

## 1. Executive Summary

The aim of this work is to develop scalable Django application. During development various techniques and tools were used. Including Redis as a cache, Docker to run image, Nginx as a load balancer. By following outlined objectives we developed functional e-commerce Django application.

## 2. Introduction

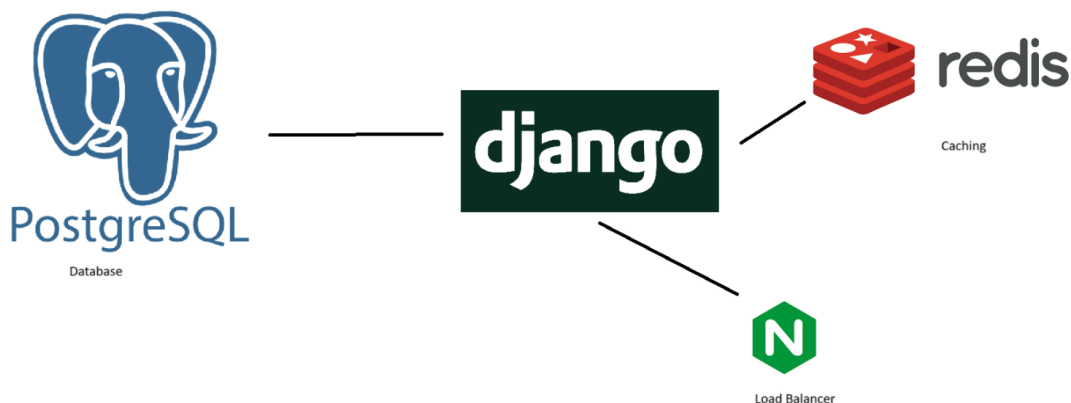
By word high-load systems we typically refer to architectures that were designed to handle a large volume of requests and data. These systems are crucial in today's web applications, especially in e-commerce sector, where user traffic periodically hit peak traffics during shopping periods.

## 3. Project Objectives

Project Objectives:

- Scalability: Design the application to handle a significant increase in user traffic
- Availability: implementing load balancing mechanisms to minimize downtime.
- Throughput: Enable the system to process many requests during peak traffic periods.

## 4. System Architecture



### Client-Side Applications:

- **Web Application:** Django application with templates

### API Layer:

- **Django REST Framework (DRF):** The core of the backend that provides RESTful API endpoints for user authentication, product management, shopping cart functionalities, order processing, and reviews.

- **JWT Authentication:** Secure user authentication mechanism to manage sessions and permissions.

#### Database Layer:

- **PostgreSQL / MySQL:** Database used to store user data, product details, orders, reviews, and categories.
- **Indexing:** Created indexes to speed select queries.

#### Caching Layer:

- **Redis:** In-memory data store used to cache frequently accessed data (to enhance performance and reduce database load).

#### Load Balancer:

- **Nginx:** Load balancing solution that distributes incoming API requests across multiple server instances to ensure high availability and efficient resource utilization.

#### Message Broker:

- **RabbitMQ / Kafka** can be used to handle asynchronous tasks by queuing messages that can be processed in the background.

#### Monitoring and Analytics:

- **Monitoring Tools:** Tools such as Django Debugger and built in logger were used.

## 5. Backend Development

```

1 from rest_framework import serializers
2 from django.contrib.auth.hashers import make_password
3 from .models import User, Category, Product, Cart, Order, Review
4
5 class UserSerializer(serializers.ModelSerializer):
6     class Meta:
7         model = User
8         fields = ['id', 'username', 'email', 'first_name', 'last_name', 'city', 'address']
9
10 class CategorySerializer(serializers.ModelSerializer):
11     class Meta:
12         model = Category
13         fields = ['id', 'name', 'description', 'image']
14
15 class ProductSerializer(serializers.ModelSerializer):
16     category = CategorySerializer()
17
18     class Meta:
19         model = Product
20         fields = ['id', 'name', 'description', 'price', 'stock', 'category', 'image']

```

System check identified no issues (0 silenced).  
October 20, 2024 - 23:28:04  
Django version 5.1.2, using settings 'novashop.settings'  
Starting development server at http://127.0.0.1:8000/  
Quit the server with CTRL-C (or python -c 'import sys; sys.exit(1)')

Our apps structure

```
urlpatterns = [
    path('', views.category_list, name='category_list'),
    path('<int:category_id>/', views.category_detail, name='category_detail'),

    path('products/', views.product_list, name='product_list'),
    path('products/<int:product_id>/', views.product_detail, name='product_detail'),

    path('cart/', views.cart, name='cart'),
    path('cart/add/<int:product_id>/', views.add_to_cart, name='add_to_cart'),

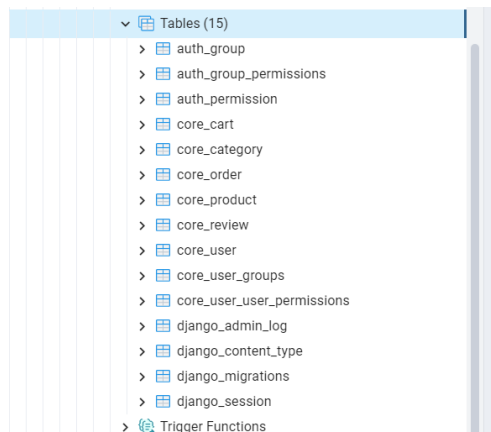
    path('orders/', views.order_list, name='order_list'),
    path('orders/<int:order_id>/', views.order_detail, name='order_detail'),

    path('products/<int:product_id>/reviews/add/', views.add_review, name='add_review'),

    path('api/', include(router.urls)),
    path('api/login/', TokenObtainPairView.as_view(), name='login'),
    path('api/register/', views.register, name='register'),
]
```

## 6. Database Design and Optimization

### 6.1 Schema Design



### 6.2 Query Optimization

We enabled database indexing to speed performance of our app.

```
33 class Product(models.Model):
34     name = models.CharField(max_length=255)
35     description = models.TextField()
36     price = models.DecimalField(max_digits=10, decimal_places=2)
37     stock = models.PositiveIntegerField(default=0)
38     category = models.ForeignKey(Category, related_name='products', on_delete=models.CASCADE)
39     image = models.ImageField(upload_to='products_images')
40
41     class Meta:
42         indexes = [
43             models.Index(fields=['price']),
44             models.Index(fields=['stock']),
45         ]
46
47     def __str__(self) -> str:
48         return self.name
```

```

59 class Order(models.Model):
60     user = models.ForeignKey(User, related_name='orders', on_delete=models.CASCADE)
61     created_at = models.DateTimeField(auto_now_add=True)
62     updated_at = models.DateTimeField(auto_now=True)
63     status = models.CharField(max_length=20, choices=(('Pending', 'Pending'), ('Completed', 'Completed'), ('Canceled', 'Canceled')))
64
65     class Meta:
66         indexes = [
67             models.Index(fields=['status']),
68         ]
69
70     def __str__(self) -> str:
71         return f'Order {self.id}'

```

## 7. Caching Strategies

- Explanation of the caching mechanisms employed, including the rationale for choices made and any challenges encountered.

Cache calls from 1 backend

**Summary**

Total calls	Total time	Cache hits	Cache misses
3	3.404499920297414 ms	0	1

**Commands**

add	get	set	get_or_set	touch	delete	clear	get_many	set_many	delete_many	has_key	incr	decr	incr_version	decr_version
0	1	2	0	0	0	0	0	0	0	0	0	0	0	0

**Calls**

Time (ms)	Type	Arguments	Keyword arguments	Backend
1.4418	get	(views.decorators.cache.cache_header.3f6aa48f4ecc0083b8a44e01cd92840a.en-us.UTC)	{}	<django_redis.cache.RedisCache object at 0x0000024FAD0E6BF0>
1.1796	set	(views.decorators.cache.cache_header.3f6aa48f4ecc0083b8a44e01cd92840a.en-us.UTC', [], 600)	{}	<django_redis.cache.RedisCache object at 0x0000024FAD0E6BF0>
0.7831	set	(views.decorators.cache.cache_page.GET.3f6aa48f4ecc0083b8a44e01cd92840a.d41d8cd98f00b204e9800998ecf8427e.en-us.UTC', <HttpResponse status_code=200, 'text/html; charset=utf-8'>, 600)	{}	<django_redis.cache.RedisCache object at 0x0000024FAD0E6BF0>

**Versions**  
Django 5.1.2

**Time**  
Total: 83.12ms

**Settings**

**Headers**

**Request**  
category\_detail

**SQL**  
2 queries in 3.68ms

**Static files**  
0 files used

**Templates**  
products.html

**Alerts**

**Cache**  
3 calls in 3.40ms

**Signals**  
37 receivers of 15 signals

**Intercept redirects**

**Profiling**

```

@cache_page(600)
def category_detail(request, category_id):
    category = get_object_or_404(Category, pk=category_id)
    products = category.products.all()
    return render(request, 'products.html', {'category': category, 'products': products})

```

```

36 class ProductViewSet(viewsets.ModelViewSet):
37     queryset = Product.objects.all()
38     serializer_class = ProductSerializer
39
40     def list(self, request, *args, **kwargs):
41         cache_key = 'product_list'
42         products = cache.get(cache_key)
43
44         if not products:
45             products = super().list(request, *args, **kwargs).data
46             cache.set(cache_key, products, timeout=60*15)
47
48         return Response(products)
49

```

## 8. Load Balancing Techniques

```

C: > Users > wwwis > nginx > nginx > conf > ⚙️ nginx.conf
17 http {
117     server 127.0.0.1:8001;
118     server 127.0.0.1:8002;
119 }
120
121 server {
122     listen 80;
123
124     server_name your-domain.com;
125
126     location / {
127         proxy_pass http://django_servers;
128         proxy_set_header Host $host;
129         proxy_set_header X-Real-IP $remote_addr;
130         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
131         proxy_set_header X-Forwarded-Proto $scheme;
132     }
133 }
134
135 }

```

Configured Nginx to listen our Django app(Nginx typically has Round Robin implemented)

## 9. Distributed Systems and Data Consistency

## 10. Testing and Quality Assurance

Testing is one of the crucial parts of application development. Integration testing, load testing, and many more testing results can give us idea how our app performs and works.

```

projects > novashop > locustfile.py > UserBehavior > load_product_detail
1  from locust import HttpUser, TaskSet, task, between
2
3  class UserBehavior(TaskSet):
4
5      @task
6      def load_homepage(self):
7          self.client.get("/")
8
9      @task
10     def load_product_list(self):
11         self.client.get("products/")
12
13     @task
14     def load_product_detail(self):
15         self.client.get("products/1/")
16
17
18     class WebsiteUser(HttpUser):
19         tasks = [UserBehavior]
20         wait_time = between(1, 5)

```

Locust file to load test our app.

## 11. Monitoring and Maintenance

By monitoring our app we can get actual information on time. For this we can use built in logger and other tools like Django Debug.

```

168     LOGGING = {
169         'version': 1,
170         'disable_existing_loggers': False,
171         'handlers': {
172             'file': {
173                 'level': 'DEBUG',
174                 'class': 'logging.FileHandler',
175                 'filename': 'django_debug.log',
176             },
177         },
178         'loggers': {
179             'django': {
180                 'handlers': ['file'],
181                 'level': 'DEBUG',
182                 'propagate': True,
183             },
184         },

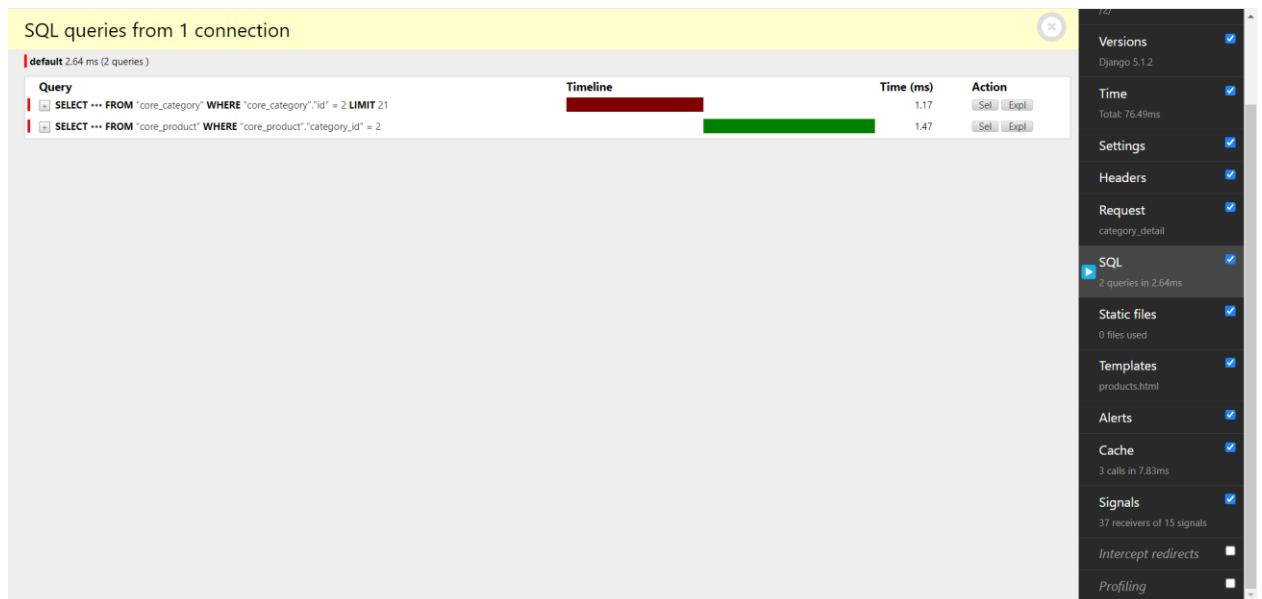
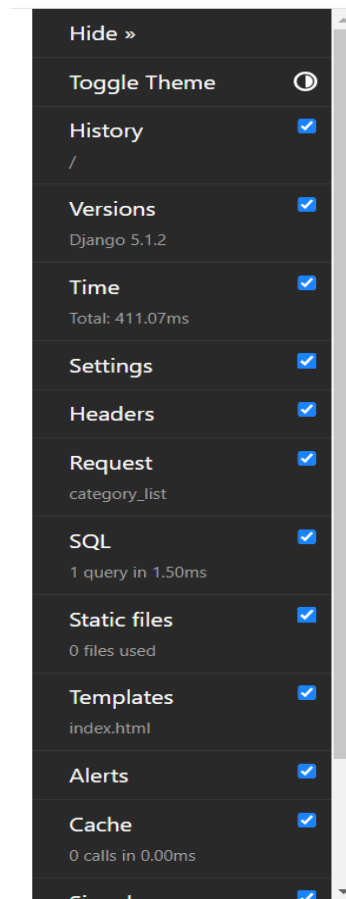
```

### Built in logger

```

projects > novashop >  django_debug.log
1  Watching for file changes with StatReloader
2  Waiting for apps ready_event.
3  Apps ready_event triggered. Sending autoreload_started signal.
4  Watching dir C:\Users\wwwis\Documents\highload\projects\.venv\lib\site-packages\debug_toolbar\templates with glob **/*.
5  Watching dir C:\Users\wwwis\Documents\highload\projects\novashop\core\templates with glob **/*.
6  Watching dir C:\Users\wwwis\Documents\highload\projects\.venv\lib\site-packages\rest_framework\templates with glob **/*.
7  Watching dir C:\Users\wwwis\Documents\highload\projects\novashop\locale with glob **/*.mo.
8  Watching dir C:\Users\wwwis\Documents\highload\projects\.venv\lib\site-packages\debug_toolbar\locale with glob **/*.
9  Watching dir C:\Users\wwwis\Documents\highload\projects\.venv\lib\site-packages\rest_framework\locale with glob **/*.
10 Watching dir C:\Users\wwwis\Documents\highload\projects\.venv\lib\site-packages\rest_framework_simplejwt\locale with glob **/*.
11 Watching dir C:\Users\wwwis\Documents\highload\projects\novashop\core\locale with glob **/*.mo.
12 File C:\Users\wwwis\AppData\Local\Programs\Python\Python310\Lib\http\server.py first seen with mtime 1642407912.0
13 File C:\Users\wwwis\Documents\highload\projects\.venv\Lib\site-packages\django\db\models\constraints.py first seen with mtime 1642407912.0
14 File C:\Users\wwwis\AppData\Local\Programs\Python\Python310\Lib\zoneinfo\tzpath.py first seen with mtime 1642407912.0
15 File C:\Users\wwwis\Documents\highload\projects\.venv\Lib\site-packages\django\contrib\postgres\search.py first seen with mtime 1642407912.0
16 File C:\Users\wwwis\Documents\highload\projects\.venv\Lib\site-packages\urllib3\util\url.py first seen with mtime 1642407912.0
17 File C:\Users\wwwis\Documents\highload\projects\.venv\Lib\site-packages\debug_toolbar\panels\sql\panel.py first seen with mtime 1642407912.0
18 File C:\Users\wwwis\AppData\Local\Programs\Python\Python310\Lib\os.py first seen with mtime 1642407912.0
19 File C:\Users\wwwis\AppData\Local\Programs\Python\Python310\Lib\codecs.py first seen with mtime 1642407912.0
20 File C:\Users\wwwis\AppData\Local\Programs\Python\Python310\Lib\xml\dom\__init__.py first seen with mtime 1642407912.0

```



## Django Debug

### 12. Challenges and Solutions

During development of this app several challenges has occurred, like JWT implementation, load balancer, caching etc. But with proper documentation, lecture materials we were able to overcome them.

### 13. Conclusion



In summary, the project has successfully developed a high-performing e-commerce platform that leverages advanced technologies and best practices. By implementing a robust architecture with efficient API design, comprehensive authentication, and effective caching strategies, the application is well-equipped to handle significant traffic and user demands. The focus on distributed systems and data consistency ensures reliability and responsiveness, enhancing the overall user experience.