Shopit Documentation

Release 0.5.2

Dino Perovic

Oct 21, 2019

Contents

A fully featured shop application built on djangoSHOP **framework.**

This project aims to provide a **quick & easy** way to set up a fully featured shop without much hassle.

CHAPTER 1

Features

Shopit comes with the most useful features that a classic shops needs, out of the box.

Here's what you can expect:

- Easily manage **Products** and their variations with custom **Attributes**.
- **Attach** images, videos & files on products.
- Set *Up-sell*, *Cross-sell* and other customized **Relations** between products.
- Create custom checkbox **Flags** for products and categorization.
- Group products in **Category**, **Brand** and **Manufacturer** groups.
- Create discounts and promotional codes with **Modifiers**.
- Add custom **Taxes** for categories and products.
- Enable customer **Reviews** on products.

CHAPTER 2

Contents

2.1 Getting started

Get started with installing and configuring **Shopit**.

2.1.1 Requirements

- Django 1.11
- django-shop as shop framework.
- django-cms for placeholders.
- django-parler to translate everything.
- django-mptt for tree management.
- django-admin-sortable2 to sort stuff.
- django-measurement to add measurements.

2.1.2 Installation

Install using **pip**:

```
pip install django-shopit
```

You should follow django-cms & django-shop installation guide first, and then add the following to your settings:

```
INSTALLED_APPS = [
    ...
    'adminsortable2',
    'mptt',
    'parler',
    'shopit',
]

SHOP_APP_LABEL = 'shopit'
SHOP_PRODUCT_SUMMARY_SERIALIZER = 'shopit.serializers.ProductSummarySerializer'
SHOP_CART_MODIFIERS = (
    'shop.modifiers.DefaultCartModifier',
    'shopit.modifiers.ShopitCartModifier',
    ...
)
```

Urls

There are two ways to configure the urls. First would be to add to your `urls.py`:

```
urlpatterns = [
    url(r'^shop/', include('shopit.urls')),
    ...
]
```

The second option is to use django-cms apphooks. **Shopit** comes with a couple of those for different application parts. `ShopitApphook` is the main one, and one that should always be attached to a page (if the urls are not already added). Then there are other optional apphooks for *account*, *categorization* & *products*. If you want to keep it simple, and not have to set every application part individually. You can add to your settings:

```
SHOPIT_SINGLE_APPHOOK = True
```

This will load all the neccesary urls under the `ShopitApphook`.

2.2 Product

About the **Product** model.

2.2.1 Type of Products

There are 3 kinds of products:

- `SINGLE` are products without variations.
- `GROUP` are products that hold variants and their common info, they cannot be added to cart.
- `VARIANT` are variations of a product that must select a Group, and set their unique attributes.

Each of the kinds have their set of rules, and a different validation in admin.

Single products

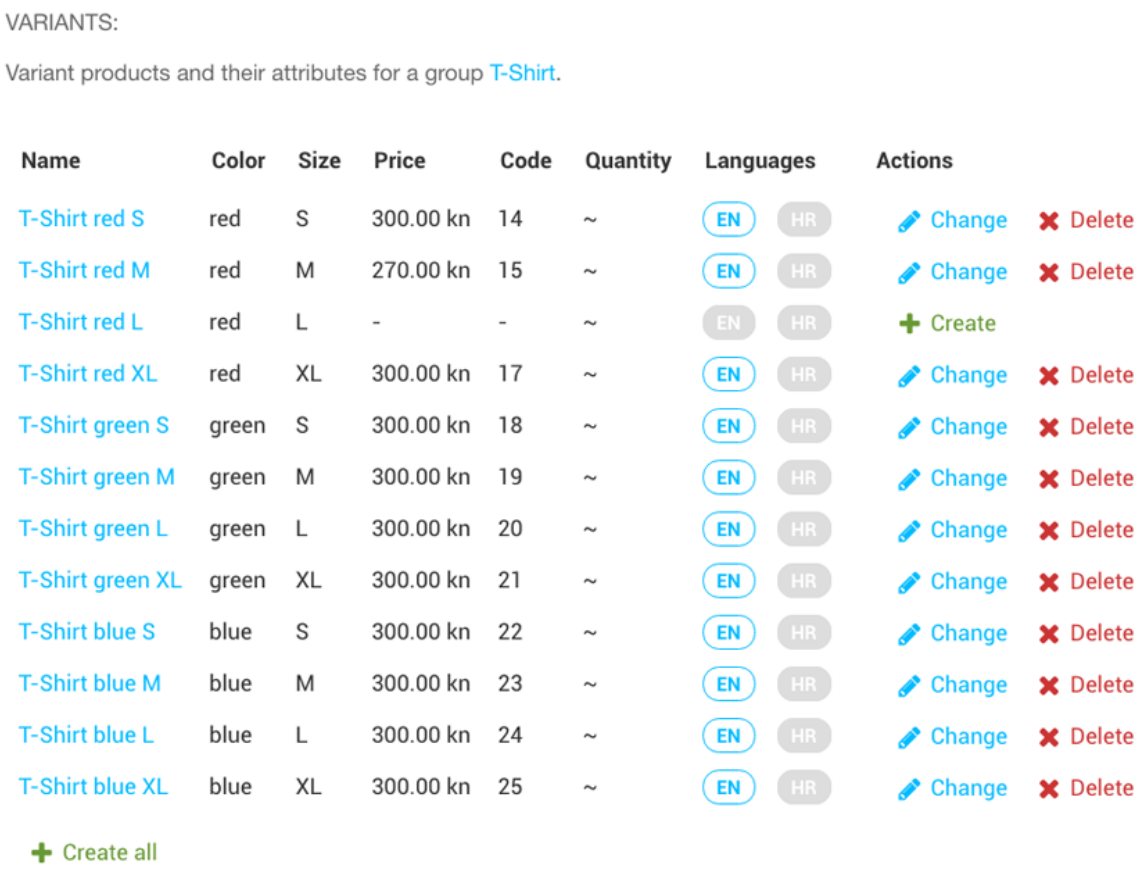
A simplest form of a **Product**. Only requirement is to set the `name`, `slug`, `code` & `unit_price`.

Group products

Group products hold common info of their variations and are not considered an actual product that can be added to the cart. They have to specify `availible_attributes` for the variants to use. Variants of a group that don't use those attributes will be considered *invalid*.

Variant products

Variants must specify a group, as well as their unique set of attributes. All attributes specified in `availible_attributes` of a group, need to be added. Variants are best added through the custom *variants* field in products admin. Every combination of a variant can be created automatically, and invalid variants deleted as well.



Variants can leave most of their fields empty to inherit them from a group. Or choose to override them.

2.2.2 Categorization

Categorization is separated into `Category`, `Brand` & `Manufacturer` models. Tree management is handled by the django-mptt project. **Modifiers** and in case of category, **Tax** can be set in categorization to apply to a group of products.

2.2.3 Pricing & availability

Product pricing section consists of a `unit_price`, `discount`, `tax` & `summary` fields. Discount and tax field are ment for per products use. When different taxes or discounts are required on a product. These values are embeded into a price, when for example you want to have tax included in a price. A summary field shows calculated values in a custom admin field, for convenience.

Stock

`quantity` field is used to keep a record of availible units to ship. Leave empty if product is always available, or set to `0` if product is not available.

2.2.4 Flags

Custom checkbox flags can be added to the **Product** or to a categorization layer. This allows to easily separate a group of products to display in a different way on site.

2.2.5 Measurements

Measuerments fields are powered by the django-measurement project. `width`, `height`, `depth`, and `weight` is available.

2.2.6 Attributes

Attribute model lets you design custom attributes with their choices. These attributes are then selected in `availible_attributes` field on a group product, and used to create product variations.

2.2.7 Other inlines

Product model has couple of inline models like `Attachment` that allows you to add **image**, **video** & **file** attachments to a product. `Relation` that allows you to add customized relations between products. `Review` that let's you manage product customer reviews.

2.3 Modifier

Modifiers allow you to create different cart and cart item modifications on a specific set of **Products**. You can assign them to any **Categorization** model and to a **Product**.

There are 3 kind of **Modifiers**:

- `STANDARD` affects the product regardles. Usefull for taxing specific set of products.

- `DISCOUNT` checks for a "Discountable" flag on a product and should be negative.
- `CART` will affect an entire cart.

Modifiers allow you set either the amount or the percentage with what the price will be modified. Those values should be negative when creating discounts.

2.3.1 Modifier Conditions

Conditions can be created for a modifier that then must be met to make the modifier valid. You can use the default ones, or create custom conditions.

Default conditions

Shopit comes with a couple of simple conditions available for use. When creating a **Modifier** in admin, you'll get to choose from:

- `PriceGreaterThanCondition`
- `PriceLessThanCondition`
- `QuantityGreaterThanCondition`
- `QuantityLessThanCondition`

They all accept a value. Quantity conditions control only modifiers on cart items.

Create custom conditions

To create a custom conditions you must extend from `shopit.modifier_conditions.ModifierCondition` and then you can implement methods `cart_item_condition` and `cart_condition`. They both accept an optional `value` argument as decimal number that can be passed in when selecting the condition.

```
from datetime import datetime

from shopit.modifier_conditions import ModifierCondition


class DayIsOddCondition(ModifierCondition):
    name = 'Day is odd'

    def cart_item_condition(self, request, cart_item, value=None):
        return self.day_is_odd()

    def cart_condition(self, request, cart, value=None):
        return self.day_is_odd()

    def day_is_odd(self):
        return datetime.today().day % 2 == 1
```

Now with the condition above, when selected on a modifier it will only be active when the day is odd. Since both methods `cart_item_condition` and `cart_condition` are overriden, the condition will control the modifier in cases both when it's applied to a cart item, or an entire cart. By default when not overriden those methods return `True`.

Last thing do to is add the path to your condition to `SHOPIT_MODIFIER_CONDITIONS` list.

```
SHOPIT_MODIFIER_CONDITIONS = [
    'shopit.modifier_conditions.PriceGreaterThanCondition',
    'shopit.modifier_conditions.PriceLessThanCondition',
    'shopit.modifier_conditions.QuantityGreaterThanCondition',
    'shopit.modifier_conditions.QuantityLessThanCondition',
    'myapp.modifier_conditions.DayIsOddCondition',
]
```

2.3.2 Discount codes

Other than conditions, modifiers can be limited to a set of discount codes that makes them valid. To achive that you need to create a `DiscountCode` and assign it to a modifier. When active discount codes exist on a modifier, it is no longer active without one of those codes applied to the cart.

Discount codes can also be limited for the specific customer to use only.

2.4 Templates

You can use django-cms cascade plugins provided by django-shop to generate your cart, watch, checkout, account & catalog pages. But if don't want to add the plugins yourself, **Shopit** comes with prebuild html templates for those pages. Barebones and with simple jQuery implementation of front-end actions for you to easily override. This will help you have a clean & simple starting boilerplate to build apon.

2.4.1 Account

Account templates are located in `templates/shopit/account/*` and consist of:

- `account_detail.html`
- `account_login.html`
- `account_order_detail.html`
- `account_order_list.html`
- `account_register.html`
- `account_reset.html`
- `account_reset_confirm.html`
- `account_settings.html`

2.4.2 Catalog

Catalog templates are located in `templates/shopit/catalog/*` and consist of:

- `categorization_detail.html`
- `categorization_list.html`
- `product_detail.html`
- `product_list.html`

There are general categorization templates that handle all categorization views by default. Categorization objects and lists are passed into context as `categorization` and `categorization_list` as well as an actual model name representation, for example **Category** views will also have `category` and `category_list` accessible. You can also create a template for a specific categorization by using it's model name. For eg. for **Brand** model you can create `brand_detail.html` and `brand_list.html`.

2.4.3 Shop

Shop templates are located in `templates/shopit/shop/*` and consist of:

- `cart.html`
- `checkout.html`
- `thanks.html`
- `watch.html`

2.4.4 Templatetags

To use the **Shopit** templatetags library. Put `{% load shopit_tags %}` in your templates.

Filters

```
# Cast a number to a Money format.
{{ number|moneyformat }}
```

Simple tags

```
# Update the querystring maintaining the existant keys.
{% query_transform color 'black' size='XL' %}

# Fetch a set of products.
{% get_products 3 categories=3 brands='apple,samsung' flags='featured,awesome' as␣
↪products %}
{% get_products categories='phones' price_from=120 as products %}

# Fetch a set of categorization objects.
{% get_categorization 'category' limit=3 level=1 depth=2 as categories %}
{% get_categorization 'brand' limit=3 level=1 depth=2 as brands %}
{% get_categorization 'manufacturer' products=product_list limit=3 level=1 depth=2 as␣
↪manufacturers %}

# Fetch a single flag, or a set of flags.
{% get_flags 'featured' as featured_flag %}
{% get_flags products=product_list level=1 parent='featured' as featured_flags %}

# Fetch a single modifier, or a set of modifiers. Setting filtering to True
# returns only the modifiers eligible for filtering.
{% get_modifiers 'special-discount' as special_discount_mod %}
{% get_modifiers products=product_list filtering=True %}

# Fetch attributes for the set of products.
```

(continues on next page)

(continued from previous page)

```
{% get_attributes product as attributes %}

# Get min and max price with the steps in between for a set of products.
{% get_price_steps 3 product as price_steps %}
```

Inclusion tags

These are the templates to be included with inclusion tags. They are located in `templates/shopit/includes/*` and consist of:

- `add_to_cart.html`
- `cart.html`
- `order.html`

To include the templates you can write the following:

```
# Show add to cart button for the 'product' in context.
{% add_to_cart %}

# Show add to cart button for specified product with watch button included.
{% add_to_cart product watch=True %}

# Show editable cart.
{% cart %}

# Show static cart.
{% cart editable=False %}

# Show latest order.
{% order number="2018-00001" %}

# show specific order.
{% order instance %}
```

2.5 Settings

Available settings to override.

2.5.1 Error messages

A dictionary with error messages used in **Shopit**.

```
SHOPIT_ERROR_MESSAGES = {
    'duplicate_slug': _("This slug is already used. Try another one."),
    'group_has_group': _("Only variant products have a group."),
    'variant_no_group': _("Variants must have a group selected."),
    'variant_has_category': _("Variant products can't specify categorization. It's␣
↪inherited from their group."),
    'varinat_group_variant': _("Can't set group to variant."),
```

(continues on next page)

(continued from previous page)

```
    'not_group_has_variants': _("This product has variants, you need to delete them␣
↪before changing it's kind."),
    'not_group_has_available_attributes': _("Only group products can have Availible␣
↪attributes specified."),
    'group_no_available_attributes': _("Group product should have Availible␣
↪attributes specified."),
    'variant_has_tax': _("Variant products can't specify tax, their group tax␣
↪percentage will be used instead."),
    'variant_no_attributes': _("Variant must have their unique set of attributes␣
↪specified."),
    'variant_already_exists': _("A Variant with this attributes for selected group␣
↪already exists."),
    'not_variant_has_attributes': _("Only Variant products can have attributes."),
    'attribute_no_choices': _("Choices must be specified."),
    'attribute_duplicate_choices': _("Attribute can't have duplicate choices."),
    'incorrect_attribute_choice': _("Selected choice doesn't match the seleced␣
↪attribute."),
    'no_attachment_or_url': _("Missing the attachment or url."),
    'wrong_extension': _("File extension not allowed for this attachment kind."),
    'discount_not_negative': _('A discount should be subtracting the price, amount or␣
↪percent needs to be negative.'),
    'variant_has_relations': _('Only Single and Group products can have relations.'),
    'relation_base_is_product': _("You can't set relation to self."),
    'modifier_no_condition_path': _("You have to select a condition."),
    'cart_discount_code_exists': _("Code is already applied to your cart."),
    'cart_discount_code_invalid': _("Code is invalid or expired."),
    'cart_discount_code_wrong_customer': _("Code is invalid or expired."),
}
```

2.5.2 Address

Country choices used in checkout address forms. If empty all countries are used from `shopit.models.address.ISO_3166_CODES`.

```
SHOPIT_ADDRESS_COUNTRIES = ()
```

A primary address to be used in a checkout proccess. Can be 'shipping' or 'billing'. Depending on wich address is selected, the other one will get the option to use the primary one instead of having to fill it out.

```
SHOPIT_PRIMARY_ADDRESS = 'shipping'
```

2.5.3 Customer

A flag to control if customer's phone number is required.

```
SHOPIT_PHONE_NUMBER_REQUIRED = False
```

2.5.4 Product

A list of base serializer fields for a common product.

```
SHOPIT_PRODUCT_SERIALIZER_FIELDS = [
    'id', 'name', 'slug', 'caption', 'code', 'kind', 'url', 'add_to_cart_url', 'price
↪', 'is_available',
]
```

Above is the default config, here's a list of all available fields:

```
['id', 'name', 'slug', 'caption', 'code', 'kind', 'url', 'add_to_cart_url', 'price',
↪'is_available',
 'description', 'unit_price', 'discount', 'tax', 'availability', 'category', 'brand',
↪'manufacturer',
 'discountable', 'modifiers', 'flags', 'width', 'height', 'depth', 'weight',
↪'available_attributes',
 'group',  'attributes', 'published', 'quantity', 'order', 'active', 'created_at',
↪'updated_at',
 'is_single', 'is_group', 'is_variant', 'is_discounted', 'is_taxed',  'discount_
↪percent', 'tax_percent',
 'discount_amount', 'tax_amount', 'variants', 'variations', 'attachments', 'relations
↪', 'reviews']
```

A list of serializer fields for a product detail.

```
SHOPIT_PRODUCT_DETAIL_SERIALIZER_FIELDS = SHOPIT_PRODUCT_SERIALIZER_FIELDS + [
↪'variants', 'attributes']
```

Template choices used when rendering an attribute.

```
SHOPIT_ATTRIBUTE_TEMPLATES = ()
```

Relation kind choices on a `ProductRelation` model.

```
SHOPIT_RELATION_KINDS = (
    ('up-sell', _('Up-sell')),
    ('cross-sell', _('Cross-sell')),
)
```

Rating choices for product reviews.

```
SHOPIT_REVIEW_RATINGS = ()
```

Is review active by default when created.

```
SHOPIT_REVIEW_ACTIVE_DEFAULT = True
```

A boolean that enables you to optimize `ProductListView` and `CategoryDetailView` when products are fetched asynchronously (ajax).

```
SHOPIT_ASYNC_PRODUCT_LIST = False
```

A boolean to control if `product_list` is added to context when accessing `ProductListView` or `CategoryDetailView`.

```
SHOPIT_ADD_PRODUCT_LIST_TO_CONTEXT = not SHOPIT_ASYNC_PRODUCT_LIST
```

A default product list ordering. Must be on of 'namel-namelpricel-price'.

```
SHOPIT_DEFAULT_PRODUCT_ORDER = None
```

2.5.5 Flag

Template choices used when rendering a Flag.

```
SHOPIT_FLAG_TEMPLATES = ()
```

2.5.6 Modifier

A list of `ModifierCondition` classes that will be used as choices for conditions in a Modifier.

```
SHOPIT_MODIFIER_CONDITIONS = [
    'shopit.modifier_conditions.PriceGreaterThanCondition',
    'shopit.modifier_conditions.PriceLessThanCondition',
    'shopit.modifier_conditions.QuantityGreaterThanCondition',
    'shopit.modifier_conditions.QuantityLessThanCondition',
]
```

2.5.7 Text editor

A text editor widget used to render a rich textarea in **Shopit**.

```
SHOPIT_TEXT_EDITOR = 'djangocms_text_ckeditor.widgets.TextEditorWidget'
```

2.5.8 Single apphook

Load urls under a single `ShopitApphook`, or leave the ability to add apps separately.

```
SHOPIT_SINGLE_APPHOOK = False
```

2.5.9 Filter attributes

Designates if products of kind `VARIANT` should be included in attribute filtered results.

```
SHOPIT_FILTER_ATTRIBUTES_INCLUDES_VARIANTS = False
```

2.6 Release notes

Release notes for **Shopit**.

2.6.1 0.5.2

- Fix setup requirement.

2.6.2 0.5.1

- Drop *Django 1.10* support.
- In *ProductDetailView*, check for renderer format before adding django-cms menu related items.
- Remove *PhoneNumberField* from the project, use simple *CharField* instead.
- Lock requirements.

2.6.3 0.5.0

- Rename package from *djangoshop-shopit* to *django-shopit*.

2.6.4 0.4.3

- Fix encoding error in product admin *get_name* method.
- Add *phonenumbers* library to requirements.

2.6.5 0.4.2

- Fixes #7 - "unhashable type: 'MoneyInEUR'" error in *get_price_steps* templatetag.

2.6.6 0.4.1

- Small fixes in admin.
- Fix indentation in admin help text for `djangocms-admin-style`.
- Refactor tests.

2.6.7 0.4.0

- Add support for *Django 1.11* & *DjangoSHOP 0.12.x*.
- Handle tousand separator when displaying money in admin.
- Add ability to pass in `order_number` to `order` templatetag.
- Add `num_uses` to list display for Discount Code.
- After order was populated with cart data, delete discount codes.
- Add ability to send *validate* key when updating the cart via POST. In which case the promo code gets validated without applying it to cart.
- Add setting `SHOPIT_DEFAULT_PRODUCT_ORDER` to control default ordering of products.
- Add ability to override `ProductSerializer` fields through the `fields` GET property.
- Add `attribute_choices` to product serializer fields.
- Add `template` field to Flag model, adn a `SHOPIT_FLAG_TEMPLATES` setting.
- Add `path` to the Flag serializer.
- Include categorization flags on a product.

- Fix flag serializer field.
- Use attachment `subject_location` when generating a thumbnail.
- Add ability to pass in `get_count` as boolean through the `request.GET` object when in `ProductListView` and `CategoryDetailView`. This applies in non HTML formated response and returns the count of all (filtered) products as `{'count': 300}`.
- Simplify urls into a single `urls.py` since https://github.com/divio/django-cms/pull/5898 was merged.
- Separate admin modules into multiple files.
- Move settings from `settings.py` to `conf.py` and re-format based on *djangoSHOP's* settings pattern.
- Add `SHOPIT_ASYNC_PRODUCT_LIST` and `SHOPIT_ADD_PRODUCT_LIST_TO_CONTEXT` settings to optimize `ProductListView` and `CategoryDetailView`.
- Bump `django-cms` requirement to 3.5.
- Set default prices to zero.
- Fix field indentation in models and forms to follow Django's style guide.
- Various bugfixes.

Attention: Requires `python manage.py migrate shopit` to set default price and amount Money fields, and add a template field to the Flag model.

2.6.8 0.3.0

- Handle `InvalidImageFormatError` error when generating thumbnails.
- Add support for djangoSHOP 0.11.

2.6.9 0.2.3

- Add `never_cache` decorators to account, review and watch views.
- Optimize `get_flags` templatetag when filtering by products.
- Add `content` field as `PlaceholderField` to categorization models.
- Force setting priority on address form, order existant addresses by priority.
- Update `query_transform` templatetag to remove empty values.
- Add missing `FlagModelForm` to `FlagAdmin`.
- Fix Flag unicode error in `__str__`.
- Re-work the reviews, making them non-translatable. Not compatible with the old reviews, make sure to save them (if you have any) before upgrading. A way for adding reviews was not provided before so this should not be the case.
- Add setting `SHOPIT_REVIEW_ACTIVE_DEFAULT`. This decides if created reviews are active by default.
- Handle updating shopping cart via ajax, add success messages to it.
- Remove *CartDiscountCode's* from cart when emptying it, make last applied code appears as active.
- Add *PhoneNumberField* field to the customer, add setting `SHOPIT_PHONE_NUMBER_REQUIRED` that defaults to `False`.

- Refactor address forms, enable using either 'shipping' or 'billing' form as primary. added setting `SHOPIT_PRIMARY_ADDRESS`.
- Fix address country choices.
- Add and track num uses on a *DiscountCode*, alter the admin to display new values.
- Enable frontend editing of categorization and product models.
- Fix *AccountOrderDetail* view not returning the correct order.
- Handle NoReverseMatch for `add_to_cart_url` in a Product serializer.

Attention: Requires `python manage.py migrate shopit` to add/remove fields on a Review model, as well as add `phone_number` field on Customer model, `content` field on Categorization models and `max_uses`, `num_uses` on *DiscountCode*.

Note: If migrating with categorization models already added. You'll need to save each models again for the `content` PlaceholderField to appear.

2.6.10 0.2.2

- Add filtering by modifiers.
- Update django-shop requirement to 0.10.2.

2.6.11 0.2.1

- Fixes problem with migrations.

2.6.12 0.2.0

- Add support for *Django 1.10* & *DjangoSHOP 0.10.x*.
- Alter templates to use Bootstrap 4 by default.
- Create example project, move tests.
- Rename description & caption fields to start with underscore.

Attention: Requires `python manage.py migrate shopit` to add a product code to the CartItem, rename description & caption fields, as well as adding an additional setting `SHOP_PRODUCT_SUMMARY_SERIALIZER = 'shopit.serializers.ProductSummarySerializer'`.

2.6.13 0.1.4

- Add *description* field to categorization models.
- Move variant generator methods from admin to the model. Now `create_all_variants` and `create_variant` are available on the model.

- Update add to cart `get_context` to ensure correct product translation is returned.

Attention: Requires `python manage.py migrate shopit` to create description field on categorization models.

2.6.14 0.1.3

- Bugfixes.
- Fix `get_object` and `get_queryset` in product views returning inconsistant results.
- Add `get_view_url` to product detail view to return correct translated url.

2.6.15 0.1.2

- Add price range filtering in `get_products` templatetag.
- Move product filtering to a manager.
- Allow mutiple flags to be passed to the `get_products` templatetag.
- Optimize attribute filtering with *prefetch_related*.
- Enable sorting the products.
- Don't fetch flags from categorization on a product. Categorization flags are used separately to mark categorization and the don't affect the products.
- Fix templatetags.
- Add option to limit `get_categorization` templatetag to a set of products.
- Enable filtering categorization and flags via querystring. Change price range querystrings.
- Add `get_flags` templatetag.
- Make *Flag* model an mptt model with a parent field.
- Show flags as filter_horizontal instead of CheckboxInput in product admin.
- Show localized amounts in product admin summary field.
- Use `as_decimal` when displaying price steps in template instead of floatformat.

Attention: Requires `python manage.py migrate shopit` to create mptt fields on a Flag model.

2.6.16 0.1.1

- Ensure customer is recognized before registering a new account. This works around an error **"Unable to proceed as guest without items in the cart"** when registering without a cart.
- Make fields in product serializer editable through settings, set optimized defaults.
- Fix error when mergin dictionaries in python3.
- Remove redundant code.
- Fix trying to generate image thumbnail on attachment when *file* is None.

- Fix weight setter setting width instead of weight.

2.6.17 0.1.0

- Initial release.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search