E-commerce Backend Logic and Steps

1. Setup Project

- Create new projects **ecomprj**
- Create new app (core)
- Install Apps in Settings.py
- Configure Templates
- Configure Static and Media Files in settings and Urls.py
- Create a new view, urls and template then runserver.
- Configure template inheritance and partials

2. Configure Admin Page, Superuser and Jazzmin

- Install **Jazzmin** (pip install django-jazzmin)
- Add jazzmin in **INSTALLED_APPS**
- Add Jazzmin Config Code in Settings.py
- Create Superuser
- Login To Admin Section

3. Custom User Model

- Create new app **userauths**
- Install App in settings.py
- Create custom `class User(AbstractUser):` in **models.py**
- Add **AUTH_USER_MODEL = 'userauths.User'** in settings.py
- Comment out **django.contrib.admin** and **admin urls in Setting.py**
- Run Makemigrations and Migrate and Uncomments Comments
- Create New Superuser
- Register User model In Admin.py
- Login to admin with **email and password**

4. User Register System

- Create new form `class UserRegisterForm(UserCreationForm):` in forms.py
- Write view to **register** `def RegisterView(request):` user
- Configure template to show form
- Login to website from Frontend

5. User Login System

- Write view to **login** `def LoginView(request):` user
- Configure template to grab input field
- Login to website from Frontend

6. User Logout System

- Write view `def LogoutView(request):` to Logout a user
- Configure URL
- Test the Feature

7. Alerts In Django

- Grab Alert Snippet from Bootstrap (version 4)
- Copy and Paste CDN
- Write alert conditional statements

8. Product Model Structure

- Create new Model Class and Add Field for product
- Register Model Classes in Admin

9. List View for Products

- Create Logic to Display only Featured Products in Homepage
- Create New view to List All The Active products from the DB
- Configure Urls.py and Template

10. Category List View

- Create New view to List All active categories.
- Configure Urls.py and Template

11. Product Category List View

- Create New view to List All The Active products from the DB depending on the category selected.
- Configure Urls.py and Template

12. Django Context Processor for Template

- Create a new file context_processor.py in core app
- Install In Setting.py TEMPLATES Section List as `'core.context_processors.default',`
- Now Add Code for Context Processor.

13. Vendor List View

- Create New view to List All active vendors.
- Configure Urls.py and Template

14. Vendor Detail

- Create New view to show the details of the active vendor.
- Configure Urls.py and Template

15. Product Detail View

- Create New view to showcase the Details of a selected Product using **pid**
- Configure Urls.py and Template

16. Related Products in Detail View

- Create query in product detail view

- Refactor related product section in product detail view

17. Tags in Django Using django-taggit

- Install package **pip install django-taggit**
- Configure in settings.py
- Import in models.py and settup tags field using Taggable manager
- Add new tags from django admin and talk about slug
- Create new view to list out all the products related to a simple tags
- Create urls and configure template

18. Rich Text Editor In DJango

- Install package **pip install django-ckeditor**
-

19. Product Rating and Review

- Get all reviews in Product Detail and Lists them out in the template
- Calculate the rating_average for a product
- Create new form for adding reviews
- Write View to Add review
- Create ajax function to create reviews
- Check if a user have made a review before then restrict them from making multiple reviews

20. Searching for Products

- Create view to search for products
- Create Template
- Configure URL

21. Filtering Products using Ajax Jquery

- Iterate Over the vendor and category in the product-lists filter section in templates and add data-filter then the name of the filter type e.g category or vendor or size or type or color

```
{% for c in categories %}
  <li>
    <input class="form-check-input filter-checkbox"  type="checkbox" name="checkbox"
data-filter="category" id="exampleCheckbox2" value="{{ c.id }}" />

   <a href="shop-grid-right.html"> <img src="{{c.image.url}}" alt="" />{{c.title}}</a>
   </li>
{% endfor %}


{% for v in vendors %}
   <input class="form-check-input filter-checkbox" data-filter="vendor" type="checkbox"
name="checkbox" id="exampleCheckbox1" value="{{ v.id }}" />
    <label class="form-check-label" for="exampleCheckbox1"><span>{{v.title}}</span></label>
    <br />
{% endfor %}
```

- Collect the data and create the object that would be sent to the server using jquery ajax.

```
$(document).ready(function(){
    $(".loader").hide()

    $(".filter-checkbox").on("click", function(){
        let filter_object = {}

        $(".filter-checkbox").each(function(index){
            let filter_value = $(this).val()
            let filter_key = $(this).data("filter")
            console.log(filter_value, filter_key);
            filter_object[filter_key] =
Array.from(document.querySelectorAll('input[data-filter='+filter_key+']:checked')).map(function
(element){
            return element.value
            })
        })
       console.log(filter_object);
     })
  })
```

- In the products forloop in product-list.html add an id=””filtered-product” in the div rapping the forloop

```
<div class="row product-grid" id="filteredProduct">
```

```
    <div class="loader">
        <img src="https://miro.medium.com/max/1400/1*CsJ05WEGfunYMLGfsT2sXA.gif" alt="">
    </div>
    {% for p in products %}
        <div class="col-lg-1 ...
        ...
```

- Grab the selected checkbox in the server and perform an operation

```
$.ajax({
     url: '/filter-product',
     data: filter_object,
     dataType: 'json',
     beforeSend: function(){
         $(".loader").show()
      },
     success: function(res){
          console.log(res);
         $(".loader").show()
   }
 1})
```

- Create a view to filter the product in views.py

```
def filter_products(request):\
    return JsonResponse({'data' : "filtering product..."})
```

- Create url in urls.py file

```
path('filter-product/',views.filter_products, name='filter-product'),
```

- Get all data in the filter_product views

```
def filter_products(request):
    categories = request.GET.getlist('category[]')
    vendors = request.GET.getlist('vendor[]')
    products = Product.objects.all().order_by('-id').distinct()
    if len(categories) > 0:
        products = products.filter(category__id__in=categories).distinct()
    if len(vendors) > 0:
        products = products.filter(vendor__id__in = vendors).distinct()


    n = render_to_string('core/async/product-list.html', {'products':products})
       return JsonResponse({'data' : n})
```

- Create a new folder in core called async then a new template in async called product-list.html and copy only the loop of product and paste here, we would now replace this new filtered product in the main product list using filtered-product id.

```
{% for p in products %}
  . . .
{% endfor %}
```

22. Filter Product By Price

- Create Range Input in template

```
 <input type="range" id="range" class="slider-range" min="0" max="100"
oninput="max_price.value=this.value">
```

- Create number input to display the range input value and also onkeyup to auto slide range when we add price manually

```
 <input type="number" id="max_price" class="form-check-ifnput" min="0" max="100" value=""
placeholder="Current Price" type="number" onkeyup="range.value=this.value" name="checkbox"
id="max_price" value="" />
```

- Getting Minimum and Maximum Price of all Products
  We are adding this in the context processor so we can be able to access it in all out template files.

```
from django.db.models import Count, Min, Max
def default(request):
    ...
    'min_max_price'= Product.objects.aggregate(Min("price"), Max("price"))
        context = {
          ...,
        'min_max_price': 'min_max_price',
```

```
        }
        return context
```

- We need to no display the max_min_price in out template accordingly
  First try **{{ min_max_price }}**

```
<input type="range" id="range" class="slider-range" min="{{ min_max_price.price__min }}"
max="{{ min_max_price.price__max }}" oninput="max_price.value=this.value">
```

- Also set the min and max value for the number input

```
<input id="max_price" class="form-check-ifnput" min="{{ min_max_price.price__min }}"
max="{{ min_max_price.price__max }}" value="" placeholder="Current Price" type="number"
onkeyup="range.value=this.value" name="checkbox" id="max_price" >
```

- Add the **from:** min_price **to:** max_price

```
  <div class="caption">From: <strong id="slider-range-value1" class="text-brand">{{
min_max_price.price__min|floatformat:2 }}</strong></div>

  <div class="caption">To: <strong id="slider-range-value2" class="text-brand">{{
min_max_price.price__max|floatformat:2 }}</strong></div>
```

- Add Filter Button Belo number input

```
  <button class="btn mt-20 w-100">Filter By Price</button>
```

- Add Default value for number input

```
 <input id="max_price" class="form-check-ifnput" min="{{ min_max_price.price__min }}" max="{{
min_max_price.price__max }}" value="Minimum: {{ min_max_price.price__min|floatformat:2 }}"
placeholder="Current Price" type="number" onkeyup="range.value=this.value" name="checkbox"
id="max_price" />
```

- Also Set the range value to the minimum price

```
<input type="range" id="range" class="slider-range" min="{{
min_max_price.price__min }}" max="{{ min_max_price.price__max }}"
oninput="max_price.value=this.value">
```

- Now we need to use the blur function in javascript to check if the value that the user is entering in the number input is not less than or greater than the min_max_price of all products.

```
<!-- First Add id=”max_price” for the number input →
 <input id="max_price" type="number"  . . ./>

<!-- First Add id=”max_price” for the number input →
 <input id="range type="range"  . . ./>
```

- Let’s write the javascript code now...

23. Filter Product By Price part 2

- Add id to the filter button

```
 <button id="price-filter-btn" type="button" class="btn mt-20 w-100">Filter
By Price</button>
```

- We need to send the min price and the max price to the server in the filter_product view in views.py

```
    min_price = request.GET['min_price']
    max_price = request.GET['max_price']
    ...
    products = products.filter(price__gte=min_price)
    products = products.filter(price__lte=max_price)
```

- Javascript to send the min max to the server

```
let min_price = $("#max_price").attr("min")
let max_price = $("#max_price").val()

filter_object.min_price = min_price;
filter_object.max_price = max_price;
```

- Add another selector to the lists

```
 $(".filter-checkbox, #priceFilterBtn").on("click", function(){
```

24. Add to cart part

- In the product detail page add an input field with product id and title and also add id to the cart button and also add classes to the input fields, add id to the quantity input field.

```
 <input type="hidden" class="product-id" value="{{ product.id }}" name="" id="">

<input type="hidden" class="product-title" value="{{ product.title }}" name="" id="">

<input type="number" value="1" name="" id="product-quantity" class="w-25 mb-20">

<button type="submit" id="add-to-cart-btn" class="button button-add-to-cart"><i
class="fi-rs-shopping-cart"></i>Add to cart</button>
```

- In js file, add new code to create add to cart functionality

```
$("#add-to-cart-btn").on("click", function(){
        let quantity = $("#product-quantity").val()
        let product_title = $(".product-title").val()
        let product_id = $(".product-id").val()
        let product_price = $(".current-product-price").text()
        let this_val = $(this)

        console.log("Quantity:", quantity);
        console.log("Id:", product_id);
        console.log("Title:", product_title)
        console.log("Price:", product_price)
        console.log("This is:", this_val)

        $.ajax({
            url:'/add-to-cart',
            data:{
                'id':product_id,
                'qty':quantity,
                'title':product_title,

               'price':product_price
            },
            dataType:'json',
            beforeSend:function(){
                this_val.html("Added To Cart")
            },
```

```
            success:function(response){
                console.log(response);
                $(".cart-items-count").text(response.totalcartitems)
                this_val.attr('disabled',false);
            }
        });
    })

})
```

- Write a new add_to_cart function in views.py

```
def add_to_cart(request):
    # Define a new variable cart_product
    # assign it to an empty dictionary
    cart_p = {}

    # get the current product id with this line
    cart_p[str(request.GET['id'])] = {
        #  Get all the data which are title, quantity and price
        'title': request.GET['title'],
        'qty': request.GET['qty'],
        'price': request.GET['price']
    }
    return JsonResponse({'data': cart_p})
```

- Configure the add_to_cart function in urls.py

```
    path("add-to-cart", views.add_to_cart, name="add-to-cart"),
```

- Complete the add_to_cart functionality

```
def add_to_cart(request):
    # del request.session['cartdata']
    cart_p = {}
```

```
    cart_p[str(request.GET['id'])] = {
        'title': request.GET['title'],
        'qty': request.GET['qty'],
        'price': request.GET['price']
    }
    if 'cart_data_obj' in request.session:
        if str(request.GET['id']) in request.session['cart_data_obj']:
            cart_data = request.session['cart_data_obj']
            cart_data[str(request.GET['id'])]['qty'] =
int(cart_p[str(request.GET['id'])]['qty'])
            cart_data.update(cart_data)
            request.session['cart_data_obj'] = cart_data
        else:
            cart_data = request.session['cart_data_obj']
            cart_data.update(cart_p)
            request.session['cart_data_obj'] = cart_data
    else:
        request.session['cart_data_obj'] = cart_p
    return JsonResponse({'data': request.session['cart_data_obj'], 'totalcartitems':
len(request.session['cart_data_obj'])})
```

- Add session data in base.html cart count

```
<span class="pro-count blue cart-items-count">{{ request.session.cart_data_obj|length }}</span>
```

- Add This code to get updated cart data length in real time

```
success:function(response){
    console.log(response);
    $(".cart-items-count").text(response.totalcartitems)
     this_val.attr('disabled',false);
   }
```

25. Add to cart from all Pages and Loops

- Append the product id to all hidden and text input fields classes in index.html, product-list.html and product-detail.html

```
///// Index.html and PRoduct-list.html
<div class="add-cart">
    <input type="hidden" value="1" name="" id="product-quantity" class="w-25 mb-20
product-quantity-{{p.id}}">
    <input type="hidden" name="" id="" class="product-pid-{{p.id}}" value="{{p.pid}}">
    <input type="hidden" name="" id="" class="product-image-{{p.id}}" value="{{p.image.url}}">
    <input type="hidden" class="product-id-{{p.id}}" value="{{ p.id }}" name="" id="">
    <input type="hidden" class="product-title-{{p.id}}" value="{{ p.title }}" name="" id="">
    <button class="add add-to-cart-btn b-none" data-index="{{ p.id }}"><i
class="fi-rs-shopping-cart mr-5" id="add-to-cart-btn"></i>Add </button>
</div>
```

```
////// PRoduct-detail.html

<span class="current-price current-product-price-{{ product.id }}
text-brand">{{product.price}}</span>

<input type="hidden" name="" id="" class="product-pid-{{product.id}}" value="{{product.pid}}">
<input type="hidden" name="" id="" class="product-image-{{product.id}}"
value="{{product.image.url}}">
<input type="hidden" class="product-id-{{product.id}}" value="{{ product.id }}" name="" id="">
<input type="hidden" class="product-title-{{product.id}}" value="{{ product.title }}" name=""
id="">
<input type="number" value="1" name="" id="product-quantity" class="w-25 mb-20
product-quantity-{{product.id}}">
<button type="submit" id="add-to-cart-btn" class="button button-add-to-cart add-to-cart-btn"
data-index="{{ product.id }}"><i class="fi-rs-shopping-cart"></i>Add to cart</button>
```

- Add class `add-to-cart-btn` to all add to cart buttons
- Update the add to cart ajax code

```
    // Adding product to cart
    $(".add-to-cart-btn").on("click", function(){
      let this_val = $(this)
      let index = this_val.attr("data-index")
      let quantity = $(".product-quantity-"+_index).val()
      let product_title = $(".product-title-"+_index).val()
      let product_image = $(".product-image-"+_index).val()
      let product_pid = $(".product-pid-"+_index).val()
      let product_id = $(".product-id-"+_index).val()
      let product_price = $(".current-product-price-"+_index).text()
```

```
beforeSend:function(){
    this_val.html("✓")
 },
```

- Add the same classes for shop.html, product-category-list.html

26. Cart page and listing all products in cart

- Create new cart_view in views.py
- Create URLs and Template then View in Broser
- We need to cart image in the data we are sending to the cart

```
<input type="hidden" name="" id="" class="product-image" value="{{product.image.url}}">
```

- Grab the image value in js

```
 let product_image = $(".product-image").val()
let product_pid = $(".product-pid").val()


'Image':product_image,
'pid':product_pid,
```

- Also get the image in server

```
...
'image': request.GET['image'],
'pid': request.GET['pid'],
```

- Pass in the cart data in the render function

```
 return render(request, "core/cart.html", {'cart_data':
request.session['cart_data_obj'], 'totalcartitems':
len(request.session['cart_data_obj'])})
```

- Call the `'totalcartitems'` in the template

```
<h6 class="text-body">There are <span class="text-brand">{{totalcartitems}}</span>
product(s) in your cart</h6>
```

- Loop through the data in the cart onto the cart page

```
 {% for product_id, item in cart_data.items %}
<td class="image product-thumbnail pt-40"><img src="{{item.image}}" alt="#"></td>
<h6 class="mb-5"><a class="product-name mb-10 text-heading" href="{% url 'core:product-detail'
item.pid %}">{{item.title|truncatechars:40}}</a></h6>
 {% endfor %}
```

- Complete the rest of the data in the cart page like price, qty, image, title, url etc.
- Calculate the subtotal for each product based on quantity using `widthratio`

```
<h4 class="text-brand">${% widthratio item.price 1 item.qty %} </h4>
```

- Calculate the total prize for all the product in the cart

```
def cart_view(request):
    cart_total_amount = 0
    for p_id, item in request.session['cart_data_obj'].items():
        cart_total_amount+= int(item['qty']) * float(item['price'])
    return render(request, "core/cart.html", {..., 'cart_total_amount':cart_total_amount})
```

- Add the cart_total_amount in the template

```
<h4 class="text-brand text-end">${{cart_total_amount}}</h4>
```

27. Deleting product from cart page

- Add Class delete-Item and **data-item={{ product_id }}** in cart-list page

```
<td class="action text-center delete-item" data-item="{{ product_id }}"><a href="#"
class="text-body"><i class="fi-rs-trash"></i></a></td>
```

- Write jquery code to delete items from cart

```
$(document).on("click", ".delete-item", function(){
        let product_id = $(this).attr("data-item")
        let this_val = $(this)
        console.log(product_id);
        $.ajax({
            url:'/delete-from-cart',
            data:{
                'id':product_id,
            },
            dataType:'json',
            beforeSend:function(){
                // this_val.html("Added To Cart")
                this_val.attr('disabled',true);
```

```
            },
            success:function(response){
                console.log(response);
                $(".cart-items-count").text(response.totalcartitems)
                this_val.attr('disabled',false);
                $("#cartList").html(response.data)
            }
        });
    })
```

- Write a view to delete the item from the cart and also update the session

```
def delete_from_cart_view(request):
    product_id = str(request.GET['id'])
    if 'cart_data_obj' in request.session:
        if product_id in request.session['cart_data_obj']:
            cart_data = request.session['cart_data_obj']
            del request.session['cart_data_obj'][product_id ]
            request.session['cart_data_obj'] = cart_data
    cart_total_amount = 0
    for product_id, item in request.session['cart_data_obj'].items():
        cart_total_amount+= int(item['qty']) * float(item['price'])

    t = render_to_string('core/async/cart-list.html',
{'cart_data':request.session['cart_data_obj'],'totalcartitems':len(request.session['cart_data_o
bj']),'cart_total_amount':cart_total_amount})
    return JsonResponse({'data':t,'totalcartitems':len(request.session['cart_data_obj'])})
```

- Create new template in **core/async/cart-list.html**
- Copy and paste everything in the cart.html except from the the jinja codes

28. Updating product from cart page

- Add Class update-Item and **data-item={{ product_id }}** in cart-list page

```
<td class="action text-center" ><button data-item="{{ product_id }}"
style="border: none; background: none; " class="text-body update-item"><i
class="fi-rs-refresh"></i></button></td>
```

- Create vie to update the cart items

```
def update_from_cart_view(request):
    # p_id = request.GET['id']
    product_id = str(request.GET['id'])
```

```
    p_qty=request.GET['qty']

    if 'cart_data_obj' in request.session:
        if product_id in request.session['cart_data_obj']:
            cart_data = request.session['cart_data_obj']
            cart_data[str(request.GET['id'])]['qty'] = p_qty

            request.session['cart_data_obj'] = cart_data

    cart_total_amount = 0
    for product_id, item in request.session['cart_data_obj'].items():
        cart_total_amount+= int(item['qty']) * float(item['price'])

    t = render_to_string('core/async/cart-list.html',
{'cart_data':request.session['cart_data_obj'],'totalcartitems':len(request.session['cart_data_o
bj']),'cart_total_amount':cart_total_amount})
    return JsonResponse({'data':t,'totalcartitems':len(request.session['cart_data_obj'])})
```

- Create ajax jquery code to update the cart

```
    // updating items from cart
    $(document).on("click", ".update-item", function(){
        let product_id = $(this).attr("data-item")
        let product_qty = $(".product-qty-"+product_id).val()
        let this_val = $(this)
        console.log(product_qty);
        $.ajax({
            url:'/update-cart',
            data:{
                'id':product_id,
                'qty':product_qty,
            },
            dataType:'json',
            beforeSend:function(){
                // this_val.html("Added To Cart")
                this_val.attr('disabled',true);
            },
            success:function(response){
                console.log(response);
                $(".cart-items-count").text(response.totalcartitems)
                this_val.attr('disabled',false);
                $("#cartList").html(response.data)
            }
        });
    })
```

- Remove checkout button if there is no item in cart

```
{% if totalcartitems %}
   <a href="#" class="btn mb-20 w-100">Proceed To CheckOut<i class="fi-rs-sign-out ml-15"></i></a>
{% endif %}
```

29. Checkout Page

- Create new view checkout_view in views.py

```
@login_required
def checkout_view(request):
    cart_total_amount = 0
    if 'cart_data_obj' in request.session:
        for p_id, item in request.session['cart_data_obj'].items():
            cart_total_amount += int(item['qty']) * float(item['price'])
        return render(request, 'core/checkout.html',
{'cart_data':request.session['cart_data_obj'],'totalcartitems':len(request.session['cart_data_obj']),'cart_t
otal_amount':cart_total_amount})
```

- Create new URL for the checkout_view

```
path("checkout/", views.checkout_view, name="checkout"),
```

- Add new template checkout.html and setup the template
- Loop thorugh all the product in the session into the checkout.html template
- Calculate subtotal and totals price of all products

```
// Subtotal price
<h4 class="text-brand">${% widthratio item.price 1 item.qty %}</h4>



// Total PRice
<h4 class="text-brand text-end">${{cart_total_amount|floatformat:2}}</h4>
```

- Conclude

30. Paypal Checkout using DJango Paypal

- Install the django paypal package `pip install django-paypal`
- Install the paypal package in the settings.py INSTALLED_APPS Section "paypal.standard.ipn"

- Also Add the paypal.standard.ipn url

```
    path('paypal/', include('paypal.standard.ipn.urls')),
```

- Now add the payal email and test status in setttings.py

```
PAYPAL_RECEIVER_EMAIL = 'sb-c5xgx6555500@business.example.com'
PAYPAL_TEST = True
```

- Let's create the payment-successfull page and payment-failed page and also configure url

```
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def payment_completed_view(request):
    context=request.POST
    return render(request, 'core/payment-completed.html', {'context':context})

@csrf_exempt
def payment_failed_view(request):
    return render(request, 'core/payment-failed.html')
```

- Configure URL

```
    path("payment-completed/", views.payment_completed_view, name="payment-completed"),

    path("payment-failed/", views.payment_failed_view, name="payment-failed"),
```

- Create vie to process the checkout using dummy data in checkout_view

```
from django.urls import reverse
from django.conf import settings
from django.views.decorators.csrf import csrf_exempt
from paypal.standard.forms import PayPalPaymentsForm
```

```
@login_required
def checkout_view(request):
    host = request.get_host()
    paypal_dict = {
        'business': settings.PAYPAL_RECEIVER_EMAIL,
        'amount': "200",
        'item_name': 'Fresh Pear',
        'invoice': 'INV-200',
        'currency_code': 'USD',
        'notify_url': 'http://{}{}'.format(host,reverse('paypal-ipn')),
        'return_url': 'http://{}{}'.format(host,reverse('core:payment_done')),
        'cancel_return': 'http://{}{}'.format(host,reverse('core:payment_cancelled')),
    }
    # Form to render the paypal button
    payment_button_form = PayPalPaymentsForm(initial=paypal_dict)

    print("Host is ###########3", request.get_host())
    cart_total_amount = 0
    if 'cart_data_obj' in request.session:
        for p_id, item in request.session['cart_data_obj'].items():
            # print("Item PRice is ##################", item['price'])
            cart_total_amount += int(item['qty']) * float(item['price'])

        return render(request, 'core/checkout.html',
{'cart_data':request.session['cart_data_obj'],'totalcartitems':len(request.session['cart_data_o
bj']),'cart_total_amount':cart_total_amount, 'payment_button_form':payment_button_form})
```

- Render the button in the checkout.html template {{ `payment_button_form.render` }}

31. Get all Paypal Processing Items Dynamically and Save All Cart Items to Database and

- Re-code the checkout page function to get data dynamically and also save item to db

```
@login_required
def checkout_view(request):

    cart_total_amount = 0
    total_amount = 0
    if 'cart_data_obj' in request.session:
        ############ Create Order ################
        order = CartOrder.objects.create(
```

```
            user=request.user,
            total_amount=total_amount,
        )
        ############ End Create Order ################

        for p_id, item in request.session['cart_data_obj'].items():
            cart_total_amount += int(item['qty']) * float(item['price'])

            ############ Create Order Items ################
            items = CartOrderItems.objects.create(
                order=order,
                invoice_no='INV-'+str(order.id),
                item=item['title'],
                image=item['image'],
                qty=item['qty'],
                price=item['price'],
                total=float(item['qty']) * float(item['price'])
            )

        # Payment
        host = request.get_host()
        paypal_dict = {
        'business': settings.PAYPAL_RECEIVER_EMAIL,
        'amount': cart_total_amount,
        'item_name': 'OrderNo-'+str(order.id),
        'invoice': 'INV-'+str(order.id),
        'currency_code': 'USD',
        'notify_url': 'http://{}{}'.format(host,reverse('core:paypal-ipn')),
        'return_url': 'http://{}{}'.format(host,reverse('core:payment-completed')),
        'cancel_return': 'http://{}{}'.format(host,reverse('core:payment-failed')),
        }
        # Form to render the paypal button
        payment_button_form = PayPalPaymentsForm(initial=paypal_dict)

        return render(request, 'core/checkout.html',
{'cart_data':request.session['cart_data_obj'],'totalcartitems':len(request.session['cart_data_o
bj']),'cart_total_amount':cart_total_amount, 'payment_button_form':payment_button_form})
```

32. Create Invoice Page and Get All Recently ORdered Products

- Create vie

```
@csrf_exempt
def payment_completed_view(request):
    cart_total_amount = 0
    for product_id, item in request.session['cart_data_obj'].items():
        cart_total_amount+= int(item['qty']) * float(item['price'])
    context=request.POST
    return render(request, 'core/payment-completed.html',
{'cart_data':request.session['cart_data_obj'],'totalcartitems':len(request.session['cart_data_obj']),'cart_t
otal_amount':cart_total_amount})
```

- Loop through the order

```
{% for product_id, item in cart_data.items %}
    <tr>
   <td>
      <div class="item-desc-1">
     <span>{{item.title}}</span>
       </div>
       </td>
        <td class="text-center"><img style="height: 70px; width: 70px; border-radius: 20px; object-fit:
cover;" src="{{item.image}}" alt=""></td>
        <td class="text-center">${{item.price}}</td>
        <td class="text-center">{{ item.qty }}</td>
         <td class="text-right">${% widthratio item.price 1 item.qty %}</td>
      </tr>
     {% endfor %}
```

33. Customer Dashboard

- Create View to display user dashboard

```
@login_required
def c(request):
    orders = CartOrder.objects.filter(user=request.user).order_by("-id")
    context = {
        "orders": orders
    }
    return render(request, 'core/dashboard.html', context)
```

- Configure URL and template

```
<tbody>
   {% for o in orders %}
   <tr>
```

```
      <td>ID:{{o.id}}</td>
      <td>{{o.order_dt}}</td>
      <td>{{o.order_status|title}}</td>
      <td>${{o.total_amount}}</td>
      <td><a href="{% url 'core:order-detail' o.id %}" class="btn-small d-block">View</a></td>
   </tr>
  {% endfor %}
</tbody>
```

- Create Order Detail also

```
@login_required
def order_detail(request, id):
    order = CartOrder.objects.get(user=request.user, id=id)
    order_items = CartOrderItems.objects.filter(order=order)
    context = {
        "order_items": order_items
    }
    return render(request, 'core/order_details.html', context)
```

- Create URL

```
    path("dashboard/", views.Dashboard, name="dashboard"),
    path("dashboard/order/<int:id>", views.order_detail, name="order-detail"),
```

34. User Address

- Add Query to Dashboard View to display all addresses

```
    address = AddressBook.objects.filter(user=request.user)
```

- Create function in dashboard view to save address

```
if request.method == "POST":
        address = request.POST.get("address")
        phone = request.POST.get("phone")

        nAddress = AddressBook.objects.create(
            user=request.user,
            address=address,
            mobile=phone
        )
        messages.success(request, "Address Saved")
        return redirect("core:dashboard")
```

- Create template and form

```
<form class="mb-4" method="POST" > {% csrf_token %}
     <div class="card-hdeader">
      <h5>Add Address</h5>
     </div>
      <div class="row">
      <div class="form-group col-md-6">
     <input placeholder="Address" required="" class="form-control" name="address" type="text" />
     </div>
     <div class="form-group col-md-6">
     <input placeholder="Phone" required="" class="form-control" name="phone" />
      </div>
    <div class="col-md-12">
      <button type="submit" class="btn btn-fill-out submit font-weight-bold" name="submit"
value="Submit">Save Address</button>
    </div>
    </div>
     </form>
```

- List out all Addresses

35. Make Default Address for Orders

- Create View to make an address default and remove other default addresses

```
def make_address_default(request):
    id = str(request.GET['id'])
    AddressBook.objects.update(status=False)
    AddressBook.objects.filter(id = id).update(status=True)

    return JsonResponse({'boolean':True})
```

- Configure URL

```
 path('make-default-address',views.make_address_default, name='make-default-address'),
```

- Setup Template to display selected and activated address

```
{% if a.status %}
   <i data-address-id="{{a.id}}" class="fa fa-check-circle text-success check{{a.id}} check""></i>
   <button data-address-id="{{a.id}}" style="display: none;"  class="btn   make-default-address btn{{a.id}} ht_btn">Use
Address</button>
{% else %}
   <i style="display: none;" data-address-id="{{a.id}}" class="fa fa-check-circle text-success check{{a.id}} check""></i>
   <button data-address-id="{{a.id}}" class="btn   make-default-address btn{{a.id}} ht_btn">Use Address</button>
{% endif %}
```

- Write Js Code to make address default

```
$(document).on('click','.make-default-address',function(){
        var id=$(this).attr('data-address-id');
        var this_val=$(this);
        // Ajax
        $.ajax({
            url:'/make-default-address',
            data:{
                'id':id,
            },
            dataType:'json',
            success:function(response){
                if(response.boolean==true){

                    $(".check").hide();
                    $(".ht_btn").show();

                    $(".check"+id).show();
                    $(".btn"+id).hide();
                }
            }
        });
    });
```

- Show default address in checkout page

```
try:
    active_address = AddressBook.objects.get(user=request.user, status=True)
except:
    messages.warning(request, "There are multiple address, only one should be activated.")
   active_address = None
```

- Show Address in Template

36. Wishlist List Page

- Create View to list out all products in wishlist

```
def WishlistPage(request):
    try:
        wishlist = Wishlist.objects.filter(user=request.user)
    except:
        wishlist = None
    context = {
        "w": wishlist
    }
    return render(request, 'core/wishlist.html', context)
```

- Create URL

```
    path('wishlist/',views.WishlistPage, name='wishlist'),
```

- Configure template and loop through products

37. Adding Products to wishlist

- Create View to add product to list

```
def add_to_wishlist(request):
    product_id = request.GET['product_id']
    product = Product.objects.get(id=product_id)
    context = {}
    wishlist_count = Wishlist.objects.filter(product=product, user=request.user).count()
    if wishlist_count > 0:
        context = {
            'bool':False
        }
    else:
        new_wishlist = Wishlist.objects.create(
            product=product,
            user=request.user
        )
        context = {
            'bool':True
        }
    return JsonResponse(context)
```

- Create URL

```
    path('add-to-wishlist',views.add_to_wishlist, name='add-to-wishlist'),
```

- Configure template index html, product-list.html and detail page

```
<button style="border: none; background: none;" class="add-to-wishlist" data-product-item="{{p.id}}"><i
class="fi-rs-heart" style="fill: aqua;"></i></button>
```

- Create function in ajax

```
    $(document).on('click',".add-to-wishlist",function(){
        var product_id = $(this).attr('data-product-item');
        var label = $(this).attr("aria-label")
        var this_val = $(this);

        console.log(label);
```

```
    $.ajax({
        url:"/add-to-wishlist",
        data:{
            product_id : product_id
        },
        dataType:'json',
        beforeSend: function(){
            this_val.html("✓")

        },
        success:function(response){
            if(response.bool == true){
                console.log("Added...");
            }
        }
    });
});
```

38. Remove from wishlist

- Create View to remove products from the wishlist

```
from django.core import serializers

def RemoveWishlist(request):
    pid = request.GET['id']
    wishlist = Wishlist.objects.filter(user=request.user).count()
    wishlist_d = Wishlist.objects.get(id=pid)
    delete_product = wishlist_d.delete()
    context = {
        "bool":True,
        "wishlist":wishlist
    }
    qs_json = serializers.serialize('json', wishlist)
    t = render_to_string('core/async/wishlist-list.html', context)
    return JsonResponse({'data':t,'wishlist':qs_json})
```

- Configure URL

```
    path('remove-wishlist/',views.RemoveWishlist, name='remove-wishlist'),
```

- Create a new template “wishlist-lists.html” in **async/core/...**

- Copy everything in wishlist.html and add in new template, then remove template tags
- Add id=”wishlist-list” to wishlist.html main tag

```
    <main class="main" id="wishlist-list">
```

- Add class to delete button to grab id for ajax

```
<button style="border: none;" data-wishlist-product="{{ w.id }}"
class="text-body delete-wishlist-product"><i class="fi-rs-trash"></i></a>
```

- Create function using ajax to remove the product

```
    $(document).on("click", ".delete-wishlist-product", function(){
       let this_val = $(this)
       let product_id = $(this).attr("data-wishlist-product")
       console.log(this_val);
       console.log(product_id);
       $.ajax({
           url: "/remove-wishlist",
           data: {
               "id": product_id
           },
           dataType: "json",
           beforeSend: function(){
               console.log("Deleting...");
           },
           success: function(res){
               $("#wishlist-list").html(res.data)
           }

       })
   })
```

39. Chart Using Chart.Js

- Create add order and month code

```
def Dashboard(request):

orders =
CartOrder.objects.annotate(month=ExtractMonth("order_date")).values("month").annotate
(count=Count("id")).values("month", "count")
    month= []
    total_orders= []
```

```
    for d in orders:
        month.append( calendar.month_name[d['month']] )
        total_orders.append(d['count'])
```

- Add Chart.Js HTML Element to dashboard.html

```
<div>
 <canvas id="myChart" style="height: 50px;"></canvas>
</div>
```

- Add Js Code to initiate the chart feature

```
<script>
    const labels = {{ month|safe }}

    const data = {
      labels: labels,
      datasets: [{
        label: 'Orders',
        backgroundColor: 'rgb(59,183,126)',
        borderColor: 'rgb(25, 99, 132)',
        data: {{ total_orders|safe }}
      }]
    };

    const config = {
      type: 'bar',
      data: data,
      options: {}
    };

    const myChart = new Chart(
        document.getElementById('myChart'),
        config
    );
  </script>
```

-

40. Create profile automatically using Django Signal

- Add signal code to userauths models.py

```
from django.db.models.signals import post_save

def create_user_profile(sender, instance, created, **kwargs):
    if created:
        Profile.objects.create(user=instance)

def save_user_profile(sender, instance, **kwargs):
    instance.profile.save()

post_save.connect(create_user_profile, sender=User)
post_save.connect(save_user_profile, sender=User)
```

CDNS

- **Jquery Ajax CDN**
  <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>

- **Django CKEditor Docs**
  https://django-ckeditor.readthedocs.io/en/latest/#

- **Chart.Js CDN**
  https://www.chartjs.org/docs/latest/getting-started/

CSS Code

```
<style>
    input[type="range"] {
    -webkit-appearance: none;
    margin-right: 15px;
    height: 7px;
    background: rgba(255, 255, 255, 0.6);
    border-radius: 5px;
    background-image: linear-gradient(#3bb77e, #3bb77e);
    background-size: 100% 100%;
    background-repeat: no-repeat;
    }

    input[type="range"]::-webkit-slider-thumb {
    -webkit-appearance: none;
    height: 20px;
    width: 20px;
    border-radius: 50%;
    background: #3bb77e;
    cursor: ew-resize;
    box-shadow: 0 0 2px 0 #555;
    transition: background .3s ease-in-out;
    }

    input[type=range]::-webkit-slider-runnable-track  {
    -webkit-appearance: none;
    box-shadow: none;
    border: none;
    background: transparent;
    }
  </style>
```

Instagram Clone using React

1. **Setup Project**
    - Npx create-react-app instagram_ui
    - Yarn add react-icons
    -