Follow mbrochh

MARTIN BROCHHAUS

ENTREPRENEUR, SOFTWARE ENGINEER, FREELANCE WEBDESIGNER, HOUSE DJ, PHOTOGRAPHER NOOB, WORLD TRAVELER

ASK ARCHIVE RANDOM RSS SEARCH

30TH JUN 2011 | 12 NOTES

DJANGO-SHOP ADVENTURES: TAXES AND FLAT RATE SHIPPING

Today we are finishing the last bit that the **getting started document** of the shop suggests as of now: Adding a value added tax.

Because this is so simple that it is barely worth its own blog post I will also add a little hack that introduces a flat shipping fee. I know, **earlier** we have already activated the FlatRateShipping backend that ships with the shop. However I don't really like that solution because it shows a page saying "We will add XX EUR shipping costs" as the last step in the checkout process. I would rather want to see the shipping costs as an item in the cart in the checkout view.

Adding value added tax

We will add a cart modifier that will inject a new cart item called 'Total tax' to the cart. For this we need to create a file "modifiers.py" in our "myshop" app. The code should look like this:

```
""" Cart modifierts for the myshop app of django-shop-adventures.""
import decimal

from django.conf import settings

from shop.cart.cart_modifiers_base import BaseCartModifier


class FixedTaxRate(BaseCartModifier):
    """
    This will add 19% of the subtotal of the order to the total.
```

```python

    It will also take all extra price fields into consideration and
    to them as well.
    """

    def process_cart(self, cart):
        total = cart.subtotal_price
        for item in cart.extra_price_fields:
            total += item[1]
        taxes = total * decimal.Decimal('0.19')
        to_append = ('Taxes total', taxes)
        cart.extra_price_fields.append(to_append)
        return cart
```

gistfile1.py hosted with ❤ by **GitHub** **view raw**

Note that we override the “process_cart” method, not the “add_extra_cart_price_field” method. This makes sure that the tax will be added as the very last step, because I want the shipping costs that I will add in the next step to be taxable as well.

After that we need to add ‘myshop.modifiers.FixedTaxRate’ to our SHOP_CART_MODIFIERS setting. Easy!

Adding shipping fees to the cart

At the moment, after entering our shipping and billing address we will be forwarded to the FlatRateShipping backend. This simple class will just show us the shipping fees and set the order to complete. I don’t like this approach. Especially for German (or even EU?) law showing shipping costs that late is illegal (even showing them in the cart is too late, you need to add a hint to the product description. I know. Thank god our politicians assume that we are all stupid braindead zombies).

Anyways if we want the shipping costs to show up in our cart we need to create yet another cart modifier. Since we have our “modifiers.py” already, we can just add the following code to it:

```python
class FixedShippingCosts(BaseCartModifier):
    """
    This will add a fixed amount of money for shipping costs.
    """
    def add_extra_cart_price_field(self, cart):
        cart.extra_price_fields.append(
            ('Shipping costs', decimal.Decimal(
                settings.SHOP_SHIPPING_FLAT_RATE)))
        return cart
```

gistfile1.py hosted with ❤ by **GitHub** **view raw**

Again we need to add the new modifier to our settings.py. The setting should now look like this:

```python
SHOP_CART_MODIFIERS = [
    'shop_simplevariations.cart_modifier.ProductOptionsModifier',
    'myshop.modifiers.FixedShippingCosts',
    'myshop.modifiers.FixedTaxRate',
]
```

gistfile1.py hosted with ❤ by GitHub view raw

Creating a new shipping backend

Now that we have our shipping fees in the cart we don't need the FlatRateShipping backend any more. Because the shop system works in such a way that we get forwarded to any shipping backend after the checkout, we need to provide a SkipShippingBackend that does nothing but forward to the payment backend (which does nothing but forward to the final thank you page).

The SkipShippingBackend could look like this and could live in a new "shipping.py" file within our "myshop" app:

```python
# -*- coding: utf-8 -*-
"""Shipping backend that skips the whole shipping process."""

from django.conf.urls.defaults import patterns, url


class SkipShippingBackend(object):

    backend_name = "Skip Shipping Backend"
    url_namespace = "skip-shipping"

    def __init__(self, shop):
        self.shop = shop

    def simple_view(self, request):
        """
        This simple view does nothing but forward to the final URL.
        money is sent, the shop owner can set this order to complet
        """
        order = self.shop.get_order(request)
        return self.shop.finished(order)

    def get_urls(self):
        urlpatterns = patterns('',
            url(r'^$', self.simple_view, name='skip-shipping' ),
        )
        return urlpatterns
```

gistfile1.py hosted with ❤ by GitHub view raw

You already guess it: After this we need to change our SHOP_SHIPPING_BACKEND setting to 'myshop.shipping.SkipShippingBackend'.

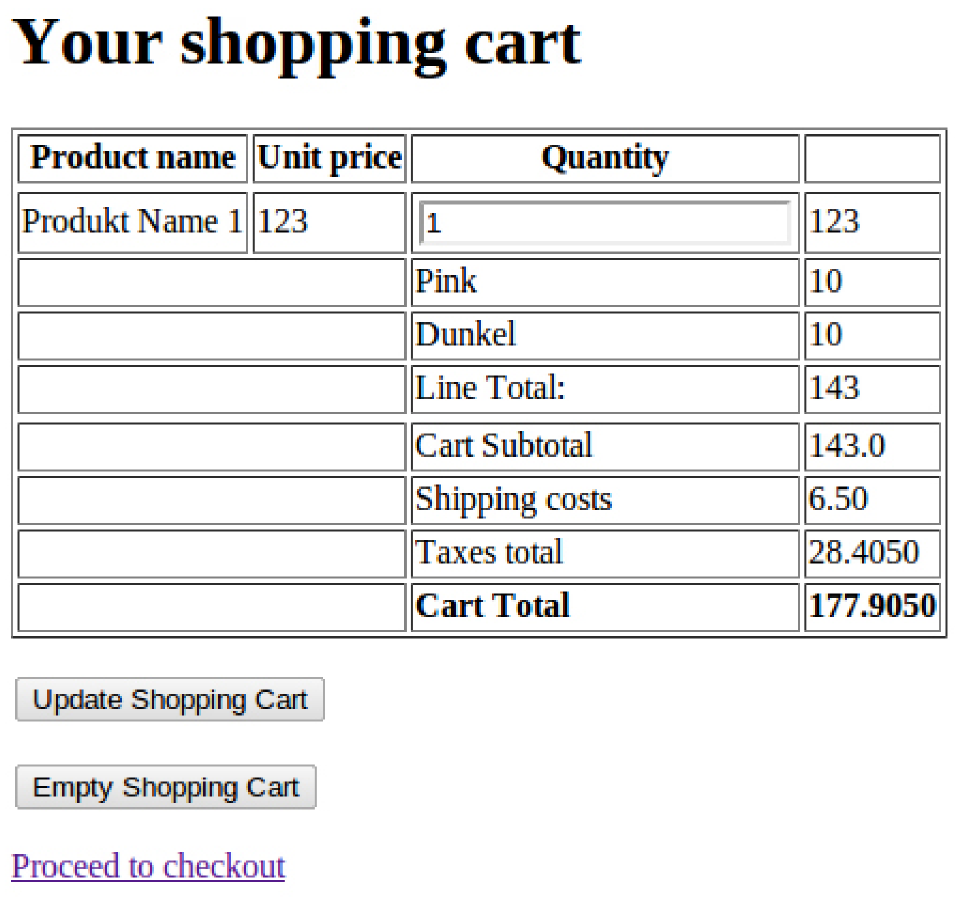
Skipping the backends selection

Now there is only one glitch left. After the checkout view, where we have to enter shipping and billing address we also have to select the shipping and payment backends. In our case that doesn't make a lot of sense because we only have one option for each. Therefore we can override the template at "shop/checkout/selection.html" and hide the {{ shipping_billing_form.as_p }} part. However this form's fields are mandatory, so we need to add two hidden fields and fake the user's selection:

```
<html>
<body>
  <h1>Shipping and Billing</h1>
  <form method="POST">
    {% csrf_token %}
    <h3>Your shipping address</h3>
    {{ shipping_address.as_p }}
    <h3>Your billing address</h3>
    {{ billing_address.as_p }}
    <input type=hidden name="shipping_method" value="skip-shipping'
    <input type=hidden name="payment_method" value="pay-on-delivery
    <button type="submit">Save</button>
  </form>
</body>
</html>
```

gistfile1.html hosted with ❤ by GitHub view raw

Bingo. Your cart now should look something like this:



Your shopping cart

Product name	Unit price	Quantity	
Produkt Name 1	123	1	123
		Pink	10
		Dunkel	10
		Line Total:	143
		Cart Subtotal	143.0
		Shipping costs	6.50
		Taxes total	28.4050
		Cart Total	**177.9050**

As always, the code for this blog post can be found on **github**.

mbrochh posted this