable).

update_shipping()
Update the shipping costs for order and order items.

class shopkit.shipping.basemodels.**ShippedOrderItemBase**(**args*, ***kwargs*)
Bases: *shopkit.shipping.basemodels.ShippedItemBase*

Mixin class for *OrderItem*'s with shipping costs associated with them.

get_shipping_costs(***kwargs*)
Return the shipping costs for this item.

update_shipping()
Update shipping costs - does *not* save the object.

5.4.4 Category

shopkit.category Django-shopkit, by default, contains base classes for two kinds of categories:

- Simple categories, which define a base class for products that belong to exactly one category.
- Advanced categories, that belong to zero or more categories.

Furthermore, generic abstract base models are defined for 'normal' categories and for nested categories, allowing for the hierarchical categorization of products.

Todo

We want a setting allowing us to limit the nestedness of categories. For 'navigational' reasons, a number of 3 should be a reasonable default.

Contents:

Base models

class shopkit.category.basemodels.**CategoryBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract base class for a category.

The *in_shop* property should be a Manager containing all the items which should be enabled in the shop's frontend.

classmethod get_categories()
Gets all the available categories.

classmethod get_main_categories()
Gets the main categories, which for unnested categories implies all of them. This method exists purely for uniformity reasons.

get_products()
Get all active products for the current category.

class shopkit.category.basemodels.**NestedCategoryBase**(**args*, ***kwargs*)
Bases: *shopkit.category.basemodels.CategoryBase*

Abstract base class for a nested category.

classmethod get_main_categories()
Gets the main categories; the ones which have no parent.

get_parent_list(*reversed=False*)
Return a list of all parent categories of the current category.

By default it lists the categories from parent to child, ie.:

```
[<categoryt>, <subcategory>, <subsubcategory>, ...]
```

If the argument *reversed* evaluates to *True*, the list runs in reverse order. This *saves* an extra reverse operation.

Todo

Cache this. It is a slow operation which requires as many queries as the category tree is deep.

---

get_products()
Get all active products for the current category.

For performance reasons, and added control, this only returns only products explicitly associated to this category - as opposed to listing also products in subcategories of the current category.

This would take a lot more requests and is probably not what we should wish for.

get_subcategories()
Gets the subcategories for the current category.

class shopkit.category.basemodels.**MPTTCategoryBase**(**args*, ***kwargs*)
Bases: mptt.models.MPTTModel, *shopkit.category.basemodels.NestedCategoryBase*

classmethod get_main_categories()
Gets the main categories; the ones which have no parent.

get_products()
Get all active products for the current category.

As opposed to the original function in the base class, this also includes products in subcategories of the current category object.

get_subcategories()
Gets the subcategories for the current category.

Settings

shopkit.category.settings

To set the values below from *settings.py*, prepend their names with *SHOPKIT_*. For example:

```
SHOPKIT_CATEGORY_MODEL = 'myapp.MyCategory'
```

Tests

shopkit.category.tests

class shopkit.category.tests.**CategoryTestMixinBase**
Bases: object

Base class for testing categories.

make_category()
Abstract function for creating a test category. As the actual properties of Products depend on the classes actually implementing it, this function must be overridden in subclasses.

setUp()
We want to have the category class available in *self*.

test_basic_category()
Test if we can make and save a simple category.

Simple

shopkit.category.simple Simple category support, allowing products to only belong to a single category.

Models

class shopkit.category.simple.models.**CategorizedItemBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Advanced base class for a simple categorized item, belonging to only once single category.

Views

class shopkit.category.simple.views.**CategoriesMixin**
Bases: object

View Mixin providing a list of categories.

get_context_data(***kwargs*)
Adds the available categories to the context as *categories*.

Tests

class shopkit.category.simple.tests.**CategoryTestMixin**
Bases: *shopkit.category.tests.CategoryTestMixinBase*

Test base class for simple categories.

Advanced

shopkit.category.advanced

Models

class shopkit.category.advanced.models.**CategorizedItemBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract base class for an advanced categorized item, possibly belonging to multiple categories.

Views

Tests

class shopkit.category.advanced.tests.**CategoryTestMixin**
Bases: *shopkit.category.tests.CategoryTestMixinBase*

Test base class for advanced categories.

5.4.5 Value Added Tax

VAT Extension, implementing Value Added Tax mechanisms for products, carts and orders.

Simple

shopkit.vat.simple

Models

class shopkit.vat.simple.models.**VATItemBase**(**args*, ***kwargs*)
Bases: *shopkit.core.basemodels.AbstractPricedItemBase*

This item extends any priced item (subclasses of AbstractPricedItemBase) with functions that yield the prices with and without VAT. In doing this, it might be imported in what order the base classes for the VAT'ed item are listed. Feedback about this is welcomed.

..todo:: Write unittests for this piece of code.

get_price(*with_vat=True*, ***kwargs*)
If *with_vat=False*, simply returns the original price. Otherwise it takes the result of *get_vat()* and adds it to the original price.

get_price_with_vat(***kwargs*)
Gets the price including VAT. This is a wrapper function around get_price as to allow for specific prices to be queried from within templates.

get_price_without_vat(***kwargs*)
Gets the price excluding VAT. This is a wrapper function around get_price as to allow for specific prices to be queried from within templates.

get_vat(***kwargs*)
Gets the amount of VAT for the current item.

Views

Settings

Advanced

shopkit.vat.advanced

Models

Views

Settings

5.4.6 Currency

shopkit.currency Currency handling for django-shopkit. It comes in a simple and an advanced variant. The simple variant assumes a single currency throughout the webshop project, advanced currency support allows for using multiple currencies throughout the site.

Todo

Use a setting to define the way in which prices are formatted site-wide. This way we have a simple mechanism for formatting prices everywhere on the site while leaving everything loosely coupled.

Contents:

Simple

shopkit.currency.simple Simple currency support. This assumes a single currency throughout the webshop, configured in *settings.py* as *SHOPKIT_CURRENCY_DEFAULT*.

Contents:

Settings

Utils

shopkit.currency.simple.utils.**format_price**(*amount*)
Format the given float-like object in the current locale.

Fields

class shopkit.currency.simple.fields.**PriceField**(***kwargs*)
Bases: django.db.models.fields.DecimalField

A PriceField is simply a subclass of DecimalField with common defaults set by *CURRENCY_MAX_DIGITS* and *CURRENCY_DECIMALS*.

Advanced

shopkit.currency.advanced Contents:

Models

Views

Settings

5.4.7 Configurable

Configurable products with several properties and variants.

Simple

shopkit.configurable.simple

Models

Views

Settings

Advanced

shopkit.configurable.advanced

Models

Views

Settings

5.4.8 Variations

shopkit.variations Base classes for *Product*, *Cart* and *OrderItem* models which have several variations. The base classes have no natural properties or fields, these have to be defined in the application which uses the variations.

Within the models module, two kinds of models are defined:

1. Unordered variations.
2. Ordered variations - for which an integer ordering has to be specified.

Contents:

Admin

shopkit.variations.admin

class `shopkit.variations.admin.`**`ProductVariationInline`**(*parent_model*, *admin_site*)
Bases: `django.contrib.admin.options.TabularInline`

Inline admin for product variations.

Models

shopkit.variations.models

class `shopkit.variations.models.`**`OrderedProductVariationBase`**(**args*, ***kwargs*)
Bases: *shopkit.variations.models.ProductVariationBase*, *shopkit.core.basemodels.OrderedInlineItemBase*

Base class for ordered product variations.

classmethod **`get_default_variation`**()
By default, this returns the first variation according to the default sortorder.

`get_related_ordering`()
Related objects for generating default ordering.

class shopkit.variations.models.**ProductVariationBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Base class for variations of a product.

classmethod get_default_variation()
Return the default variation selected for this product. As there is no inherent way to order these, this function should be overridden in classes actually implementing the variation model.

This might, for example, be overriden by taking the first product in the list or by some function selecting a specific variation as default.

class shopkit.variations.models.**VariationCartItemMixin**(**args*, ***kwargs*)
Bases: django.db.models.base.Model, *shopkit.variations.models.VariationItemBase*

Mixin class for cart items which can have variations.

class shopkit.variations.models.**VariationItemBase**
Bases: object

Abstract base class for (order/cart) items with variations.

class shopkit.variations.models.**VariationOrderItemMixin**(**args*, ***kwargs*)
Bases: django.db.models.base.Model, *shopkit.variations.models.VariationItemBase*

Mixin class for order items which can have variations.

classmethod from_cartitem(*cartitem*, *order*)
Create *OrderItem* from *CartItem*.

Settings

shopkit.variations.settings

5.4.9 Images

shopkit.images Images extension, allowing us to attach images to products using an ImageField. It provides a *ProductImageBase* abstract base class and a setting for defining the actual class implementing product images. This extension also provides an AdminInline class for updating product images from within the Admin interface and an Admin Mixin for showing thumbnails from within the list view.

This extension is loosely coupled to the new sorl-thumbnail - it will scale thumbnails when Sorl is available but should work just fine without it.

Contents:

Admin

shopkit.images.admin

class shopkit.images.admin.**ImagesProductAdminMixin**
Bases: object

Mixin class adding a function for easily displaying images in the product list display of the admin. To use this, simply add *'default_image'* to the *list_display* tuple.

Like such:

```
ProductAdmin(ImagesProductAdminMixin, <Base classes>):
   list_display = ('name', 'default_image')
```

default_image(*obj*)
Renders the default image for display in the admin list. Makes a thumbnail if *sorl-thumbnail* is available.

Todo

Add a setting for returning stub images when no default image currently exists.

class shopkit.images.admin.**ProductImageInline**(*parent_model*, *admin_site*)
Bases: sorl.thumbnail.admin.current.AdminImageMixin, django.contrib.admin.options.TabularInline

Inline admin for product images.

Models

shopkit.images.models

class shopkit.images.models.**ImagesProductMixin**
Bases: object

Mixin representing a product with multiple images associated to it.

get_default_image()
By default, this returns the first image according to whatever sortorder is used.

class shopkit.images.models.**OrderedProductImageBase**(**args*, ***kwargs*)
Bases: *shopkit.images.models.ProductImageBase*, *shopkit.core.basemodels.OrderedInlineIte*

Base class for explicitly ordere image relating to a product.

get_related_ordering()
Related objects for generating default ordering.

class shopkit.images.models.**ProductImageBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Base class for image relating to a product.

Settings

shopkit.images.settings

5.4.10 Discounts

shopkit.discounts Base classes for discounts.

Todo

Figure out what to do when multiple discounts are valid. Really, some though should be put into this. Maybe something like a *combine_with* boolean or other customizable behaviour.

Contents:

Base models

shopkit.discounts.basemodels

class shopkit.discounts.basemodels.**DiscountedCartBase**(**args*, ***kwargs*)
Bases: *shopkit.discounts.basemodels.DiscountedItemBase*

Base class for shopping carts which can have discounts applied to them.

get_order_discount(***kwargs*)
Calculate the whole order discount, as distinct from discount that apply to specific order items. This method must be implemented elsewhere.

get_total_discount(***kwargs*)
Return the total discount. This consists of the sum of discounts applicable to orders and the discounts applicable to items.

class shopkit.discounts.basemodels.**DiscountedCartItemBase**(**args*, ***kwargs*)
Bases: *shopkit.discounts.basemodels.DiscountedItemBase*

Base class for shopping cart items which can have discounts applied to them.

get_item_discount(***kwargs*)
Calculate the order item discount, as distinct from the whole order discount. This method must be implemented in elsewhere.

get_total_discount(***kwargs*)
Return the total discount for the CartItem, which is simply a wrapper around *get_item_discount*.

class shopkit.discounts.basemodels.**DiscountedItemBase**(**args*, ***kwargs*)
Bases: *shopkit.core.basemodels.AbstractPricedItemBase*

Mixin class for discounted items.

get_discount(***kwargs*)
Return the most sensible discount related to this item. By default, it returns the total discount applicable as yielded by *get_total_discount*.

..todo:: The mechanism making sure the discount is never higher than the original price is implemented here as well as in get_total_discount of DiscountedCartBase and DiscountedOrderBase.

get_piece_discount(***kwargs*)
Get the discount per piece. Must be implemented in subclasses.

get_piece_price_with_discount(***kwargs*)
Get the piece price with the discount applied.

get_piece_price_without_discount(***kwargs*)
The price per piece without discount. Wrapper around the *get_piece_price* method of the superclass.

get_price(***kwargs*)
Get the price with the discount applied.

get_price_without_discount(***kwargs*)
The price without discount. Wrapper around the *get_price* method of the superclass.

get_total_discount(***kwargs*)
Return the total discount applicable for this item. Must be implemented in subclasses.

class shopkit.discounts.basemodels.**DiscountedOrderBase**(**args*, ***kwargs*)
Bases: *shopkit.discounts.basemodels.DiscountedItemBase*

Base class for orders which can have discounts applied to them. This stores rather than calculates the discounts for order persistence.

get_order_discount()
Return the discount for this order. This basically returns the *order_discount* property. To recalculate/update this value, call the *update_discount* method.

get_total_discount(***kwargs*)
Return the total discount. This consists of the sum of discounts applicable to orders and the discounts applicable to items.

update_discount()
Update discounts for order and order items

class shopkit.discounts.basemodels.**DiscountedOrderItemBase**(**args*, ***kwargs*)
Bases: *shopkit.discounts.basemodels.DiscountedItemBase*

Base class for order items which can have discounts applied to them.

get_item_discount(***kwargs*)
Return the discount for this item. This basically returns the *discount* property. To recalculate/update this value, call the *update_discount* method.

get_total_discount(***kwargs*)
Return the total discount for the CartItem, which is simply a wrapper around *get_item_discount*.

update_discount()
Update the discount

Settings

shopkit.discounts.settings

Advanced discounts

shopkit.discounts.advanced Contents:

Admin

shopkit.discounts.advanced.admin

Models

shopkit.discounts.advanced.models Model base and mixin classes for building discount model and logic.

Todo

Provide a listing/overview of the types of DiscountMixin's available, how they should be used and... whether they have been tested or not.

---

class shopkit.discounts.advanced.models.discount_models.**AccountedUseDiscountMixin**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

Mixin class for discounts for which the number of uses is accounted.

classmethod register_use(*qs*, *count=1*)
Register *count* uses of discounts in queryset *qs*.

class shopkit.discounts.advanced.models.discount_models.**CategoryDiscountMixin**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

Mixin defining discounts based on a single category.

classmethod get_valid_discounts(***kwargs*)
Return valid discounts for a specified product

class shopkit.discounts.advanced.models.discount_models.**CouponDiscountMixin**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

Discount based on a specified coupon code.

static generate_coupon_code()
Generate a coupon code of *COUPON_LENGHT* characters consisting of the characters in *COUPON_CHARACTERS*.

Todo

Unittest this function.

classmethod get_valid_discounts(*coupon_code=None*, ***kwargs*)
Return only items for which no coupon code has been set or ones for which the current coupon code matches that of the discounts.

class shopkit.discounts.advanced.models.discount_models.**DateRangeDiscountMixin**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

Mixin for discount which are only valid within a given date range.

classmethod get_valid_discounts(***kwargs*)
Return valid discounts for a specified date, taking the current date if no date is specified. When no start or end date are specified, a discount defaults to be valid.

Todo

Test this code.

class shopkit.discounts.advanced.models.discount_models.**DiscountBase**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

Base class for discounts.

classmethod get_all_discounts()
Get all discounts, whether valid or not.

get_discount(***kwargs*)
Get the total amount of discount produced by this *Discount*. This method should be implemented by subclasses of *:class:DiscountBase*.

classmethod get_valid_discounts(***kwargs*)
Get all valid discount objects for a given *kwargs*. By default, all discounts are invalid.

is_valid(***kwargs*)
Check to see whether an individual discount is valid under the given circumstances.

class shopkit.discounts.advanced.models.discount_models.**ItemDiscountAmountMixin**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

Mixin for absolute amount discounts, valid only for the particular items in this order.

get_discount(***kwargs*)
Get the total amount of discount for the current item.

classmethod get_valid_discounts(***kwargs*)
We want to be able to discriminate between discounts valid for the whole order and those valid for order items.

Parameters **item_discounts** – When *True*, only items for which *item_amount* has been specified are valid. When *False*, only items which have no *item_amount* specified are let through.

class shopkit.discounts.advanced.models.discount_models.**ItemDiscountPercentageMixin**(**args*, ***kwarg*

Bases: django.db.models.base.Model

Mixin for percentual discounts, valid only for the particular items in this order.

get_discount(***kwargs*)
Get the total amount of discount for the current item.

classmethod get_valid_discounts(***kwargs*)
We want to be able to discriminate between discounts valid for the whole order and those valid for order items.

Parameters **item_discounts** – When *True*, only items for which *item_amount* has been specified are valid. When *False*, only items which have no *item_amount* specified are let through.

class shopkit.discounts.advanced.models.discount_models.**LimitedUseDiscountMixin**(**args*, ***kwargs*)
Bases: *shopkit.discounts.advanced.models.discount_models.AccountedUseDiscountMixin*

Mixin class for discounts which can only be used a limited number of times.

get_uses_left()
Return the amount of uses left.

classmethod get_valid_discounts(***kwargs*)
Return currently valid discounts: ones for which either no use limit has been set or for which the amount of uses lies under the limit.

class shopkit.discounts.advanced.models.discount_models.**ManyCategoryDiscountMixin**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Mixin defining discounts based on a collection of categories.

Todo

Test this! There are likely to be bugs...

---

classmethod get_valid_discounts(***kwargs*)
Return valid discounts for a specified product

class shopkit.discounts.advanced.models.discount_models.**ManyProductDiscountMixin**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Mixin defining discounts based on products.

classmethod get_valid_discounts(***kwargs*)
Return valid discounts for a specified product

class shopkit.discounts.advanced.models.discount_models.**OrderDiscountAmountMixin**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Mixin for absolute amount discounts which act on the total price for an order.

get_discount(***kwargs*)
Get the total amount of discount for the current item.

classmethod get_valid_discounts(***kwargs*)
We want to be able to discriminate between discounts valid for the whole order and those valid for order items.

Parameters **order_discounts** – When *True*, only items for which *order_amount* has been specified are valid. When *False*, only items which have no *order_amount* specified are let through.

class shopkit.discounts.advanced.models.discount_models.**OrderDiscountPercentageMixin**(**args*, ***kwar*
Bases: django.db.models.base.Model

Mixin for discounts which apply as a percentage from the total order amount.

get_discount(***kwargs*)
Get the total amount of discount for the current item.

classmethod get_valid_discounts(***kwargs*)
We want to be able to discriminate between discounts valid for the whole order and those valid for order items.

Parameters **order_discounts** – When *True*, only items for which *order_amount* has been specified are valid. When *False*, only items which have no *order_amount* specified are let through.

class shopkit.discounts.advanced.models.discount_models.**ProductDiscountMixin**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Mixin defining a discount valid for a single product.

Todo

Test this! There are likely to be bugs...

---

classmethod get_valid_discounts(***kwargs*)
Return valid discounts for a specified product

Model base and mixin classes for carts and orders with calculated discounts.

Todo

Provide a listing/overview of the types of DiscountMixin's available, how they should be used and... whether they have been tested or not.

---

class shopkit.discounts.advanced.models.order_models.**AccountedDiscountedItemMixin**
Bases: object

Model mixin class for orders for which the use is automatically accounted upon confirmation.

confirm()
Register discount usage.

class shopkit.discounts.advanced.models.order_models.**CalculatedDiscountMixin**
Bases: object

Base class for items for which the discount is calculated using a *Discount* model.

get_valid_discounts(***kwargs*)
Return valid discounts for the given arguments.

class shopkit.discounts.advanced.models.order_models.**CalculatedItemDiscountMixin**
Bases: *shopkit.discounts.advanced.models.order_models.CalculatedDiscountMixin*

Mixin class for discounted objects for which an item discount can be calculated by calling *get_order_discount* and valid discounts can be obtained by calling *get_valid_discounts*.

get_item_discount(***kwargs*)
Get the total discount for this OrderItem.

get_piece_discount(***kwargs*)
Get the total discount per piece for this OrderItem.

get_valid_discounts(***kwargs*)
Return valid discounts for the current order.

class shopkit.discounts.advanced.models.order_models.**CalculatedOrderDiscountMixin**
Bases: *shopkit.discounts.advanced.models.order_models.CalculatedDiscountMixin*

Mixin class for discounted objects for which an order discount can be calculated by calling *get_order_discount* and valid discounts can be obtained by calling *get_valid_discounts*.

get_order_discount(***kwargs*)
Get the discount specific for this *Order*.

get_valid_discounts(***kwargs*)
Return valid discounts for the current order.

class shopkit.discounts.advanced.models.order_models.**DiscountCouponItemMixin**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

Model mixin class for order or cart items for which discounts are calculated based on a coupon code.

get_valid_discounts(***kwargs*)
Return valid discounts for the current item.

class shopkit.discounts.advanced.models.order_models.**DiscountCouponMixin**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

Model mixin class for orders or cart for which discounts are calculated based on a given coupon code.

get_valid_discounts(***kwargs*)
Return valid discounts for the current order.

class shopkit.discounts.advanced.models.order_models.**DiscountedCartItemMixin**(**args*, ***kwargs*)

Bases: *shopkit.discounts.advanced.models.order_models.CalculatedItemDiscountMixin*, *shopkit.discounts.basemodels.DiscountedCartItemBase*

Mixin class for *Cart* objects which have their discount calculated.

class shopkit.discounts.advanced.models.order_models.**DiscountedCartMixin**(**args*, ***kwargs*)

Bases: *shopkit.discounts.advanced.models.order_models.CalculatedOrderDiscountMixin*, *shopkit.discounts.basemodels.DiscountedCartBase*

Mixin class for *Cart* objects which have their discount calculated.

class shopkit.discounts.advanced.models.order_models.**DiscountedOrderItemMixin**(**args*, ***kwargs*)

Bases: *shopkit.discounts.advanced.models.order_models.CalculatedItemDiscountMixin*, *shopkit.discounts.advanced.models.order_models.PersistentDiscountedItemBase*, *shopkit.discounts.basemodels.DiscountedOrderItemBase*

Mixin class for *OrderItem* objects which have their discount calculated.

class shopkit.discounts.advanced.models.order_models.**DiscountedOrderMixin**(**args*, ***kwargs*)

Bases: *shopkit.discounts.advanced.models.order_models.PersistentDiscountedItemBase*, *shopkit.discounts.basemodels.DiscountedOrderBase*, *shopkit.discounts.advanced.models.c*

Mixin class for *Order* objects which have their discount calculated.

class shopkit.discounts.advanced.models.order_models.**PersistentDiscountedItemBase**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

Mixin class for *Order*'s and *OrderItem*'s for which calculated discounts are persistently stored in a *discounts* property upon calling the *update_discount* method.

update_discount()
Call *update_discount* on the superclass to calculate the amount of discount, then store valid *Discount* objects for this order item.

5.4.11 Stock

shopkit.stock Support for stock management for items in the shop.

Todo

Write more descriptive documentation here about the stock management API's.

Contents:

Models

class shopkit.stock.models.**StockedCartBase**
Bases: object

Base class for shopping carts for which stock is kept.

add_item(*product*, *quantity=1*, ***kwargs*)
Attempt to add item to cart.

This method will raise a NoStockAvailableException when no stock items are available.

class shopkit.stock.models.**StockedCartItemBase**
Bases: object

Base class for cart items for which the stock can be maintained. By default the *is_available* method returns *True*, this method can be overridden in subclassed to provide for more extended functionality.

Todo

Decide whether this bugger belongs into :module:shopkit.core or whether it is just fine at it's place right here.

- Pro: We'll have a generic API for determining the stock state of items.
- Con: It's bad to have too much code in the core, it is better if modules within *django-shopkit* have the least possible knowledge about one another.

---

is_available(*quantity*)
The *is_available* method can be used to determine whether a cart item is eligible to be saved or not.

class shopkit.stock.models.**StockedItemBase**
Bases: object

Generic base class for *CartItem*'s or *OrderItem*'s for which the stock is represented by a stocked item somehow.

get_stocked_item()
Get the StockedItemMixin subclass instance whose *is_available* method should determine whether we are out of stock.

This method should be overridden in order to be able to specify whether the cart item is available or not.

is_available(*quantity*)
Determine whether or not this item is available.

class shopkit.stock.models.**StockedOrderBase**
Bases: object

Mixin base class for *Order*'s with items for which stock is kept.

check_stock()
Check the stock for all items in this order.

class shopkit.stock.models.**StockedOrderItemBase**
Bases: object

Mixin base class for *OrderItem*'s containing items for which stock is kept.

check_stock()
Check whether the stock for the current order item is available. This should be called right before *register_confirmation*.

confirm()
Before registering confirmation, first make sure enough stock is available. This should have already been checked when adding the product to the shopping cart but who knows: somebody might have already bought the product in the meanwhile.

For this to work well, it is important that this *register_confirmation* function is called before that of discounts and other possible accounting functions.

prepare_confirm()
Extend confirmation preparation by checking whether stock is available for this order.

Raises NoStockAvailableException

Exceptions

exception shopkit.stock.exceptions.**NoStockAvailableException**(*item*)
Bases: *shopkit.core.exceptions.ShopKitExceptionBase*

Exception raised by the save method of StockedCartItemMixinBase when no stock is available for the current item.

Simple

shopkit.stock.simple Simple stock management: *StockedItemMixin*'s will have a *stock* property which is a SmallPositiveIntegerField used to provide not an exact stock count, rather then a choice from few options used to determine whether an item is available or not.

Contents:

Settings

Models

class shopkit.stock.simple.models.**StockedCartItemMixin**
Bases: *shopkit.stock.models.StockedItemBase*, *shopkit.stock.models.StockedCartItemBase*

Mixin class for *CartItem*'s containing items for which stock is kept.

class shopkit.stock.simple.models.**StockedCartMixin**
Bases: *shopkit.stock.models.StockedCartBase*

Mixin class for *Cart*'s containing items for which stock is kept.

class shopkit.stock.simple.models.**StockedItemMixin**(**args*, ***kwargs*)
Bases: django.db.models.base.Model, *shopkit.stock.models.StockedItemBase*

Item with a simple stock selection mechanism: the possible options for the available *stock* field signify certain stock states, some of which correspond to an item being orderable.

This could be associated with a *Product*, a *Variation* or some other property that pertains to the specific state of *CartItemBase* subclasses.

is_available(*quantity=None*)
Method used to determine whether or not the current item is in an orderable state.

class shopkit.stock.simple.models.**StockedOrderItemMixin**
Bases: *shopkit.stock.models.StockedItemBase*, *shopkit.stock.models.StockedOrderItemBase*

Mixin class for *OrderItem*'s containing items for which stock is kept.

class shopkit.stock.simple.models.**StockedOrderMixin**
Bases: *shopkit.stock.models.StockedOrderBase*

Mixin class for *Order*'s containing items for which stock is kept.

Advanced

shopkit.stock.advanced

Todo

Write documentation here.

Models

shopkit.stock.advanced.models

class shopkit.stock.advanced.models.**StockedCartItemMixin**
Bases: *shopkit.stock.models.StockedItemBase*, *shopkit.stock.models.StockedCartItemBase*

Mixin class for *CartItem*'s containing items for which stock is kept.

class shopkit.stock.advanced.models.**StockedCartMixin**
Bases: *shopkit.stock.models.StockedCartBase*

Mixin class for *Cart*'s containing items for which stock is kept.

class shopkit.stock.advanced.models.**StockedItemMixin**(**args*, ***kwargs*)
Bases: django.db.models.base.Model, *shopkit.stock.models.StockedItemBase*

Item for which stock is kept in an integer *stock* field.

is_available(*quantity*)
Method used to determine whether or not the current item is in an orderable state.

class shopkit.stock.advanced.models.**StockedOrderItemMixin**
Bases: *shopkit.stock.models.StockedItemBase*, *shopkit.stock.models.StockedOrderItemBase*

Mixin class for *OrderItem*'s containing items for which stock is kept.

confirm()
Register lowering of the current item's stock.

class shopkit.stock.advanced.models.**StockedOrderMixin**
Bases: *shopkit.stock.models.StockedOrderBase*

Mixin class for *Order*'s containing items for which stock is kept.

Settings

shopkit.stock.advanced.settings

Tests

shopkit.stock.advanced.tests

5.4.12 Related products

shopkit.related Simple extension to allow for products to products to relate to one another.

Contents:

Models

shopkit.related.models

class shopkit.related.models.**RelatedProductsMixin**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Mixin to allow for relating products to one another.

Settings

shopkit.related.settings

5.4.13 Brand management

shopkit.brands Simple extension to allow for products to have brands.

Contents:

Models

shopkit.brands.models

class shopkit.brands.models.**BrandBase**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract base class for brands. Use like this:

```
class Brand(BrandBase, OrderedItemBase, NamedItemBase):
    pass
```

Todo

Add methods for listing all available products (using the *in_shop* manager) for a given brand.

class shopkit.brands.models.**BrandedProductMixin**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Mixin for product classes with a relation to brand.

Settings

shopkit.brands.settings

5.4.14 Featured products

shopkit.featured Base classes for featured products and ordered featured products in the webshop.

Contents:

Models

shopkit.featured.models

class shopkit.featured.models.**FeaturedProductMixin**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Mixin for products which have a boolean featured property and an *is_featured* manager, filtering the items from the *in_shop* manager so that only featured items are returned.

Todo

Write the *is_featured* manager - and test it.

---

class shopkit.featured.models.**OrderedFeaturedProductMixin**(**args*, ***kwargs*)
Bases: *shopkit.featured.models.FeaturedProductMixin*

Mixin for ordered featured products.

Todo

Make sure the *is_featured* manager for this base model uses the *featured_order* attribute.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Last update: June 09, 2015

b

shopkit.brands, 47
shopkit.brands.models, 47
shopkit.brands.settings, 47

c

shopkit.category, 30
shopkit.category.advanced, 32
shopkit.category.advanced.models, 33
shopkit.category.advanced.tests, 33
shopkit.category.advanced.views, 33
shopkit.category.basemodels, 31
shopkit.category.settings, 32
shopkit.category.simple, 32
shopkit.category.simple.models, 32
shopkit.category.simple.tests, 32
shopkit.category.simple.views, 32
shopkit.category.tests, 32
shopkit.configurable, 34
shopkit.configurable.advanced, 35
shopkit.configurable.advanced.models, 35
shopkit.configurable.advanced.settings, 35
shopkit.configurable.advanced.views, 35
shopkit.configurable.simple, 35
shopkit.configurable.simple.models, 35
shopkit.configurable.simple.settings, 35
shopkit.configurable.simple.views, 35
shopkit.core, 13
shopkit.core.admin, 13
shopkit.core.basemodels, 14
shopkit.core.context_processors, 22
shopkit.core.exceptions, 23
shopkit.core.forms, 20
shopkit.core.listeners, 23
shopkit.core.managers, 19
shopkit.core.models, 16
shopkit.core.settings, 21
shopkit.core.signals, 23
shopkit.core.tests, 22
shopkit.core.utils, 21
shopkit.core.utils.admin, 21
shopkit.core.utils.fields, 21
shopkit.core.utils.listeners, 22
shopkit.core.views, 19
shopkit.currency, 34
shopkit.currency.advanced, 34
shopkit.currency.advanced.models, 34
shopkit.currency.advanced.settings, 34
shopkit.currency.advanced.views, 34
shopkit.currency.simple, 34
shopkit.currency.simple.fields, 34
shopkit.currency.simple.settings, 34
shopkit.currency.simple.utils, 34

d

shopkit.discounts, 37
shopkit.discounts.advanced, 39
shopkit.discounts.advanced.admin, 39
shopkit.discounts.advanced.models, 39
shopkit.discounts.advanced.models.discount_models 39
shopkit.discounts.advanced.models.order_models, 42
shopkit.discounts.basemodels, 37
shopkit.discounts.settings, 39

f

shopkit.featured, 47
shopkit.featured.models, 47

i

shopkit.images, 36
shopkit.images.admin, 36
shopkit.images.models, 37
shopkit.images.settings, 37

p

shopkit.price, 24
shopkit.price.advanced, 25
shopkit.price.advanced.admin, 26
shopkit.price.advanced.forms, 26
shopkit.price.advanced.models, 25
shopkit.price.advanced.settings, 26
shopkit.price.advanced.tests, 26
shopkit.price.models, 25
shopkit.price.simple, 25

shopkit.price.simple.models, 25

r

shopkit.related, 46
shopkit.related.models, 47
shopkit.related.settings, 47

S

shopkit.shipping, 27
shopkit.shipping.advanced, 27
shopkit.shipping.advanced.admin, 29
shopkit.shipping.advanced.models, 27
shopkit.shipping.advanced.models.order_models, 28
shopkit.shipping.advanced.models.shipping_models, 27
shopkit.shipping.advanced.settings, 29
shopkit.shipping.basemodels, 29
shopkit.shipping.models, 29
shopkit.shipping.settings, 29
shopkit.stock, 43
shopkit.stock.advanced, 46
shopkit.stock.advanced.models, 46
shopkit.stock.advanced.settings, 46
shopkit.stock.advanced.tests, 46
shopkit.stock.exceptions, 45
shopkit.stock.models, 44
shopkit.stock.simple, 45
shopkit.stock.simple.models, 45
shopkit.stock.simple.settings, 45

V

shopkit.variations, 35
shopkit.variations.admin, 35
shopkit.variations.models, 35
shopkit.variations.settings, 36
shopkit.vat, 33
shopkit.vat.advanced, 33
shopkit.vat.advanced.models, 34
shopkit.vat.advanced.settings, 34
shopkit.vat.advanced.views, 34
shopkit.vat.simple, 33
shopkit.vat.simple.models, 33
shopkit.vat.simple.settings, 33
shopkit.vat.simple.views, 33

A

AbstractCustomerBase (class in shopkit.core.basemodels), 14
AbstractPricedItemBase (class in shopkit.core.basemodels), 14
AccountedDiscountedItemMixin (class in shopkit.discounts.advanced.models.order_models), 42
AccountedUseDiscountMixin (class in shopkit.discounts.advanced.models.discount_models), 39
ActiveItemBase (class in shopkit.core.basemodels), 14
ActiveItemInShopBase (class in shopkit.core.basemodels), 14
ActiveItemManager (class in shopkit.core.managers), 19
add_item() (shopkit.core.models.CartBase method), 16
add_item() (shopkit.stock.models.StockedCartBase method), 44
AddressBase (class in shopkit.core.models), 16
AdvancedPriceTestMixin (class in shopkit.price.advanced.tests), 26
AlreadyConfirmedException, 23
as_listener() (shopkit.core.utils.listeners.Listener class method), 22
AutomaticShippingMixin (class in shopkit.shipping.advanced.models.order_models), 28

B

BrandBase (class in shopkit.brands.models), 47
BrandedProductMixin (class in shopkit.brands.models), 47

C

CalculatedDiscountMixin (class in shopkit.discounts.advanced.models.order_models), 42
CalculatedItemDiscountMixin (class in shopkit.discounts.advanced.models.order_models), 42
CalculatedOrderDiscountMixin (class in shopkit.discounts.advanced.models.order_models), 42
cart() (in module shopkit.core.context_processors), 22
CartAddBase (class in shopkit.core.views), 19
CartAddFormMixin (class in shopkit.core.views), 20
CartBase (class in shopkit.core.models), 16
CartItemAddForm (class in shopkit.core.forms), 20
CartItemBase (class in shopkit.core.models), 16
CategoriesMixin (class in shopkit.category.simple.views), 32
CategorizedItemBase (class in shopkit.category.advanced.models), 33
CategorizedItemBase (class in shopkit.category.simple.models), 32
CategoryBase (class in shopkit.category.basemodels), 31
CategoryDiscountMixin (class in shopkit.discounts.advanced.models.discount_models), 39
CategoryTestMixin (class in shopkit.category.advanced.tests), 33
CategoryTestMixin (class in shopkit.category.simple.tests), 32
CategoryTestMixinBase (class in shopkit.category.tests), 32
CheapestShippingMixin (class in shopkit.shipping.advanced.models.order_models), 28
check_stock() (shopkit.stock.models.StockedOrderBase method), 44
check_stock() (shopkit.stock.models.StockedOrderItemBase method), 44
clean() (shopkit.core.basemodels.OrderedItemBase method), 15
clean() (shopkit.price.advanced.forms.PriceInlineFormSet method), 26
confirm() (shopkit.core.basemodels.NumberedOrderBase method), 14
confirm() (shopkit.core.models.OrderBase method), 17
confirm() (shopkit.core.models.OrderItemBase method), 18
confirm() (shopkit.discounts.advanced.models.order_models.AccountedD method), 42
confirm() (shopkit.stock.advanced.models.StockedOrderItemMixin method), 46
confirm() (shopkit.stock.models.StockedOrderItemBase method), 45
CoreTestMixin (class in shopkit.core.tests), 22

CouponDiscountMixin (class in shopkit.discounts.advanced.models.discount_models), 39
create_message() (shopkit.core.listeners.EmailingListener method), 23
CustomerAddressBase (class in shopkit.core.models), 17
CustomerCartBase (class in shopkit.core.models), 17
CustomerOrderBase (class in shopkit.core.models), 17
CustomerPaymentBase (class in shopkit.core.models), 17

D

DatedItemBase (class in shopkit.core.basemodels), 14
DateRangeDiscountMixin (class in shopkit.discounts.advanced.models.discount_models), 40
DateRangedPriceMixin (class in shopkit.price.advanced.models), 25
default_image() (shopkit.images.admin.ImagesProductAdminMixin method), 36
DiscountBase (class in shopkit.discounts.advanced.models.discount_models), 40
DiscountCouponItemMixin (class in shopkit.discounts.advanced.models.order_models), 43
DiscountCouponMixin (class in shopkit.discounts.advanced.models.order_models), 43
DiscountedCartBase (class in shopkit.discounts.basemodels), 37
DiscountedCartItemBase (class in shopkit.discounts.basemodels), 38
DiscountedCartItemMixin (class in shopkit.discounts.advanced.models.order_models), 43
DiscountedCartMixin (class in shopkit.discounts.advanced.models.order_models), 43
DiscountedItemBase (class in shopkit.discounts.basemodels), 38
DiscountedOrderBase (class in shopkit.discounts.basemodels), 38
DiscountedOrderItemBase (class in shopkit.discounts.basemodels), 38
DiscountedOrderItemMixin (class in shopkit.discounts.advanced.models.order_models), 43
DiscountedOrderMixin (class in shopkit.discounts.advanced.models.order_models), 43
dispatch() (shopkit.core.listeners.StateChangeListener method), 24

E

EmailingListener (class in shopkit.core.listeners), 23

F

FeaturedProductMixin (class in shopkit.featured.models), 47
form_valid() (shopkit.core.views.CartAddBase method), 20
format_price() (in module shopkit.currency.simple.utils), 34
formset (shopkit.price.advanced.admin.PriceInline attribute), 26
from_cart() (shopkit.core.models.CustomerOrderBase class method), 17
from_cart() (shopkit.core.models.OrderBase class method), 18
from_cartitem() (shopkit.core.models.OrderItemBase class method), 18
from_cartitem() (shopkit.variations.models.VariationOrderItemMixin class method), 36
from_request() (shopkit.core.models.CartBase class method), 16
from_request() (shopkit.core.models.CustomerCartBase class method), 17

G

generate_coupon_code() (shopkit.discounts.advanced.models.discount_models.CouponDiscou static method), 39
generate_invoice_number() (shopkit.core.basemodels.NumberedOrderBase method), 14
generate_order_number() (shopkit.core.basemodels.NumberedOrderBase method), 14
get_all_discounts() (shopkit.discounts.advanced.models.discount_models.DiscountBase class method), 40
get_all_orders() (shopkit.core.basemodels.AbstractCustomerBase method), 14
get_body_template_names() (shopkit.core.listeners.EmailingListener method), 23
get_cart_form_class() (shopkit.core.views.CartAddFormMixin method), 20
get_categories() (shopkit.category.basemodels.CategoryBase class method), 31
get_cheapest() (shopkit.price.advanced.models.PriceBase class method), 25
get_cheapest() (shopkit.shipping.advanced.models.shipping_models.ItemShippingM class method), 27

get_cheapest() (shopkit.shipping.advanced.models.shipping_models.OrderShippingMethodMixin class method), 27
get_confirmed_orders() (shopkit.core.basemodels.AbstractCustomerBase method), 14
get_context_data() (shopkit.category.simple.views.CategoriesMixin method), 32
get_context_data() (shopkit.core.listeners.EmailingListener method), 23
get_context_data() (shopkit.core.views.CartAddFormMixin method), 20
get_cost() (shopkit.shipping.advanced.models.shipping_models.ShippingMethodBase method), 28
get_default_image() (shopkit.images.models.ImagesProductMixin method), 37
get_default_variation() (shopkit.variations.models.OrderedProductVariationBase class method), 35
get_default_variation() (shopkit.variations.models.ProductVariationBase class method), 36
get_discount() (shopkit.discounts.advanced.models.discount_models.DiscountBase method), 40
get_discount() (shopkit.discounts.advanced.models.discount_models.ItemDiscountAdvanced method), 40
get_discount() (shopkit.discounts.advanced.models.discount_models.ItemDiscountBase method), 40
get_discount() (shopkit.discounts.advanced.models.discount_models.OrderDiscountAdvanced method), 41
get_discount() (shopkit.discounts.advanced.models.discount_models.OrderDiscountBase method), 41
get_discount() (shopkit.discounts.basemodels.DiscountedItemBase method), 38
get_filters() (shopkit.core.utils.admin.LimitedAdminInlineMixin method), 21
get_form_class() (shopkit.core.views.CartAddBase method), 20
get_formset() (shopkit.core.utils.admin.LimitedAdminInlineMixin method), 22
get_item() (shopkit.core.models.CartBase method), 16
get_item_discount() (shopkit.discounts.advanced.models.order_models.CalculatedItemDiscountMixin method), 42
get_item_discount() (shopkit.discounts.basemodels.DiscountedCartItemBase method), 38
get_item_discount() (shopkit.discounts.basemodels.DiscountedOrderItemBase method), 39
get_items() (shopkit.core.models.CartBase method), 16
get_items() (shopkit.core.models.OrderBase method), 18
get_language() (shopkit.core.listeners.TranslatedEmailingListener method), 24
get_latest() (shopkit.core.models.OrderStateChangeBase class method), 19
get_latest_order() (shopkit.core.basemodels.AbstractCustomerBase method), 14
get_main_categories() (shopkit.category.basemodels.CategoryBase class method), 31
get_main_categories() (shopkit.category.basemodels.MPTTCategoryBase class method), 31
get_main_categories() (shopkit.category.basemodels.NestedCategoryBase class method), 31
get_model_from_string() (in module shopkit.core.utils), 21
get_next_ordering() (shopkit.core.basemodels.OrderedInlineItemBase static method), 15
get_order_discount() (shopkit.discounts.advanced.models.order_models.CalculatedOrderDiscountMixin method), 42
get_order_discount() (shopkit.discounts.basemodels.DiscountedCartBase method), 37
get_order_discount() (shopkit.discounts.basemodels.DiscountedOrderBase method), 38
get_order_line() (shopkit.core.models.CartBase method), 16
get_order_line() (shopkit.core.models.CartItemBase method), 17
get_order_shipping_costs() (shopkit.shipping.basemodels.ShippedCartBase method), 29
get_order_shipping_costs() (shopkit.shipping.basemodels.ShippedOrderBase method), 30
get_parent() (shopkit.core.models.CartItemBase method), 17
get_parent() (shopkit.core.models.OrderItemBase method), 18
get_parent_list() (shopkit.category.basemodels.NestedCategoryBase method), 31
get_piece_discount() (shopkit.discounts.advanced.models.order_models.CalculatedItemDiscountMixin method), 42

get_piece_discount() (shopkit.discounts.basemodels.DiscountedItemBase method), 38
get_piece_price() (shopkit.core.models.CartItemBase method), 17
get_piece_price() (shopkit.core.models.OrderItemBase method), 18
get_piece_price_with_discount() (shopkit.discounts.basemodels.DiscountedItemBase method), 38
get_piece_price_without_discount() (shopkit.discounts.basemodels.DiscountedItemBase method), 38
get_price() (shopkit.core.basemodels.AbstractPricedItemBase method), 14
get_price() (shopkit.core.models.CartBase method), 16
get_price() (shopkit.core.models.CartItemBase method), 17
get_price() (shopkit.core.models.OrderBase method), 18
get_price() (shopkit.core.models.OrderItemBase method), 19
get_price() (shopkit.discounts.basemodels.DiscountedItemBase method), 38
get_price() (shopkit.price.models.PricedItemBase method), 25
get_price() (shopkit.shipping.basemodels.ShippedItemBase method), 30
get_price() (shopkit.vat.simple.models.VATItemBase method), 33
get_price_with_vat() (shopkit.vat.simple.models.VATItemBase method), 33
get_price_without_discount() (shopkit.discounts.basemodels.DiscountedItemBase method), 38
get_price_without_shipping() (shopkit.shipping.basemodels.ShippedItemBase method), 30
get_price_without_vat() (shopkit.vat.simple.models.VATItemBase method), 33
get_product_choices() (in module shopkit.core.forms), 20
get_products() (shopkit.category.basemodels.CategoryBase method), 31
get_products() (shopkit.category.basemodels.MPTTCategoryBase method), 31
get_products() (shopkit.category.basemodels.NestedCategoryBase method), 31
get_query_set() (shopkit.core.managers.ActiveItemManager method), 19
get_queryset() (shopkit.core.views.InShopViewMixin method), 20
get_recent_shipping() (shopkit.shipping.models.ShippableCustomerMixin method), 29
get_recipients() (shopkit.core.listeners.EmailingListener method), 23
get_related_ordering() (shopkit.core.basemodels.OrderedInlineItemBase method), 15
get_related_ordering() (shopkit.images.models.OrderedProductImageBase method), 37
get_related_ordering() (shopkit.variations.models.OrderedProductVariationBase method), 35
get_sender() (shopkit.core.listeners.EmailingListener method), 23
get_shipping_costs() (shopkit.shipping.basemodels.ShippedItemBase method), 30
get_shipping_costs() (shopkit.shipping.basemodels.ShippedOrderItemBase method), 30
get_shipping_method() (shopkit.shipping.advanced.models.order_models.AutomaticShipping method), 28
get_shipping_method() (shopkit.shipping.advanced.models.order_models.CheapestShipping method), 28
get_stocked_item() (shopkit.stock.models.StockedItemBase method), 44
get_subcategories() (shopkit.category.basemodels.MPTTCategoryBase method), 31
get_subcategories() (shopkit.category.basemodels.NestedCategoryBase method), 31
get_subject_template_names() (shopkit.core.listeners.EmailingListener method), 24
get_success_url() (shopkit.core.views.CartAddBase method), 20
get_total_discount() (shopkit.discounts.basemodels.DiscountedCartBase method), 38
get_total_discount() (shopkit.discounts.basemodels.DiscountedCartItemBase method), 38
get_total_discount() (shopkit.discounts.basemodels.DiscountedItemBase method), 38
get_total_discount() (shopkit.discounts.basemodels.DiscountedOrderBase method), 38
get_total_discount() (shopkit.discounts.basemodels.DiscountedOrderItemBase method), 39

get_total_items() (shopkit.core.models.CartBase method), 16
get_total_items() (shopkit.core.models.OrderBase method), 18
get_total_price() (shopkit.core.models.CartBase method), 16
get_total_price() (shopkit.core.models.CartItemBase method), 17
get_total_price() (shopkit.core.models.OrderBase method), 18
get_total_price() (shopkit.core.models.OrderItemBase method), 19
get_total_shipping_costs() (shopkit.shipping.basemodels.ShippedCartBase method), 29
get_total_shipping_costs() (shopkit.shipping.basemodels.ShippedItemBase method), 30
get_total_shipping_costs() (shopkit.shipping.basemodels.ShippedOrderBase method), 30
get_uses_left() (shopkit.discounts.advanced.models.discount_models.LimitedUseShippingMixin method), 41
get_valid_discounts() (shopkit.discounts.advanced.models.discount_models.CategoryDiscountMixin class method), 39
get_valid_discounts() (shopkit.discounts.advanced.models.discount_models.CouponDiscountMixin class method), 40
get_valid_discounts() (shopkit.discounts.advanced.models.discount_models.DateRangedDiscountMixin class method), 40
get_valid_discounts() (shopkit.discounts.advanced.models.discount_models.DiscountBase class method), 40
get_valid_discounts() (shopkit.discounts.advanced.models.discount_models.ItemDiscountAmountMixin class method), 40
get_valid_discounts() (shopkit.discounts.advanced.models.discount_models.ItemDiscountPercentageMixin class method), 41
get_valid_discounts() (shopkit.discounts.advanced.models.discount_models.LimitedUseDiscountMixin class method), 41
get_valid_discounts() (shopkit.discounts.advanced.models.discount_models.ManyCategoryDiscountMixin class method), 41
get_valid_discounts() (shopkit.discounts.advanced.models.discount_models.ManyProductDiscountMixin class method), 41
get_valid_discounts() (shopkit.discounts.advanced.models.discount_models.OrderDiscountAmountMixin class method), 41
get_valid_discounts() (shopkit.discounts.advanced.models.discount_models.OrderDiscountPercentageMixin class method), 42
get_valid_discounts() (shopkit.discounts.advanced.models.discount_models.ProductDiscou class method), 42
get_valid_discounts() (shopkit.discounts.advanced.models.order_models.CalculatedDiscou method), 42
get_valid_discounts() (shopkit.discounts.advanced.models.order_models.CalculatedItemDi method), 42
get_valid_discounts() (shopkit.discounts.advanced.models.order_models.CalculatedOrderD method), 43
get_valid_discounts() (shopkit.discounts.advanced.models.order_models.DiscountCouponI method), 43
get_valid_discounts() (shopkit.discounts.advanced.models.order_models.DiscountCouponM method), 43
get_valid_methods() (shopkit.shipping.advanced.models.shipping_models.ItemShippingM class method), 27
get_valid_methods() (shopkit.shipping.advanced.models.shipping_models.MinimumItemA class method), 27
get_valid_methods() (shopkit.shipping.advanced.models.shipping_models.MinimumOrder class method), 27
get_valid_methods() (shopkit.shipping.advanced.models.shipping_models.OrderShippingM class method), 27
get_valid_methods() (shopkit.shipping.advanced.models.shipping_models.ShippingMetho class method), 28
get_valid_prices() (shopkit.price.advanced.models.DateRangedPriceMixin class method), 25
get_valid_prices() (shopkit.price.models.PriceBase class method), 25
get_valid_prices() (shopkit.price.models.ProductPriceMixin class method), 26
get_valid_prices() (shopkit.price.models.QuantifiedPriceMixin class method), 26
get_vat() (shopkit.vat.simple.models.VATItemBase method), 33

H

handler() (shopkit.core.listeners.EmailingListener method), 24
handler() (shopkit.core.listeners.StateChangeListener method), 24
handler() (shopkit.core.listeners.StateChangeLogger method), 24
handler() (shopkit.core.listeners.TranslatedEmailingListener method), 24

I

ImagesProductAdminMixin (class in shopkit.images.admin), 36
ImagesProductMixin (class in shopkit.images.models), 37
InShopViewMixin (class in shopkit.core.views), 20
is_available() (shopkit.stock.advanced.models.StockedItemMixin method), 46
is_available() (shopkit.stock.models.StockedCartItemBase method), 44
is_available() (shopkit.stock.models.StockedItemBase method), 44
is_available() (shopkit.stock.simple.models.StockedItemMixin method), 45
is_valid() (shopkit.discounts.advanced.models.discount_models.DiscountBase method), 40
is_valid() (shopkit.shipping.advanced.models.shipping_models.ShippingMethodBase method), 28
ItemDiscountAmountMixin (class in shopkit.discounts.advanced.models.discount_models), 40
ItemDiscountPercentageMixin (class in shopkit.discounts.advanced.models.discount_models), 40
ItemShippingMethodMixin (class in shopkit.shipping.advanced.models.shipping_models), 27

L

limit_inline_choices() (shopkit.core.utils.admin.LimitedAdminInlineMixin static method), 22
LimitedAdminInlineMixin (class in shopkit.core.utils.admin), 21
LimitedUseDiscountMixin (class in shopkit.discounts.advanced.models.discount_models), 41
Listener (class in shopkit.core.utils.listeners), 22

M

make_category() (shopkit.category.tests.CategoryTestMixinBase method), 32
make_product() (shopkit.core.tests.CoreTestMixin method), 22
ManyCategoryDiscountMixin (class in shopkit.discounts.advanced.models.discount_models), 41
ManyProductDiscountMixin (class in shopkit.discounts.advanced.models.discount_models), 41
MinimumItemAmountShippingMixin (class in shopkit.shipping.advanced.models.shipping_models), 27
MinimumOrderAmountShippingMixin (class in shopkit.shipping.advanced.models.shipping_models), 27
MinMaxDecimalField (class in shopkit.core.utils.fields), 21
MPTTCategoryBase (class in shopkit.category.basemodels), 31

N

NamedItemBase (class in shopkit.core.basemodels), 14
NestedCategoryBase (class in shopkit.category.basemodels), 31
NoStockAvailableException, 45
NumberedOrderBase (class in shopkit.core.basemodels), 14

O

OrderBase (class in shopkit.core.models), 17
OrderDiscountAmountMixin (class in shopkit.discounts.advanced.models.discount_models), 41
OrderDiscountPercentageMixin (class in shopkit.discounts.advanced.models.discount_models), 41
OrderedFeaturedProductMixin (class in shopkit.featured.models), 48
OrderedInlineItemBase (class in shopkit.core.basemodels), 15
OrderedItemBase (class in shopkit.core.basemodels), 15
OrderedProductImageBase (class in shopkit.images.models), 37
OrderedProductVariationBase (class in shopkit.variations.models), 35
OrderItemBase (class in shopkit.core.models), 18
OrderShippingMethodMixin (class in shopkit.shipping.advanced.models.shipping_models), 27
OrderStateChangeBase (class in shopkit.core.models), 19

P

PaymentBase (class in shopkit.core.models), 19
PercentageField (class in shopkit.core.utils.fields), 21
PersistentDiscountedItemBase (class in shopkit.discounts.advanced.models.order_models), 43
PersistentShippedItemBase (class in shopkit.shipping.advanced.models.order_models), 28
prepare_confirm() (shopkit.core.models.OrderBase method), 18
prepare_confirm() (shopkit.stock.models.StockedOrderItemBase method), 45
PriceBase (class in shopkit.price.advanced.models), 25
PricedItemAdminMixin (class in shopkit.core.admin), 13
PricedItemBase (class in shopkit.price.models), 25
PriceField (class in shopkit.currency.simple.fields), 34
PriceInline (class in shopkit.price.advanced.admin), 26

PriceInlineFormSet (class in shopkit.price.advanced.forms), 26
ProductBase (class in shopkit.core.models), 19
ProductDiscountMixin (class in shopkit.discounts.advanced.models.discount_models), 42
ProductImageBase (class in shopkit.images.models), 37
ProductImageInline (class in shopkit.images.admin), 37
ProductPriceMixin (class in shopkit.price.advanced.models), 25
ProductVariationBase (class in shopkit.variations.models), 35
ProductVariationInline (class in shopkit.variations.admin), 35
PublishDateItemBase (class in shopkit.core.basemodels), 15

Q

QuantifiedPriceMixin (class in shopkit.price.advanced.models), 26
QuantizedItemBase (class in shopkit.core.basemodels), 15

R

register_use() (shopkit.discounts.advanced.models.discount_models.AccountedUseDiscountMixin class method), 39
RelatedProductsMixin (class in shopkit.related.models), 47
remove_item() (shopkit.core.models.CartBase method), 16

S

save() (shopkit.core.basemodels.NumberedOrderBase method), 15
save() (shopkit.core.basemodels.OrderedInlineItemBase method), 15
save() (shopkit.core.models.CustomerAddressBase method), 17
save() (shopkit.core.models.OrderBase method), 18
setUp() (shopkit.category.tests.CategoryTestMixinBase method), 32
setUp() (shopkit.core.tests.CoreTestMixin method), 22
setUp() (shopkit.price.advanced.tests.AdvancedPriceTestMixin method), 26
ShippableCustomerMixin (class in shopkit.shipping.models), 29
ShippedCartBase (class in shopkit.shipping.basemodels), 29
ShippedCartItemBase (class in shopkit.shipping.basemodels), 29
ShippedCartItemMixin (class in shopkit.shipping.advanced.models.order_models), 28
ShippedCartMixin (class in shopkit.shipping.advanced.models.order_models), 28
ShippedItemBase (class in shopkit.shipping.basemodels), 29
ShippedOrderBase (class in shopkit.shipping.basemodels), 30
ShippedOrderItemBase (class in shopkit.shipping.basemodels), 30
ShippedOrderItemMixin (class in shopkit.shipping.advanced.models.order_models), 29
ShippedOrderMixin (class in shopkit.shipping.advanced.models.order_models), 29
ShippingMethodBase (class in shopkit.shipping.advanced.models.shipping_models), 28
shopkit.brands (module), 47
shopkit.brands.models (module), 47
shopkit.brands.settings (module), 47
shopkit.category (module), 30
shopkit.category.advanced (module), 32
shopkit.category.advanced.models (module), 33
shopkit.category.advanced.tests (module), 33
shopkit.category.advanced.views (module), 33
shopkit.category.basemodels (module), 31
shopkit.category.settings (module), 32
shopkit.category.simple (module), 32
shopkit.category.simple.models (module), 32
shopkit.category.simple.tests (module), 32
shopkit.category.simple.views (module), 32
shopkit.category.tests (module), 32
shopkit.configurable (module), 34
shopkit.configurable.advanced (module), 35
shopkit.configurable.advanced.models (module), 35
shopkit.configurable.advanced.settings (module), 35
shopkit.configurable.advanced.views (module), 35
shopkit.configurable.simple (module), 35
shopkit.configurable.simple.models (module), 35
shopkit.configurable.simple.settings (module), 35
shopkit.configurable.simple.views (module), 35
shopkit.core (module), 13
shopkit.core.admin (module), 13
shopkit.core.basemodels (module), 14
shopkit.core.context_processors (module), 22
shopkit.core.exceptions (module), 23
shopkit.core.forms (module), 20
shopkit.core.listeners (module), 23
shopkit.core.managers (module), 19
shopkit.core.models (module), 16
shopkit.core.settings (module), 21
shopkit.core.signals (module), 23
shopkit.core.tests (module), 22
shopkit.core.utils (module), 21
shopkit.core.utils.admin (module), 21
shopkit.core.utils.fields (module), 21
shopkit.core.utils.listeners (module), 22
shopkit.core.views (module), 19
shopkit.currency (module), 34
shopkit.currency.advanced (module), 34
shopkit.currency.advanced.models (module), 34
shopkit.currency.advanced.settings (module), 34

shopkit.currency.advanced.views (module), 34
shopkit.currency.simple (module), 34
shopkit.currency.simple.fields (module), 34
shopkit.currency.simple.settings (module), 34
shopkit.currency.simple.utils (module), 34
shopkit.discounts (module), 37
shopkit.discounts.advanced (module), 39
shopkit.discounts.advanced.admin (module), 39
shopkit.discounts.advanced.models (module), 39
shopkit.discounts.advanced.models.discount_models (module), 39
shopkit.discounts.advanced.models.order_models (module), 42
shopkit.discounts.basemodels (module), 37
shopkit.discounts.settings (module), 39
shopkit.featured (module), 47
shopkit.featured.models (module), 47
shopkit.images (module), 36
shopkit.images.admin (module), 36
shopkit.images.models (module), 37
shopkit.images.settings (module), 37
shopkit.price (module), 24
shopkit.price.advanced (module), 25
shopkit.price.advanced.admin (module), 26
shopkit.price.advanced.forms (module), 26
shopkit.price.advanced.models (module), 25
shopkit.price.advanced.settings (module), 26
shopkit.price.advanced.tests (module), 26
shopkit.price.models (module), 25
shopkit.price.simple (module), 25
shopkit.price.simple.models (module), 25
shopkit.related (module), 46
shopkit.related.models (module), 47
shopkit.related.settings (module), 47
shopkit.shipping (module), 27
shopkit.shipping.advanced (module), 27
shopkit.shipping.advanced.admin (module), 29
shopkit.shipping.advanced.models (module), 27
shopkit.shipping.advanced.models.order_models (module), 28
shopkit.shipping.advanced.models.shipping_models (module), 27
shopkit.shipping.advanced.settings (module), 29
shopkit.shipping.basemodels (module), 29
shopkit.shipping.models (module), 29
shopkit.shipping.settings (module), 29
shopkit.stock (module), 43
shopkit.stock.advanced (module), 46
shopkit.stock.advanced.models (module), 46
shopkit.stock.advanced.settings (module), 46
shopkit.stock.advanced.tests (module), 46
shopkit.stock.exceptions (module), 45
shopkit.stock.models (module), 44
shopkit.stock.simple (module), 45
shopkit.stock.simple.models (module), 45
shopkit.stock.simple.settings (module), 45
shopkit.variations (module), 35
shopkit.variations.admin (module), 35
shopkit.variations.models (module), 35
shopkit.variations.settings (module), 36
shopkit.vat (module), 33
shopkit.vat.advanced (module), 33
shopkit.vat.advanced.models (module), 34
shopkit.vat.advanced.settings (module), 34
shopkit.vat.advanced.views (module), 34
shopkit.vat.simple (module), 33
shopkit.vat.simple.models (module), 33
shopkit.vat.simple.settings (module), 33
shopkit.vat.simple.views (module), 33
ShopKitExceptionBase, 23
StateChangeListener (class in shopkit.core.listeners), 24
StateChangeLogger (class in shopkit.core.listeners), 24
StockedCartBase (class in shopkit.stock.models), 44
StockedCartItemBase (class in shopkit.stock.models), 44
StockedCartItemMixin (class in shopkit.stock.advanced.models), 46
StockedCartItemMixin (class in shopkit.stock.simple.models), 45
StockedCartMixin (class in shopkit.stock.advanced.models), 46
StockedCartMixin (class in shopkit.stock.simple.models), 45
StockedItemBase (class in shopkit.stock.models), 44
StockedItemMixin (class in shopkit.stock.advanced.models), 46
StockedItemMixin (class in shopkit.stock.simple.models), 45
StockedOrderBase (class in shopkit.stock.models), 44
StockedOrderItemBase (class in shopkit.stock.models), 44
StockedOrderItemMixin (class in shopkit.stock.advanced.models), 46
StockedOrderItemMixin (class in shopkit.stock.simple.models), 45
StockedOrderMixin (class in shopkit.stock.advanced.models), 46
StockedOrderMixin (class in shopkit.stock.simple.models), 45

T

test_basic_category() (shopkit.category.tests.CategoryTestMixinBase method), 32
test_basic_product() (shopkit.core.tests.CoreTestMixin method), 23
test_cart() (shopkit.core.tests.CoreTestMixin method), 23
test_cartitem_from_product() (shopkit.core.tests.CoreTestMixin method), 23
test_create_usercustomer() (shopkit.core.tests.CoreTestMixin method), 23

test_order() (shopkit.core.tests.CoreTestMixin method), 23
test_orderitem_from_cartitem() (shopkit.core.tests.CoreTestMixin method), 23
test_orderstate_change_tracking() (shopkit.core.tests.CoreTestMixin method), 23
to_request() (shopkit.core.models.CartBase method), 16
TranslatedEmailingListener (class in shopkit.core.listeners), 24

U

update_discount() (shopkit.discounts.advanced.models.order_models.PersistentDiscountedItemBase method), 43
update_discount() (shopkit.discounts.basemodels.DiscountedOrderBase method), 38
update_discount() (shopkit.discounts.basemodels.DiscountedOrderItemBase method), 39
update_shipping() (shopkit.shipping.advanced.models.order_models.PersistentShippedItemBase method), 28
update_shipping() (shopkit.shipping.basemodels.ShippedOrderBase method), 30
update_shipping() (shopkit.shipping.basemodels.ShippedOrderItemBase method), 30
UserCustomerBase (class in shopkit.core.models), 19

V

VariationCartItemMixin (class in shopkit.variations.models), 36
VariationItemBase (class in shopkit.variations.models), 36
VariationOrderItemMixin (class in shopkit.variations.models), 36
VATItemBase (class in shopkit.vat.simple.models), 33