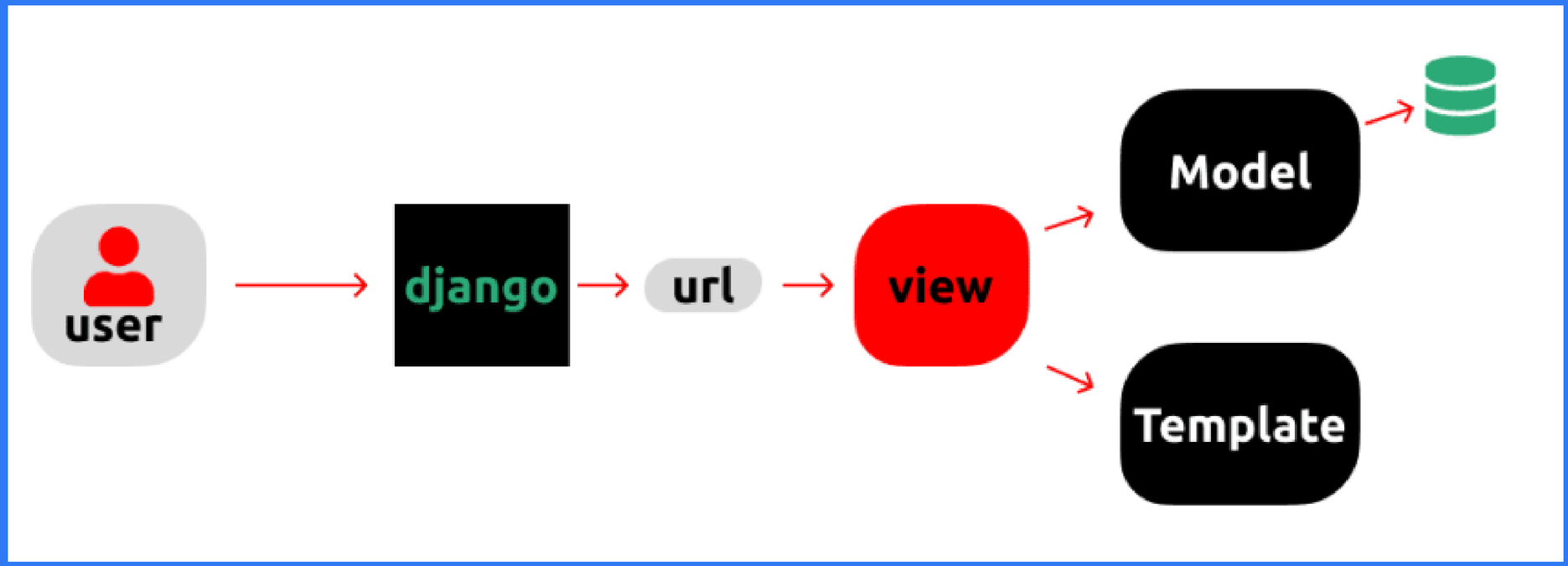
The MVT Architecture Design Pattern

By Tamer Ayoub



Model - View - Template

The Code "Processing an Order"

Template

```
// when user clicks make a payment, initiate the event to run the submitFormData function
document.getElementById('make-payment').addEventListener('click', function(e){
    submitFormData()
})


// when you click submit payment, this function will post users info and shipping info to the processOrder view
Codeium: Refactor | Explain
function submitFormData(){
    console.log('Payment button clicked')

    var userFormData = {
        'name':null,
        'email':null,
        'total':total,
    }

    var shippingInfo = {
        'address':null,
        'city':null,
        'state':null,
        'zipcode':null,
    }

    if (shipping != 'False'){
        shippingInfo.address = form.address.value
        shippingInfo.city = form.city.value
        shippingInfo.state = form.state.value
        shippingInfo.zipcode = form.zipcode.value
    }

    if (user == 'AnonymousUser'){
        userFormData.name = form.name.value
        userFormData.email = form.email.value
    }

    console.log('Shipping Info:', shippingInfo)
    console.log('User Info:', userFormData)

    // this calls fetches the processOrder view and post the userFormData and the shippingInfo
    var url = "/process_order/"
    fetch(url, {
        method:'POST',
        headers:{
            'Content-Type':'applicaiton/json',
            'X-CSRFToken':csrftoken,
        },
        body:JSON.stringify({'form':userFormData, 'shipping':shippingInfo}),

    })
    .then((response) => response.json())
    .then((data) => {
        console.log('Success:', data);
        alert('Transaction completed');

        cart = {}
        document.cookie ='cart=' + JSON.stringify(cart) + ";domain=;path=/"

        window.location.href = "{% url 'store' %}"

        })
```

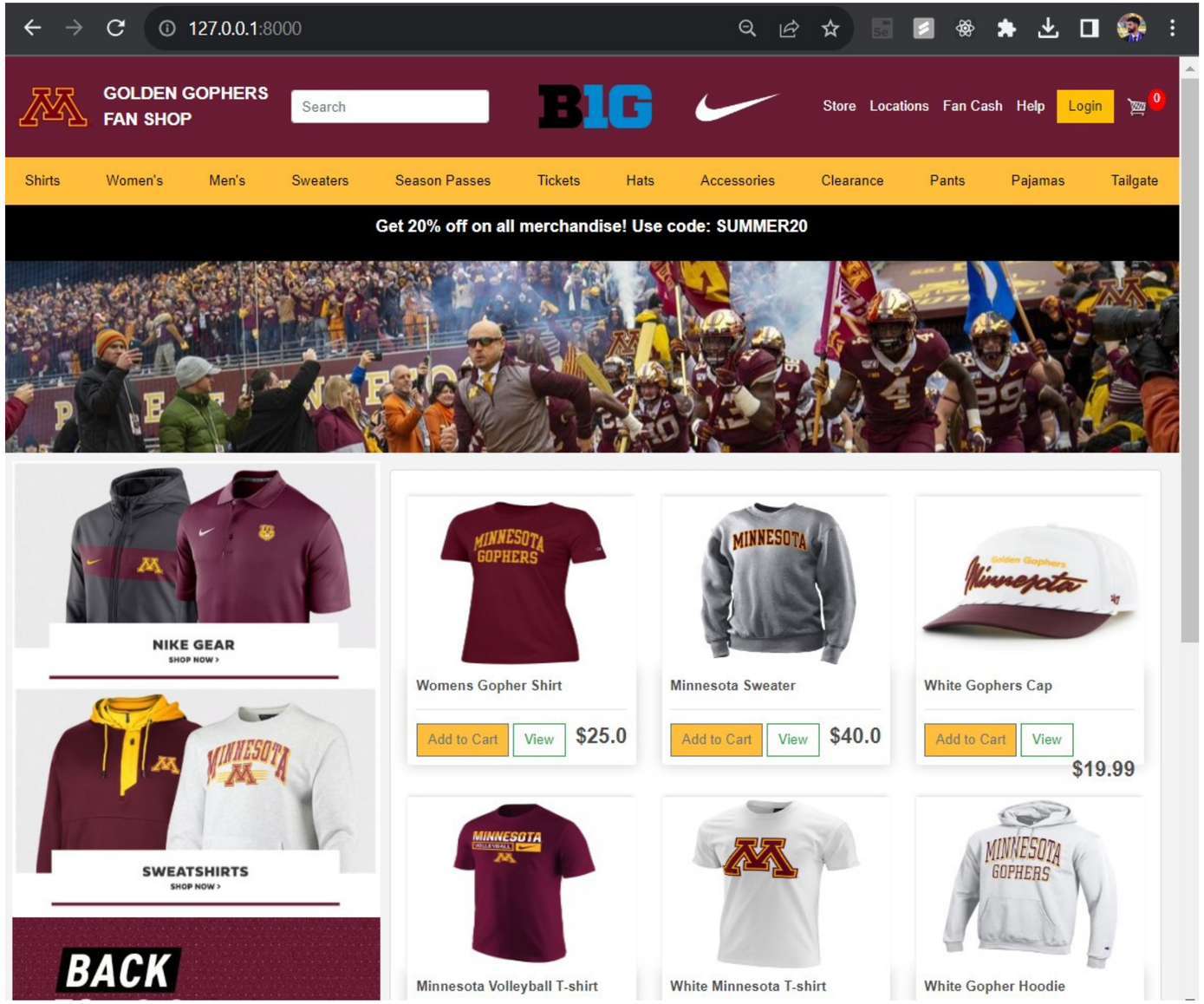
View

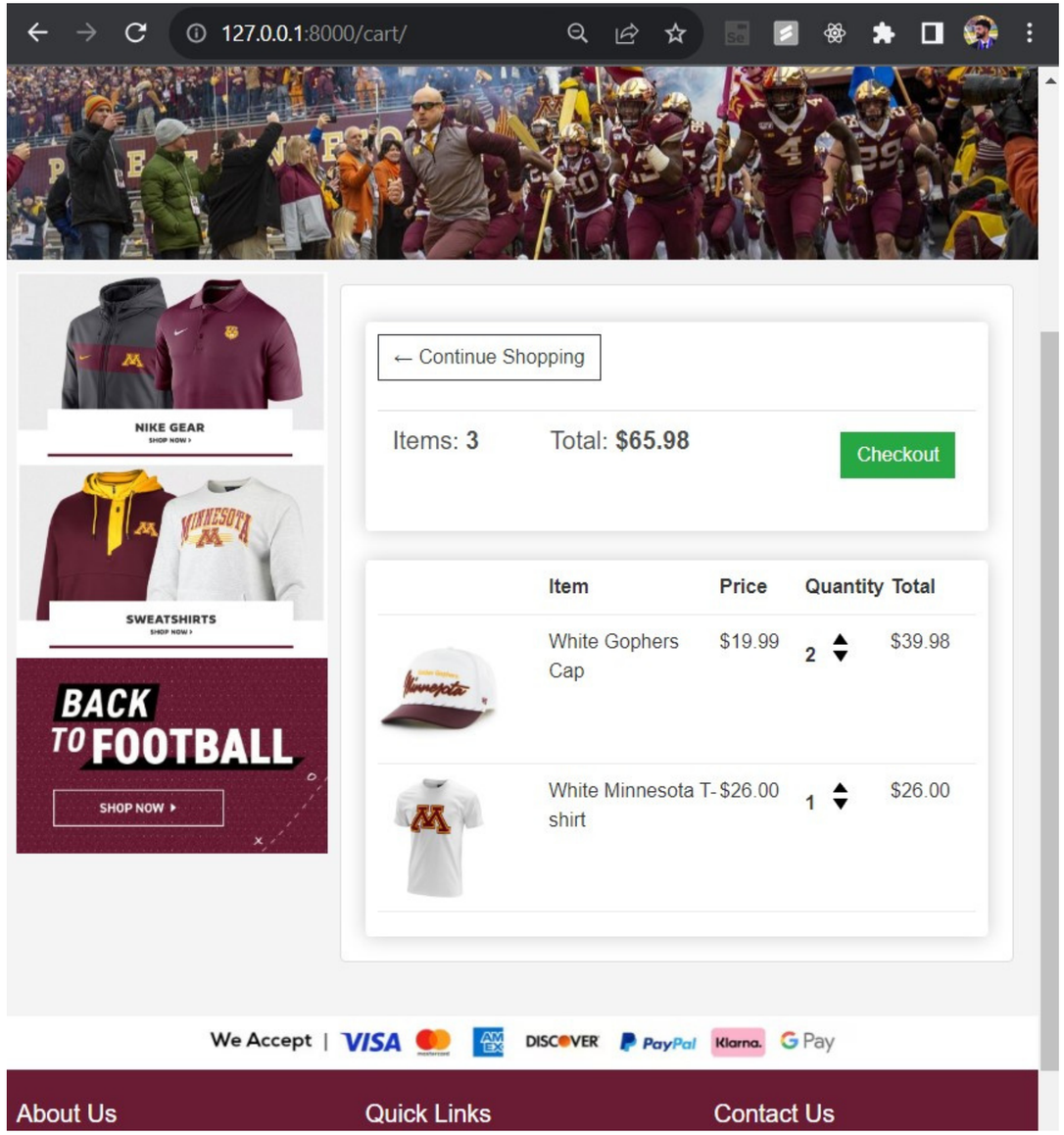
```
Codeium: Refactor | Explain | Generate Docstring
def processOrder(request):
    transaction_id = datetime.datetime.now().timestamp()
    data = json.loads(request.body)

    # if a user has a order, get it, or create it, using customer as a condition ; else use guest
    if request.user.is_authenticated:
        customer = request.user.customer
        order, created = Order.objects.get_or_create(
            customer=customer, complete=False)
    else:
        # The guestOrder function will retur our 'guest' a customer account aswell as an order
        customer, order = guestOrder(request, data)

    total = float(data['form']['total'])
    order.transaction_id = transaction_id

    if total == order.get_cart_total:
        order.complete = True
    order.save()

    if order.shipping == True:
        ShippingAddress.objects.create(
            customer=customer,
            order=order,
            address=data['shipping']['address'],
            city=data['shipping']['city'],
            state=data['shipping']['state'],
            zipcode=data['shipping']['zipcode'],
        )

    return JsonResponse('Payment submitted..', safe=False)
```

Model

```
class ShippingAddress(models.Model):
    customer = models.ForeignKey(
        Customer, on_delete=models.SET_NULL, null=True)
    order = models.ForeignKey(Order, on_delete=models.SET_NULL, null=True)
    address = models.CharField(max_length=200, null=False)
    city = models.CharField(max_length=200, null=False)
    state = models.CharField(max_length=200, null=False)
    zipcode = models.CharField(max_length=200, null=False)
    date_added = models.DateTimeField(auto_now_add=True)

    Codeium: Refactor | Explain | Generate Docstring
    def __str__(self):
        return self.address
```

The Templates





Why is the MVT architecture pattern is so effective?

Similar to the MVC pattern, MVT is a popular pattern that has been used by programmers over the years because of its reusability, organizational, scalable, and flexible effects. The ability to separate the models, templates, and views into their own categories makes it easy for a team of developers to come in and understand the code and its architecture!

In the Minnesotan Gopher Fan Shop project, one of the roles of the MVT pattern was to process orders. From the code in the 2nd slide, the View, similar to what the controller does in MVC, can fetch data, filter data, and format data. It will work to send and get data to the Model and the Template. In our case, the Process Order View is responsible for verifying if a user is authenticated or not. If not, it creates a Guest Account object as well as an Order and Shipping Address object.

These objects and their properties are defined in the Model, where we set up the data structure for our different objects like Guest Account, Orders, and Shipping Address. Having this organized Model of objects and their data structures makes it easy to GET and POST data TO and FROM our database. Looking at the Model in the 2nd slide, you can see how we defined our Shipping Address object and its properties.

Last but not least, there are our Templates. The template is our visual representation for how we display the data to our user. Templates usually use front-end languages like HTML/CSS. The template is great because it lets us separate the HTML structure from the data, which promotes reusability and maintainability.

The MVT architecture design pattern is really interesting to me because it gives us the ability to scale up our project effectively and efficiently. As our project generates more requirements, we have more objects to store, more business logic to handle, and more data to display. The Model View Template gives us an amazing solution to do just that!

Hope you enjoyed my take on MVT and its amazing effects to scale an application effectively and efficiently! Hope you have a great day!

Thank you!