

## PwA Store 2.0

### Contents

PwA Store 2.0.....	1
1. Introduction .....	1
2. Design Consideration .....	2
3. Sample Expected Result .....	2
4. Featured Products.....	4
4.1 Key Tasks .....	4
4.2 Product Model .....	4
4.3 Data Preparation.....	5
4.4 Update View.....	5
5. Category .....	5
5.1 Key Tasks .....	5
5.2 MVT and Routing .....	6
5.3 Test.....	6
6. Orders App .....	6
6.1 How it works .....	6
6.2 Order Model.....	7
6.3 OrderItem Model .....	7
6.4 New Order Page .....	7
6.4.1 View.....	8
6.4.2 Template .....	9
6.5 Order Confirmation Page .....	10
6.5.1 Tips .....	10
3. Test.....	11

### 1. Introduction

After you have built product and cart function, you will add new functions to PwA Store.

1. The home page will display featured products.
2. A user can select a product category and view the products in that category.

3. A user can place an order.

## 2. Design Consideration

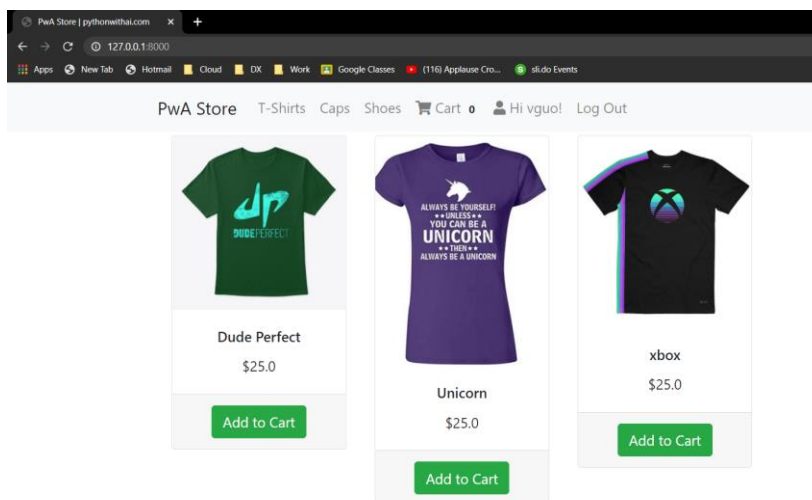
Before you jump into code, do some design work. It will help your coding a lot. Pencil and paper are your friend.

1. Start from a user's perspective. Think how a user uses your website.
2. It's important to design the flow, how to navigate from one page to another. Drawing page navigation flow on paper could help.
3. For each page, think about
  - What should the page look like?
  - What data is needed?
  - Where do I get data from? Data could be from url or model (database).
  - Do I need to filter data such as products in certain category?

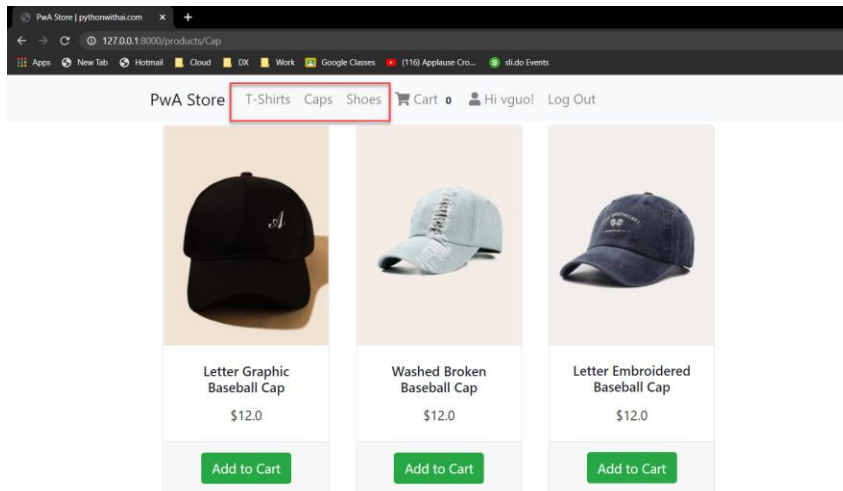
## 3. Sample Expected Result

Please turn in screenshots similar to the followings. You can use any products or categories.

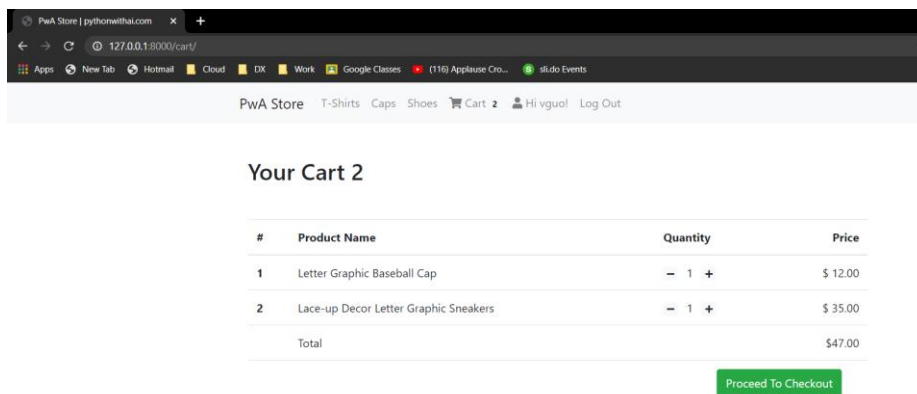
1. The home page displays featured products.



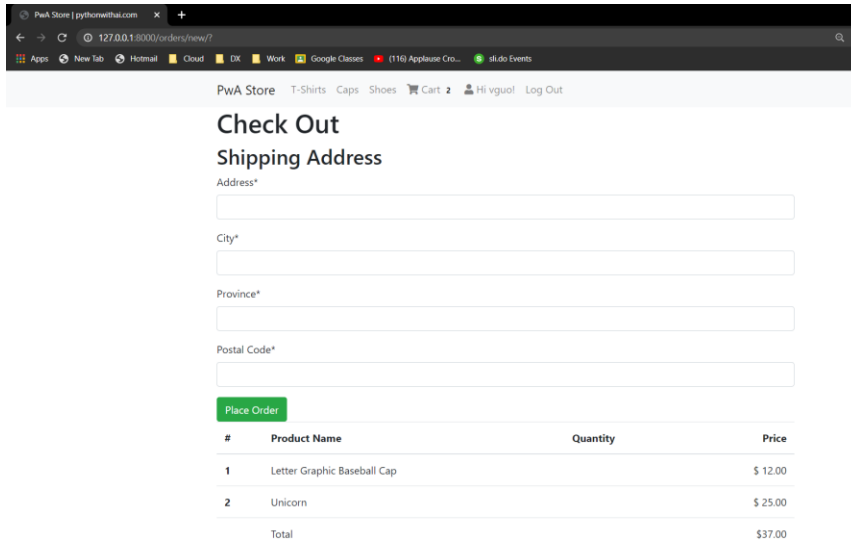
2. The user clicks a category to view products in that category.



3. The user can add products to cart.
4. The user clicks cart to view cart items.
5. The user clicks Proceed to Checkout



6. The user fills in shipping address and click Place Order.



**Check Out**  
**Shipping Address**

Address\*

City\*

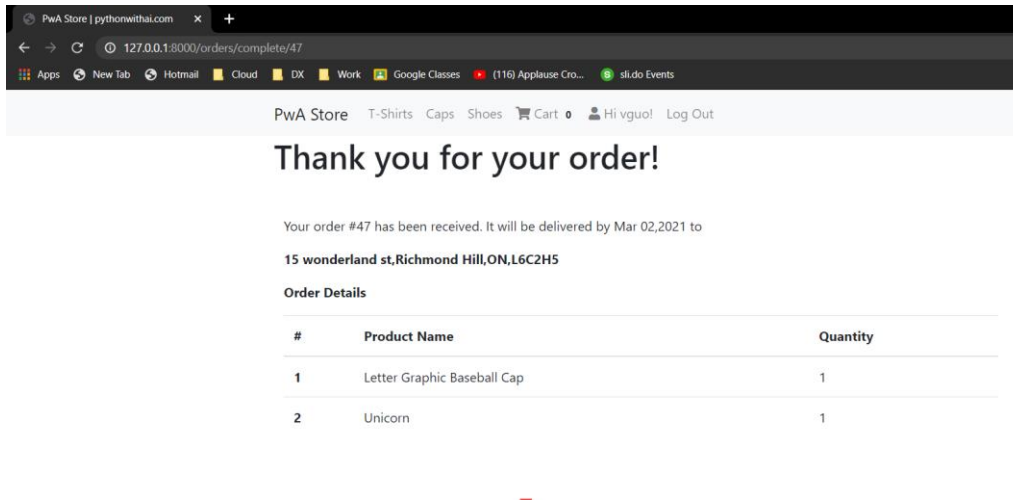
Province\*

Postal Code\*

**Place Order**

#	Product Name	Quantity	Price
1	Letter Graphic Baseball Cap		\$ 12.00
2	Unicorn		\$ 25.00
Total			\$37.00

7. The user sees the confirmation page.



**Thank you for your order!**

Your order #47 has been received. It will be delivered by Mar 02, 2021 to  
**15 wonderland st, Richmond Hill, ON, L6C2H5**

**Order Details**

#	Product Name	Quantity
1	Letter Graphic Baseball Cap	1
2	Unicorn	1

## 4. Featured Products

### 4.1 Key Tasks

1. You need to add a new field to indicate a product is a featured product. Don't forget to do migration so the table will get updated.
2. You need to set couple of products as featured products through admin site.
3. In the home page view, you will need to filter products by featured field.

### 4.2 Product Model

1. Add a new field called featured to Product model. If it is true, it will be a featured product. You will need to edit products/models.py file.

Field	Description	Data Type
Featured	Set to true to indicate a product is featured.	BooleanField, the default value is False.

2. Migrate the data model so the new field will be reflected in the database.

#### 4.3 Data Preparation

1. Login to the admin site, add three categories.
2. Add several of products to each category.
3. Mark several of products as featured products.

#### 4.4 Update the Home View

You will need to update Home view in products/views.py. Instead of get all products, you will need to filter products which featured field is True.

**Tip:** You can add logic in get\_queryset() function to only read products with featured set to True.

```
def get_queryset(self):  
    # add your filter logic here
```

Once you add proper filter logic, test your home page. You should only see products with featured set to True.

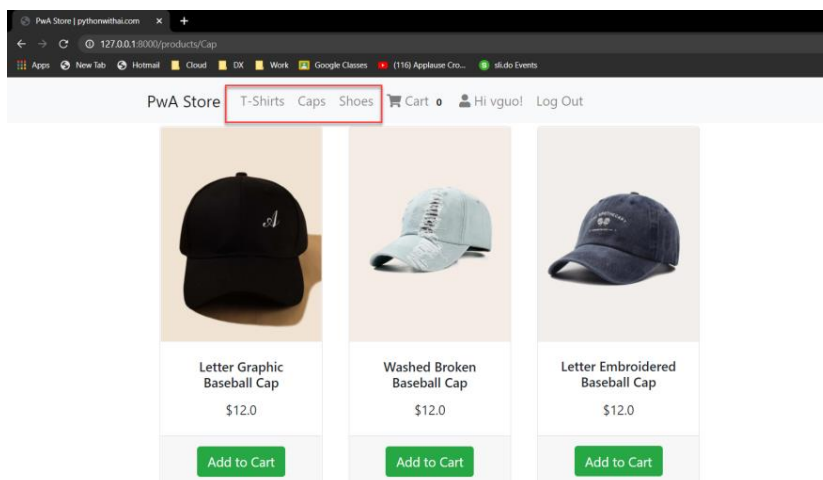
### 5. Category

#### 5.1 Key Tasks

1. Add category links in navigation bar, such as products/<category>. For example:

products/cap  
products/t-shirts

2. Create a new view called ProductListView. You will need to filter products by the category. The category is from url.
3. Create a template products/products.html



## 5.2 MVT and Routing

	Object	File
URL	products/<category>	products/urls.py
View	productListView	Products/views.py
Template	products/products.html	templates/products/products.html

Tip:

In ProductListView, you need to get category from URL, and query products in that category.

```
def get_queryset(self):
    # get category name from url, and then query category id
    categoryid = Category.objects.get(title=self.kwargs['category'])
    # query products in that category
    return Product.objects.filter(category=categoryid)
```

Add proper category links in navbar.html.

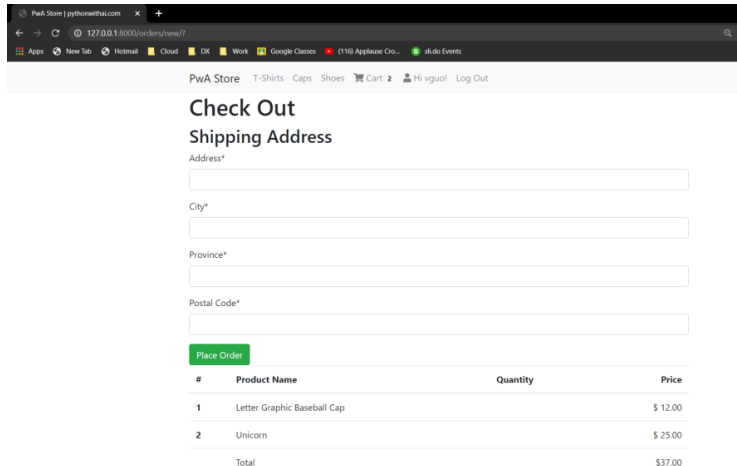
## 5.3 Test

If you click different category in the navigation bar, you should see different products.

## 6. Orders App

### 6.1 How it works

1. A user adds couple of products to cart.
2. The user clicks cart to see the order page.



**Check Out**

**Shipping Address**

Address\*

City\*

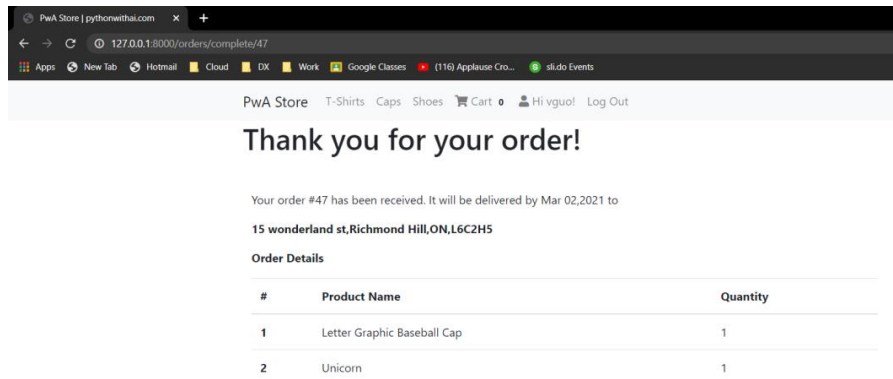
Province\*

Postal Code\*

**Place Order**

#	Product Name	Quantity	Price
1	Letter Graphic Baseball Cap		\$ 12.00
2	Unicorn		\$ 25.00
Total			\$37.00

3. The user fills in shipping address, clicks Place order button.
4. The user sees the confirmation page.



5. Cart is reset to zero.

You will need create a new app called order.

## 6.2 Order Model

Create Order model in orders/models.py. Order model will be used to track user orders.

Field	Description	Data Type
user	User who placed the order	ForeignKey to User
orderdate	When the order is placed	DateTimeField
shippingdate	The estimated shipping date	DatetimeField
address	Street of the shipping address	CharField, up to 50 chars
city	City of the shipping address	CharField, up to 50 chars
province	Province of the shipping address	CharField, up to 50 chars
postalcode	Postal Code of the shipping address	CharField, up to 7 chars

The string representation of Order model is orderdate in Feb. 28, 2021, 1:47 a.m. format.

Tips:

```
orderdate.strftime("%b.%d,%Y,%I:%M %p")
```

## 6.3 OrderItem Model

Each Order can have multiple products, which are called order items. Create OrderItem model in orders/models.py. OrderItem model has the follow fields.

Field	Description	Data Type
order	The order that the item belongs to	ForeignKey to Order
item	The product	ForeignKey to Product
quantity	Number of ordered items	Integer
Price	The unit price of the item	FloatField

The string representation of OrderItem model is product name.

## 6.4 New Order Page

	Object	File
--	--------	------

URL	orders/new/	products/urls.py
View	OrderCreateView	orders/views.py
Template	orders/order_create.html	templates/orders/order_create.html

#### 6.4.1 View

1. The view will handle HTTP GET and POST. You need to define a function called `get_context_data` in the view to get items in the user's cart.
2. The HTTP GET will display items in cart, and a form so a user can fill in shipping address.
3. The HTTP POST will get shipping address from the user and process the order. You will need to define `post(self,requestor)` function to process the order.
4. In the post function, you will need to
  - 1) Add a new order to Order table.
  - 2) Loop through all items in the user's cart, and add them into OrderItem table.
  - 3) Reset cart to zero.
  - 4) Consider the situation that the cart is empty.

```
class OrderCreateForm(ModelForm):
    class Meta:
        model = Order
        fields = ['address', 'city', 'province', 'postalcode']

class OrderCreateView(LoginRequiredMixin, CreateView):
    model = Order
    form_class = OrderCreateForm
    template_name = 'orders/order_create.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['cart_list'] = Cart.objects.filter(user=self.request.user)
        return context

    def post(self, request):
        form = self.form_class(request.POST)
        if form.is_valid():
            form.instance.user = self.request.user
            form.instance.orderdate = datetime.now()
            form.instance.shippingdate = form.instance.orderdate + timedelta(days=3)

            form.save()
            #move cart into order items
```



```

    cart_qs = Cart.objects.filter(user=self.request.user)
    if cart_qs.exists():
        for cart in cart_qs:
            if cart.quantity > 0:
                order_item, created = OrderItem.objects.get_or_create(
                    order=form.instance,
                    item=cart.item,
                    defaults={'quantity': 0, 'price': 0},)

                order_item.order = form.instance
                order_item.item = cart.item
                order_item.quantity = cart.quantity
                order_item.price = cart.item.price
                order_item.save()
                cart.delete()

            return redirect('mainapp:complete-order', pk=order_item.order.pk)
    return render(request, 'order_create.html', {'form': form})

```

#### 6.4.2 Template

```

<!-- templates/orders/order_create.html -->
{% extends 'base.html' %}
{% load cart_tag %} <!--new-->
{% load crispy_forms_tags %} <!--new-->
{% block content %}
    <h1>Check Out</h1>
    <h2>Shipping Address</h2>
    <form action="" method="post">
        {% csrf_token %}
        {{ form|crispy }}
        <input type="submit" class="btn btn-success mr-4" value="Place Order">
    </form>
    <table class="table table-hover">
        <thead>
            <tr>
                <th scope="col">#</th>
                <th scope="col">Product Name</th>
                <th scope="col">Quantity</th>
                <th scope="col" style='text-align: right;'>Price</th>
            </tr>
        </thead>
        <tbody>
            {% for cart in cart_list %}
            <tr>
                <th scope="row">{{ forloop.counter }}</th>
                <td>{{ cart.item.name }}</td>
                <td>
                </td>
            </tr>

```

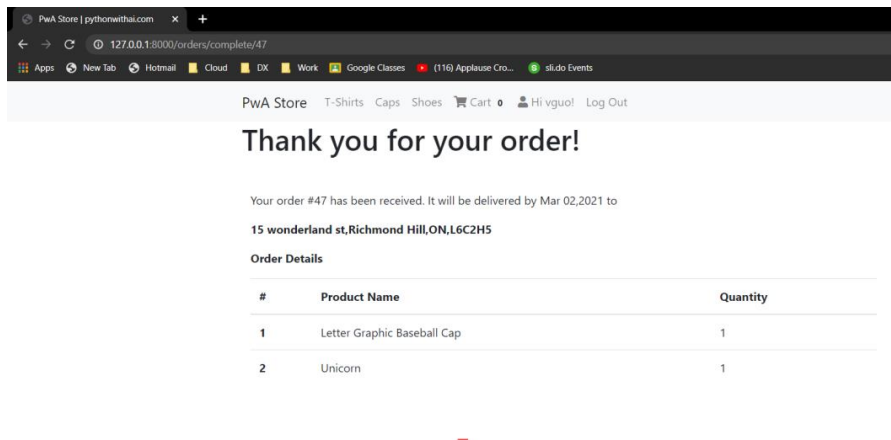
```

        <td style='text-align: right;'>${ {{ cart.get_total | floatformat:2 }} }</td>
    </tr>
    {% endfor %}
    <tr>
        <th scope="row"></th>
        <td colspan="2">Total</td>
        <td style='text-align: right;'>${ {{ request.user | cart_total_sum | floatformat:2 }}}</td>
    </tr>
</tbody>
</table>
{% endblock content %}

```

## 6.5 Order Confirmation Page

The order confirmation page should display shipping address and all items in the order.



Here are objects to create.

	Object	File
URL	orders/complete/<int:pk>	products/urls.py
View	OrderCompleteView	orders/views.py
Template	orders/order_complete.html	templates/orders/order_complete.html

### 6.5.1 Tips

1. OrderCompleteView should have a `get_context_data` function to get user's order items.

```

def get_context_data(self, **kwargs):
    context = super().get_context_data(**kwargs)
    order = Order.objects.get(pk=self.kwargs['pk'], user=self.request.user)
    context['order_list'] = OrderItem.objects.filter(order=order)
    return context

```

2. `order_complete.html` template needs to use a for-loop to display order items.

```
{% for order_item in order_list %}
```

### 3. Test

Test the follow use cases.

1. Sign up a new user.
2. Login as the user.
3. Add products to cart from home page.
4. Click a category, add products to cart.
5. Click cart icon on the navigation bar.
6. Click Process To Checkout button.
7. In check out page, filling shipping address. Click Place Order.
8. In order confirmation page, check the cart is reset to zero.