Backend for high loaded environment midterm

Student: Islam Aip

Faculty: School of Information Technology and Engineering

Specialty: Computer Systems and Software

## 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

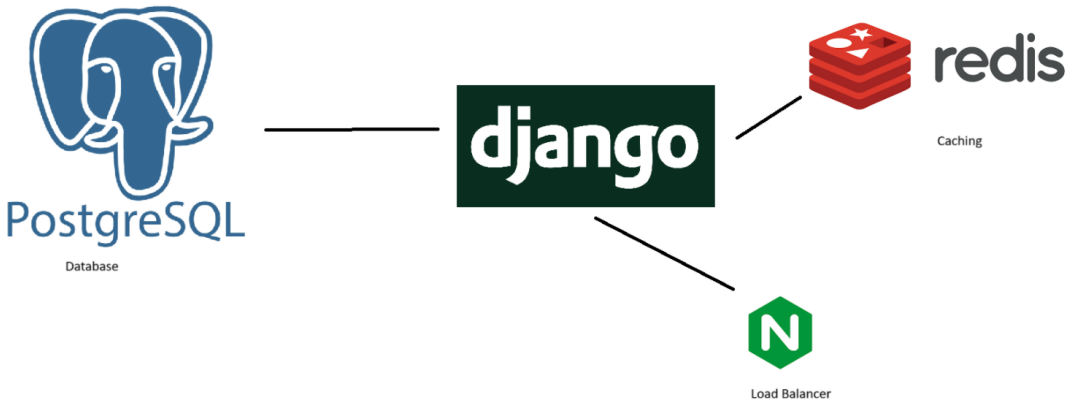
By word high-load systems we typically refer to architectures that were designed to handle a large volume of requests and data. These systems are crucial in today's web applications, especially in e-commerce sector, where user traffic periodically hit peak traffics during shopping periods.

## 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.

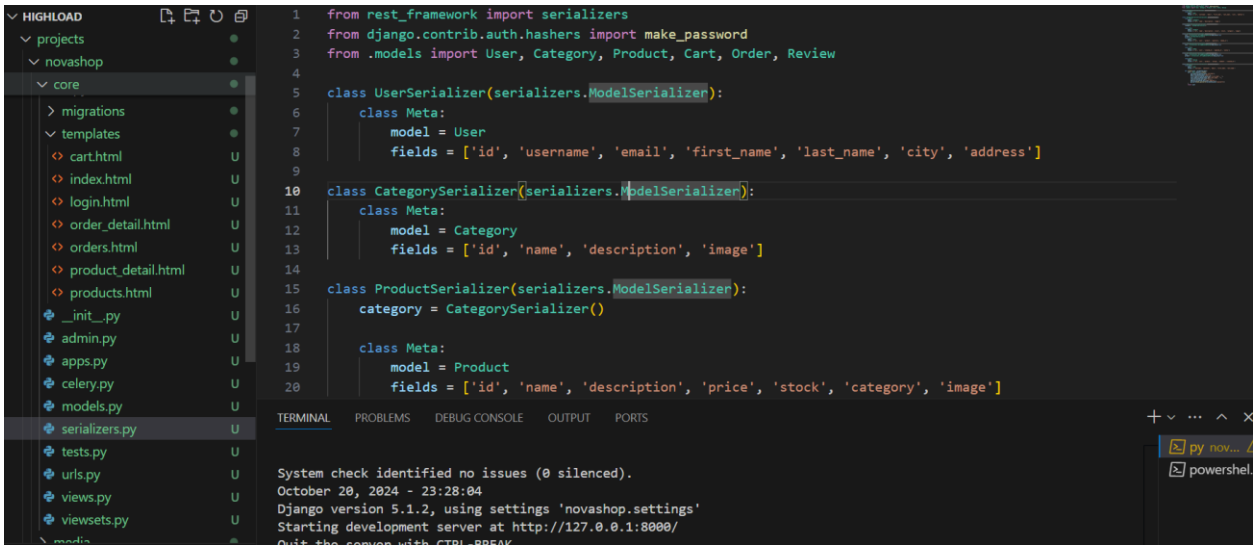
**Message Broker**:

- **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

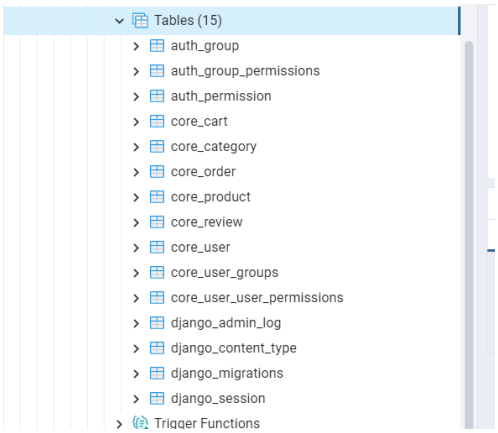



Our apps structure

```python
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

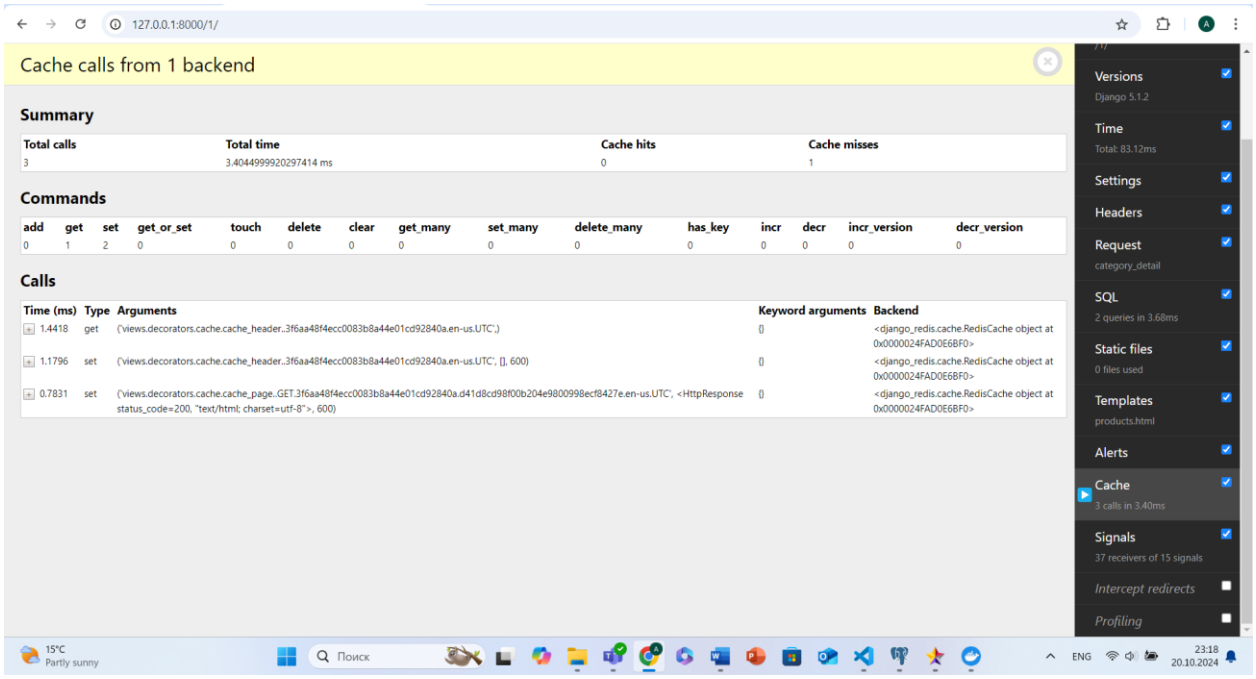
We enabled database indexing to speed performance of our app.

```python
class Product(models.Model):
    name = models.CharField(max_length=255)
    description = models.TextField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    stock = models.PositiveIntegerField(default=0)
    category = models.ForeignKey(Category, related_name='products', on_delete=models.CASCADE)
    image = models.ImageField(upload_to='products_images')

    class Meta:
        indexes = [
            models.Index(fields=['price']),
            models.Index(fields=['stock']),
        ]

    def __str__(self) -> str:
        return self.name
```

```python
class Order(models.Model):
    user = models.ForeignKey(User, related_name='orders', on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    status = models.CharField(max_length=20, choices=(('Pending', 'Pending'), ('Completed', 'Completed'), ('Cancele

    class Meta:
        indexes = [
            models.Index(fields=['status']),
        ]

    def __str__(self) -> str:
```

## 7. Caching Strategies

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




```python
@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})
```

```python
class ProductViewSet(viewsets.ModelViewSet):
    queryset = Product.objects.all()
    serializer_class = ProductSerializer

    def list(self, request, *args, **kwargs):
        cache_key = 'product_list'
        products = cache.get(cache_key)

        if not products:
            products = super().list(request, *args, **kwargs).data
            cache.set(cache_key, products, timeout=60*15)

        return Response(products)

```

**8. Load Balancing Techniques**

C: > Users > wwwis > nginx > nginx > conf > nginx.conf

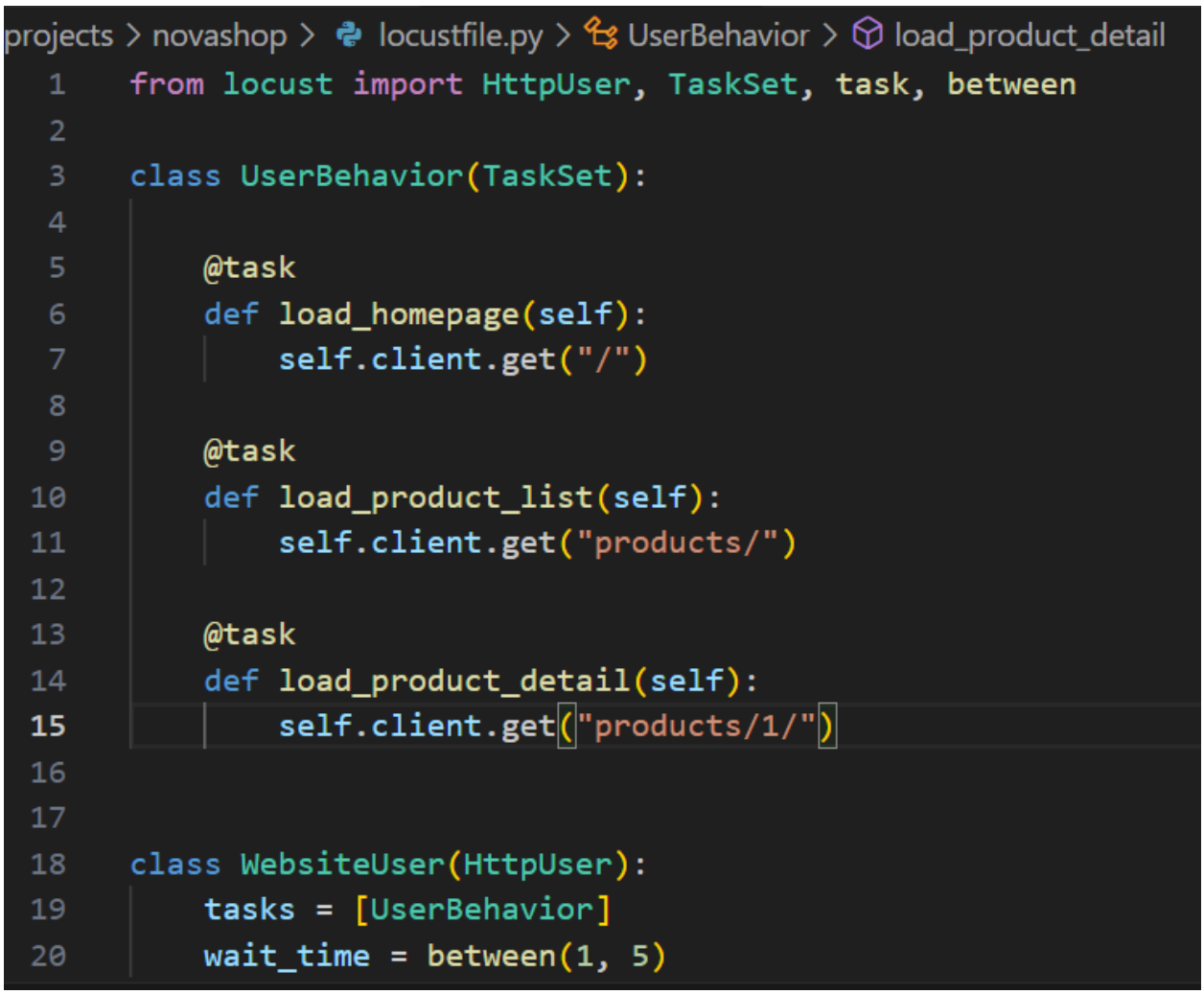
```
http {
    server 127.0.0.1:8001;
    server 127.0.0.1:8002;
}

    server {
        listen 80;

        server_name your-domain.com;

        location / {
            proxy_pass http://django_servers;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }

}
```

**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.**




Locust file to load test our app.

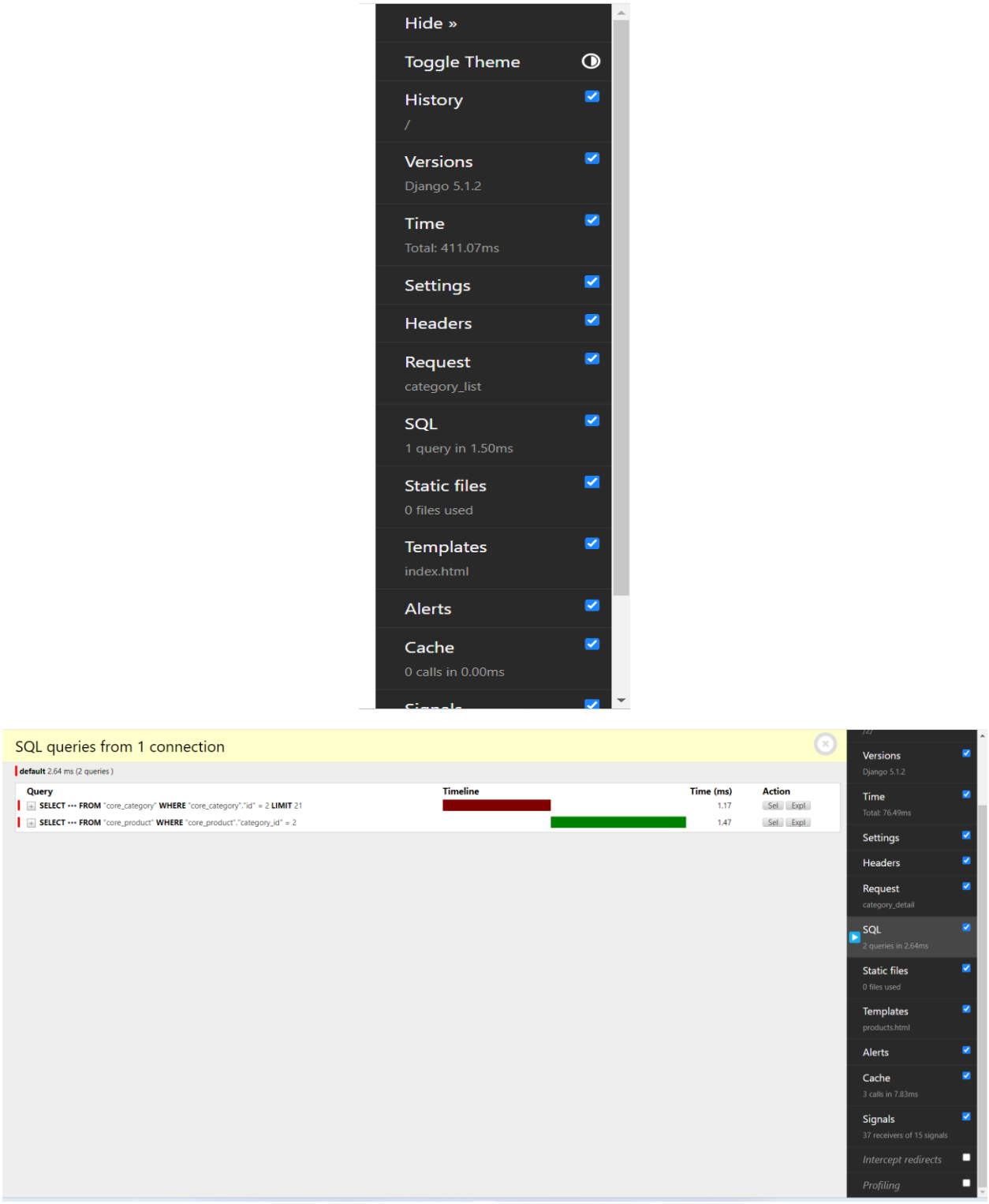
**11. Monitoring and Maintenance**

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

```python
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': 'django_debug.log',
        },
    },
    'loggers': {
        'django': {
            'handlers': ['file'],
            'level': 'DEBUG',
            'propagate': True,
        },
    },
}
```

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 wi
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
14  File C:\Users\wwwis\AppData\Local\Programs\Python\Python310\Lib\zoneinfo\_tzpath.py first seen with mtime 16424079
15  File C:\Users\wwwis\Documents\highload\projects\.venv\Lib\site-packages\django\contrib\postgres\search.py first se
16  File C:\Users\wwwis\Documents\highload\projects\.venv\Lib\site-packages\urllib3\util\url.py first seen with mtime
17  File C:\Users\wwwis\Documents\highload\projects\.venv\Lib\site-packages\debug_toolbar\panels\sql\panel.py first se
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 16424079
```




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.