Backend for High Loaded Environment: Final Assignment

Author: Sazanova Aruzhan

Date of submission: 14.12.2024

School of Information Technology and Engineering

Specialty: Information Systems
3rd academic year

2. Executive Summary

The High-Load E-Commerce Web Platform aims to deliver a scalable and high-performance solution for managing high traffic in e-commerce applications. The project leverages modern tools like PostgreSQL, Redis, Cassandra, and various caching techniques to ensure quick access to product and user data. The platform is optimized for both performance and security, using load balancing techniques with Nginx, continuous integration with GitHub Actions, and robust monitoring through Grafana and Prometheus.

Key Recommendations:

- Future scalability improvements include better database partitioning and microservices implementation.
- Continuous performance tuning using tools like JMeter will be crucial for maintaining system reliability.

3. Table of Contents

4. Introduction

Background:

This project focuses on building a scalable and efficient backend for an e-commerce web platform capable of handling high traffic and large amounts of data. It aims to

provide a smooth user experience under heavy loads by leveraging technologies like Redis for caching, PostgreSQL and Cassandra for data storage, and Nginx for load balancing.

Objectives:

- Develop and optimize a high-load backend using PostgreSQL, Redis, and Cassandra.
- Implement caching strategies to improve performance.
- Ensure data consistency in a distributed environment.
- Create secure authentication for user access.
- Optimize system scalability and performance through load balancing and microservices.

Scope:

This project covers the backend infrastructure and architecture, including database design, caching, security, performance monitoring, and continuous integration. The frontend and deployment configurations are assumed to be handled separately.

5. Project Structure

High-Level Architecture Diagram:

Generated with the help of *django-extensions* and *pygraphviz* libraries. [Appendices 1]

- **Frontend**: Communicates with the backend via RESTful APIs and html templates
  [Appendices 2]

- **Backend**: Built using Django, interacting with PostgreSQL for relational data and Cassandra for high-load, distributed data storage. [Appendices 3]

- **Cache Layer**: Redis is used to cache frequently accessed data (e.g., product listings). [Appendices 4]

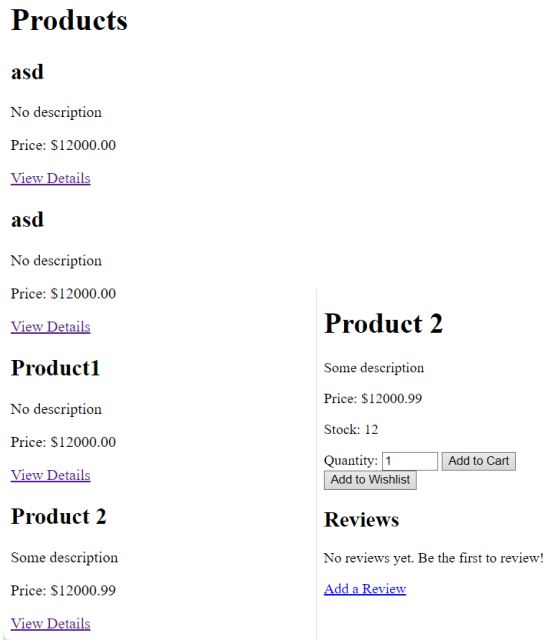
- **Load Balancer**: Nginx distributes incoming traffic across multiple Django instances. [Appendices 5]

- **Monitoring**: Prometheus and Grafana track the performance and health of the system. [Appendices 6 and 7]

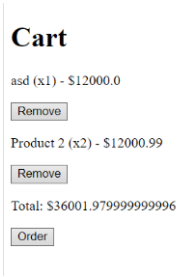
Key Components:

- **User Authentication**: Handled using email/password or token-based authentication. [Appendices 8]
- **Order Management**: Users can place orders, and the order system ensures high consistency even with large traffic.

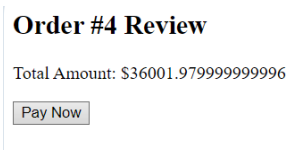
I choose a product and set quantity:



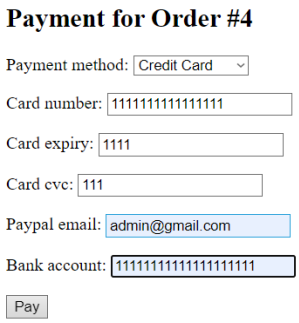
Press add to cart



Press order:

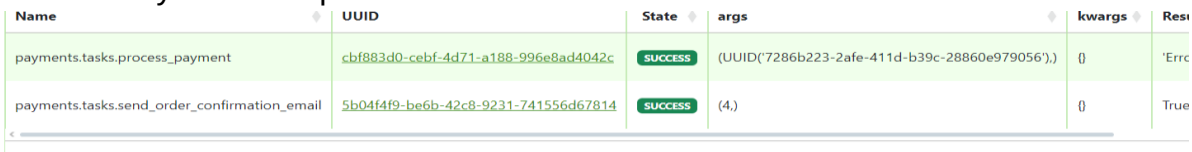


Press Pay now and fill info:



Press pay:

Check celery flower on port 5555:



- **Caching**: Redis is employed to cache products and prevent frequent database hits.

Perform caching on *api/products, api/orders, api/product/<int:id>, api/orders/<int:id>* [Appendices 9]

- **Load Balancing**: Nginx is used to handle high traffic efficiently.

6. Table Entities

User

- **id**: Primary key
- **username**: Unique identifier
- **email**: User's email address
- **password**: Encrypted password
- **created_at**: Timestamp of account creation
- **updated_at**: Timestamp of last update

Product

- **id**: Primary key
- **name**: Name of the product

- **price**: Price of the product
- **category_id**: Foreign key to Category
- **created_at**: Timestamp of product addition
- **updated_at**: Timestamp of last update

Category

- **id**: Primary key
- **name**: Name of the category
- **parent_id**: Optional reference to a parent category
- **created_at**: Timestamp of category creation
- **updated_at**: Timestamp of last update

7. Development Process

Technologies Used:

- **Django**: For building the RESTful API.
- **PostgreSQL**: Relational database for structured data.
- **Cassandra**: For handling large amounts of unstructured data at scale.
- **Redis**: For caching.
- **Nginx**: For load balancing.
- **Prometheus and Grafana**: For monitoring.
- **GitHub Actions**: For CI/CD.
- **JMeter**: For load testing.
- **pytest**: For unit and integration testing.

Implementation:

1. **Django Setup**: Configure the Django project and set up models, views, and serializers for the API.
2. **Caching**: Implement Redis caching for frequently accessed endpoints like product listings.
3. **Database Design**: Use PostgreSQL for structured data and Cassandra for high-load scenarios. Implement indexing and normalization strategies.
4. **Load Balancing**: Set up Nginx to distribute traffic across application servers.
5. **Security**: Implement email/password authentication or token-based authentication using Django's built-in user model.

8. Endpoints

Payment App Endpoints

- **GET** `/payment/order/`: View details of the current order.
- **POST** `/payment/pay/<int:order_id>/`: Process payment for the given order.
- **GET** `/payment/order_success/`: Display order success message after payment is completed.

Market App Endpoints

- **GET** `/market/order/`: View details of the current order.
- **POST** `/market/pay/<int:order_id>/`: Process payment for the given order.
- **GET** `/market/order_success/`: Display order success message after payment is completed.

API App Endpoints

- **GET** `/api/users/`: Fetch list of users.
- **GET** `/api/users/<int:id>/`: Fetch details of a specific user.
- **GET** `/api/categories/`: Fetch a list of product categories.
- **GET** `/api/categories/<int:id>/`: Fetch details of a specific category.
- **GET** `/api/products/`: Fetch a list of products.
- **GET** `/api/products/<int:id>/`: Fetch details of a specific product.
- **GET** `/api/orders/`: Fetch a list of orders.
- **GET** `/api/orders/<int:id>/`: Fetch details of a specific order.
- **POST** `/api/orders/`: Create an order.
- **GET** `/api/order-items/`: Fetch a list of order items.
- **POST** `/api/order-items/`: Create a new order item.
- **GET** `/api/shopping-carts/`: Fetch a list of shopping carts.
- **GET** `/api/shopping-carts/<int:id>/`: Fetch details of a specific shopping cart.
- **POST** `/api/shopping-carts/`: Create a shopping cart.
- **GET** `/api/cart-items/`: Fetch a list of cart items.
- **GET** `/api/cart-items/<int:id>/`: Fetch details of a specific cart item.
- **POST** `/api/cart-items/`: Create a new cart item.
- **GET** `/api/payments/`: Fetch a list of payments.
- **GET** `/api/payments/<int:id>/`: Fetch details of a specific payment.

- **POST** `/api/payments/`: Process a payment.
- **GET** `/api/reviews/`: Fetch a list of reviews.
- **GET** `/api/reviews/<int:id>/`: Fetch details of a specific review.
- **POST** `/api/reviews/`: Add a new review.
- **GET** `/api/wishlists/`: Fetch a list of wishlists.
- **GET** `/api/wishlists/<int:id>/`: Fetch details of a specific wishlist.
- **POST** `/api/wishlists/`: Create a new wishlist.
- **GET** `/api/wishlist-items/`: Fetch a list of wishlist items.
- **GET** `/api/wishlist-items/<int:id>/`: Fetch details of a specific wishlist item.
- **POST** `/api/wishlist-items/`: Add a product to the wishlist.
- **POST** `/api/auth/register/`: User registration via email/password.
- **POST** `/api/auth/login/`: User login via email/password.

Frontend App Endpoints

- **GET** `/`: Display the shop's homepage.
- **GET** `/products/`: Fetch a list of all products.
- **GET** `/products/<int:category_id>/`: Fetch products filtered by category.
- **GET** `/product/<int:product_id>/`: View detailed information about a specific product.
- **GET** `/cart/`: View the current user's shopping cart.
- **POST** `/cart/add/<int:product_id>/`: Add a product to the shopping cart.
- **POST** `/cart/remove/<int:item_id>/`: Remove an item from the shopping cart.
- **GET** `/orders/`: View the user's order history.
- **GET** `/wishlist/`: View the user's wishlist.
- **POST** `/wishlist/add/<int:product_id>/`: Add a product to the wishlist.
- **POST** `/wishlist/remove/<int:item_id>/`: Remove an item from the wishlist.
- **POST** `/reviews/<int:product_id>/`: Add a review for a product.

9. Database Design and Optimization

- **Indexes**: Index commonly queried fields (e.g., `product_id`, `category_id`, `order_id`). [Appendices 10]
- **Normalization**: Data is normalized into tables like `Product`, `Order`, `User` to avoid redundancy. [Appendices 10]

- **Query Optimization**: Write efficient queries and use database views for complex aggregations. [Appendices 11]

10. Caching Strategies

- **Redis Caching**: Cache product details and product listings to reduce database load. [Appendices 4]

11. Load Balancing

- **Nginx Configuration**: Configure Nginx to balance traffic between multiple instances of the Django app. [Appendices 5]

12. Security Measures

- **Password Encryption**: Store passwords securely using Django's built-in `make_random_password` or libraries like `bcrypt`.
- **Authentication**: Implement token-based authentication using Django Rest Framework's JWT authentication. [Appendices 8]
- **Middlware**: Wrote custom middleware `SecureHeadersMiddleware.`

13. Monitoring and Performance

- **Prometheus Configuration**: Set up Prometheus to monitor API performance. [Appendices 6]
- **Grafana**: Create dashboards to visualize API response times, request count, etc. [Appendices 7]

14. Challenges and Solutions

- **Database Load**: Implementing Cassandra for handling distributed data allowed the system to handle high concurrency without performance degradation.
- **Caching**: Redis was crucial in reducing the load on PostgreSQL for read-heavy endpoints like `/products`.
- **Challenges in Cassandra Implementation and Solutions:**
- **Data Modeling Complexity**

    - *Challenge*: Cassandra requires designing data models around queries, not normalized schema.
    - *Solution*: Focus on query-first design and denormalize data to optimize read/write efficiency.
- **Cluster Management**
    - *Challenge*: Ensuring consistent data replication and balancing across nodes.
    - *Solution*: Use Cassandra's built-in tools (e.g., nodetool, repair, and rebalancing) and monitor performance metrics.
- **Handling Large Volumes of Data**
    - *Challenge*: Managing read/write performance at scale.
    - *Solution*: Use partition keys effectively to distribute data evenly and avoid hot spots.
- **Operational Overhead**
    - *Challenge*: Maintaining fault tolerance and availability in multi-datacenter setups.
    - *Solution*: Enable replication across data centers with proper consistency levels (`QUORUM`, `LOCAL_ONE`).
- **Limited Secondary Indexes**
    - *Challenge*: Indexes can slow performance on large datasets.
    - *Solution*: Optimize queries with materialized views or custom secondary indexing strategies.
-

15. Conclusion

The project successfully implemented a scalable e-commerce platform capable of handling high traffic and offering an excellent user experience. Future improvements include microservice decomposition for scaling and even more robust monitoring systems.

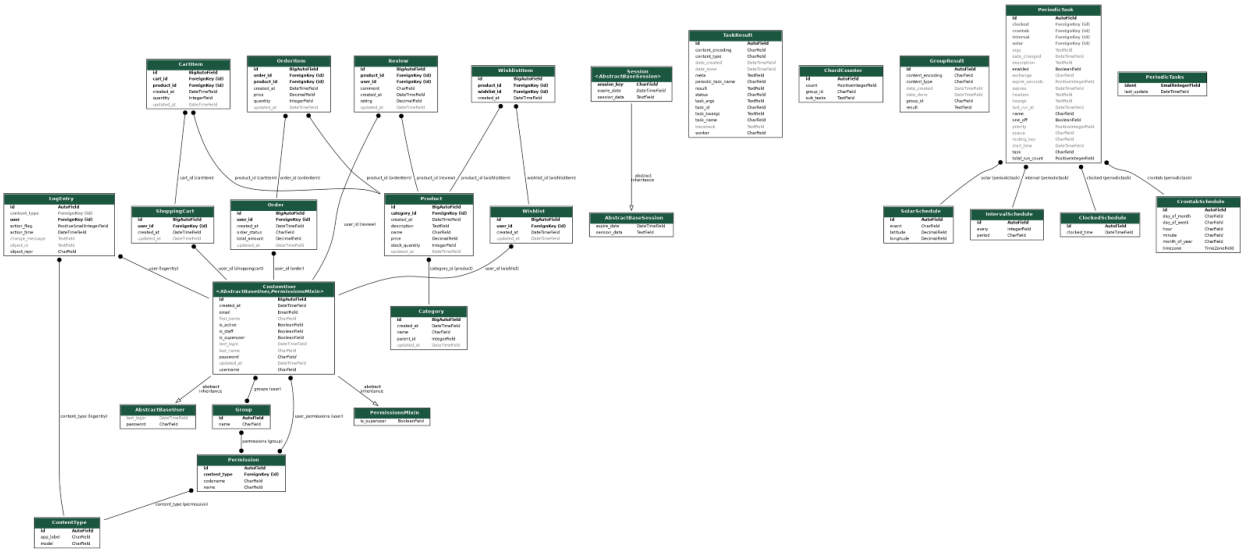
16. References

- Django Documentation
- Redis and Cassandra Documentation
- Nginx Load Balancing Guide
- Prometheus and Grafana Documentation

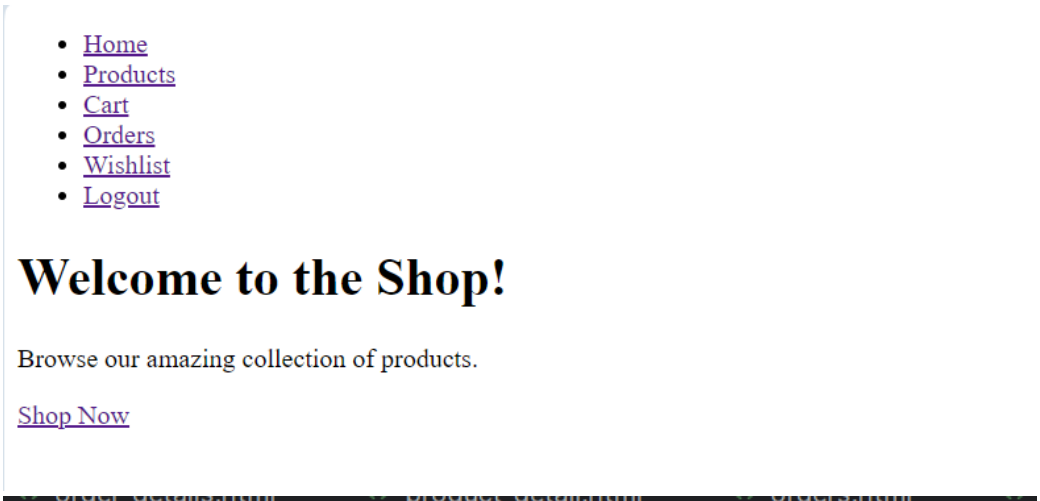
17. Appendices

Include architecture diagrams, code samples, and additional configurations.

[Appendices 1] ERD



[Appendices 2] Frontend fragmnent




```
{% extends 'base.html' %}
{% block content %}
<h1>Welcome to the Shop!</h1>
<p>Browse our amazing collection of products.</p>
<a href="{% url 'products' %}" class="button">Shop Now</a>
{% endblock %}
```

[Appendices 3] DATABASES from settings.py

```
DATABASES = {
    'cassandra': {
        'ENGINE': 'django_cassandra_engine',
        'NAME': 'my_keyspace',
        'HOST': 'localhost',
        'PORT': 9042,
        'OPTIONS': {
            'replication_factor': 3,
            'consistency_level': 'LOCAL_QUORUM',
        },
    },
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'new_db',
        'USER': 'admin',
        'PASSWORD': 'admin',
        'HOST': 'localhost',
        'PORT': '5432',
    },
    'replica2': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'db_replica2',
        'USER': 'admin',
        'PASSWORD': 'admin',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

[Appendices 4] Caching

```
CACHES = {
    'default': {
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': 'redis://127.0.0.1:6379/1',
        'OPTIONS': {
            'CLIENT_CLASS': 'django_redis.client.DefaultClient',
        }
    }
}
```

```
@method_decorator(cache_page(CACHE_TIMEOUT), name='dispatch')
class OrderItemViewSet(viewsets.ModelViewSet):
    queryset = OrderItem.objects.select_related('order_id', 'product_id')
    serializer_class = OrderItemSerializer
    permission_classes = [IsAuthenticated]
```

```python
class ProductViewSet(viewsets.ModelViewSet):
    queryset = Product.objects.select_related('category_id')
    serializer_class = ProductSerializer
    permission_classes = [IsAuthenticated]

    xydownik
    def list(self, request, *args, **kwargs):
        cache_key = "products_list"
        cached_data = cache.get(cache_key)

        if cached_data:
            return Response(cached_data)

        queryset = self.get_queryset()
        serializer = self.get_serializer( *args: queryset, many=True)
        response_data = serializer.data

        cache.set(cache_key, response_data, CACHE_TIMEOUT)
        return Response(response_data)
```

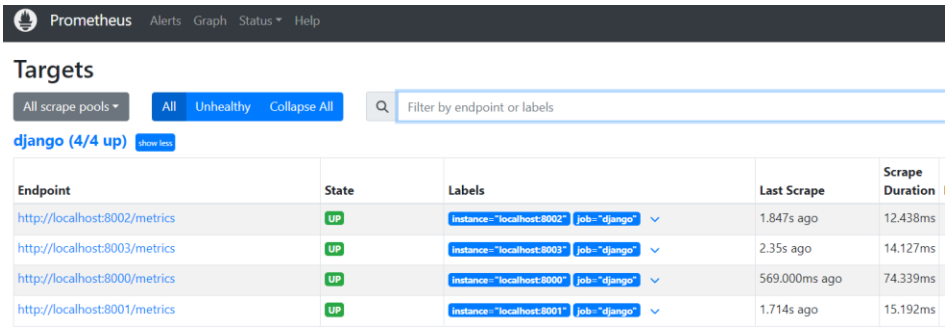
[Appendices 5] NGINX configuration

```
events {
    worker_connections 4096;
}
http{
    upstream django_servers {
        ip_hash;
        server 127.0.0.1:8000 max_fails=3 fail_timeout=30s;
        server 127.0.0.1:8001 max_fails=3 fail_timeout=30s;
        server 127.0.0.1:8002 max_fails=3 fail_timeout=30s;
        server 127.0.0.1:8003 max_fails=3 fail_timeout=30s;
    }
    server {
        listen 8000;
        location /static/ {
            alias /home/sazanova/highload/final/market/staticfiles;
            autoindex on;
        }
        location / {
            proxy_pass http://app:8000;
        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;
                }
        location /health {
            proxy_pass http://app/health;
            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;
                }
        location /metrics {
```

Endpoint	State
http://localhost:8002/metrics	UP
http://localhost:8003/metrics	UP
http://localhost:8000/metrics	UP
http://localhost:8001/metrics	UP

[Appendices 6]

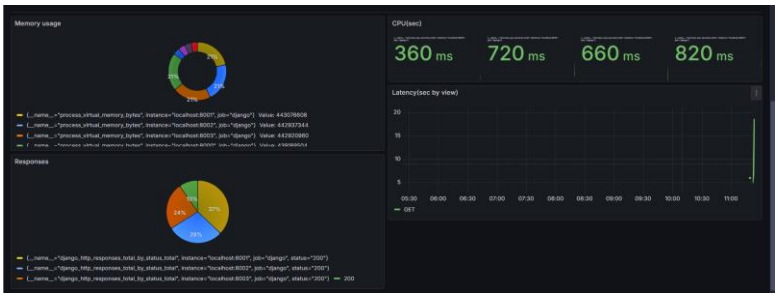
Prometheus:



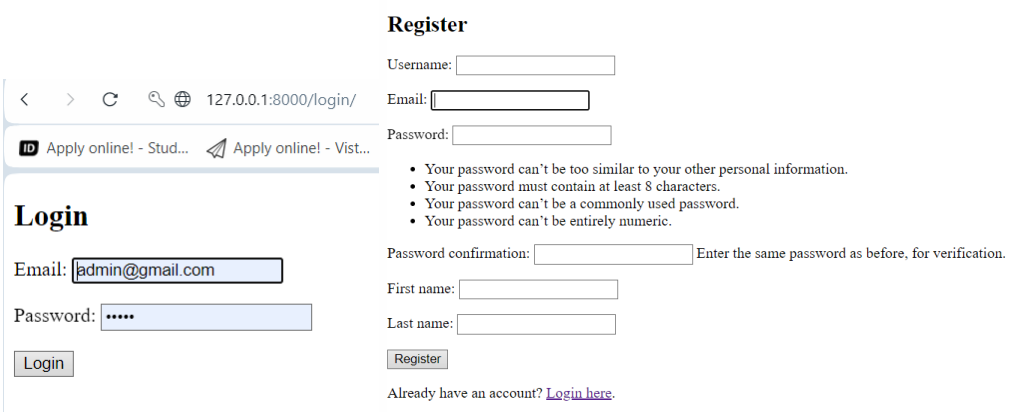

```yaml
global:
  scrape_interval: 5s
scrape_configs:
  - job_name: 'django'
    static_configs:
      - targets: ['localhost:8000','localhost:8001', 'localhost:8002', 'localhost:8003']
```

[Appendices 7]

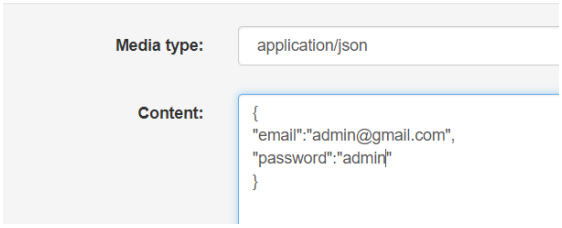
Grafana:

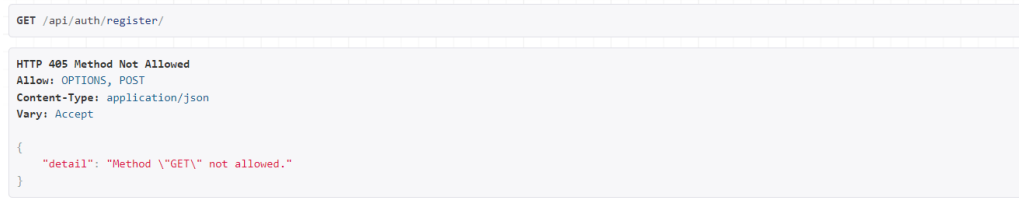


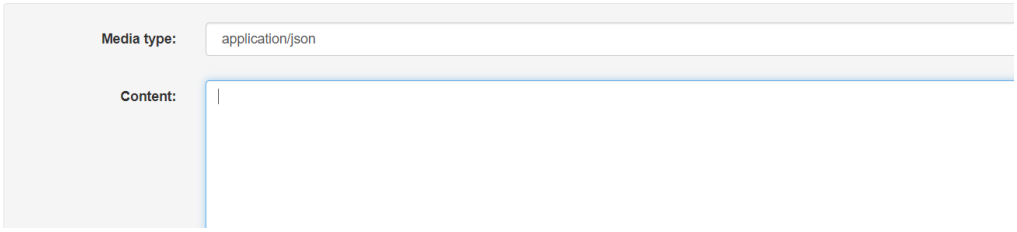
[Appendices 8] Authentication and Regitration
Fast authentication and Regitration:



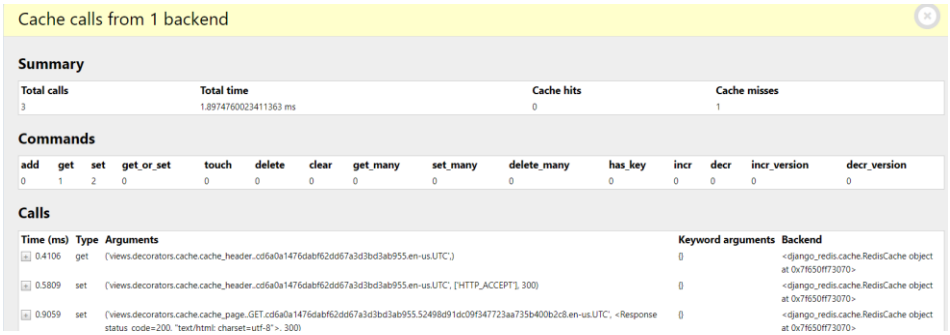
Safe authentication and Regitration:







[Appendices 9] Caching in **debug toolbar**



Cache calls from 1 backend

Summary

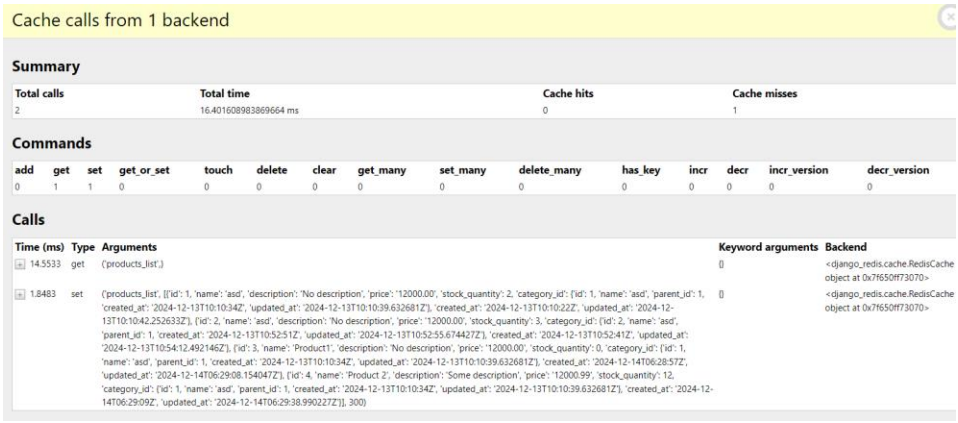
Total calls	Total time	Cache hits	Cache misses
3	1.8974760023411363 ms	0	1

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
0.4106	get	('views.decorators.cache.cache_header..cd6a0a1476dabf62dd67a3d3bd3ab955.en-us.UTC',)	{}	<django_redis.cache.RedisCache object at 0x7f650ff73070>
0.5809	set	('views.decorators.cache.cache_header..cd6a0a1476dabf62dd67a3d3bd3ab955.en-us.UTC', ['HTTP_ACCEPT'], 300)	{}	<django_redis.cache.RedisCache object at 0x7f650ff73070>
0.9059	set	('views.decorators.cache.cache_page..GET.cd6a0a1476dabf62dd67a3d3bd3ab955.52498d91dc09f347723aa735b400b2c8.en-us.UTC', <Response status_code=200, "text/html; charset=utf-8">, 300)	{}	<django_redis.cache.RedisCache object at 0x7f650ff73070>



Cache calls from 1 backend

Summary

Total calls	Total time	Cache hits	Cache misses
2	16.401608983869664 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	1	0	0	0	0	0	0	0	0	0	0	0	0

Calls

Time (ms)	Type	Arguments	Keyword arguments	Backend
14.5533	get	('products_list',)	{}	<django_redis.cache.RedisCache object at 0x7f650ff73070>
1.8483	set	('products_list', [{'id': 1, 'name': 'asd', 'description': 'No description', 'price': '12000.00', 'stock_quantity': 2, 'category_id': {'id': 1, 'name': 'asd', 'parent_id': 1, 'created_at': '2024-12-13T10:10:34Z', 'updated_at': '2024-12-13T10:10:39.632681Z'}, 'created_at': '2024-12-13T10:10:22Z', 'updated_at': '2024-12-13T10:10:42.252633Z'}, {'id': 2, 'name': 'asd', 'description': 'No description', 'price': '12000.00', 'stock_quantity': 3, 'category_id': {'id': 2, 'name': 'asd', 'parent_id': 1, 'created_at': '2024-12-13T10:52:51Z', 'updated_at': '2024-12-13T10:52:55.674427Z'}, 'created_at': '2024-12-13T10:52:41Z', 'updated_at': '2024-12-13T10:54:12.492146Z'}, {'id': 3, 'name': 'Product1', 'description': 'No description', 'price': '12000.00', 'stock_quantity': 0, 'category_id': {'id': 1, 'name': 'asd', 'parent_id': 1, 'created_at': '2024-12-13T10:10:34Z', 'updated_at': '2024-12-13T10:10:39.632681Z'}, 'created_at': '2024-12-14T06:28:57Z', 'updated_at': '2024-12-14T06:29:08.154047Z'}, {'id': 4, 'name': 'Product 2', 'description': 'Some description', 'price': '12000.99', 'stock_quantity': 12, 'category_id': {'id': 1, 'name': 'asd', 'parent_id': 1, 'created_at': '2024-12-13T10:10:34Z', 'updated_at': '2024-12-13T10:10:39.632681Z'}, 'created_at': '2024-12-14T06:29:09Z', 'updated_at': '2024-12-14T06:29:38.990227Z'}], 300)	{}	<django_redis.cache.RedisCache object at 0x7f650ff73070>

[Appendices 10] Indexing and Models

```python
class WishlistItem(models.Model):
    wishlist_id = models.ForeignKey(Wishlist, on_delete=models.CAS
    product_id = models.ForeignKey(Product, on_delete=models.CASCA
    created_at = models.DateTimeField(default=now)

    xydownik
    def __str__(self):
        return f"{self.wishlist_id} - {self.product_id}"

    xydownik *
    class Meta:
        indexes = [
            models.Index(fields=['wishlist_id', 'product_id']),
        ]
```

```python
class Wishlist(models.Model):
    user_id = models.ForeignKey(CustomUser, on_delete=mod
    created_at = models.DateTimeField(default=now)
    updated_at = models.DateTimeField(auto_now_add=True)

    xydownik
    def __str__(self):
        return str(self.user_id)

    xydownik *
    class Meta:
        indexes = [
            models.Index(fields=['user_id']),
        ]
```

```python
class Review(models.Model):
    product_id = models.ForeignKey(Product, on_delete=model
    user_id = models.ForeignKey(CustomUser, on_delete=model
    rating = models.DecimalField(max_digits=10, decimal_pla
    comment = models.CharField(max_length=255)
    created_at = models.DateTimeField(default=now)
    updated_at = models.DateTimeField(auto_now_add=True)

    xydownik
    def __str__(self):
        return self.comment

    xydownik *
    class Meta:
        indexes = [
            models.Index(fields=['product_id']),
            models.Index(fields=['user_id']),
            models.Index(fields=['rating']),
        ]
```

```python
class CartItem(models.Model):
    cart_id = models.ForeignKey(ShoppingCart, on_delete=models.CASC
    product_id = models.ForeignKey(Product, on_delete=models.CASCAD
    quantity = models.IntegerField(default=1)
    created_at = models.DateTimeField(default=now)
    updated_at = models.DateTimeField(auto_now_add=True)

    xydownik
    def __str__(self):
        return f"{self.cart_id} - {self.product_id}"

    xydownik *
    class Meta:
        indexes = [
            models.Index(fields=['cart_id', 'product_id']),
        ]
```

```python
class ShoppingCart(models.Model):
    user_id = models.ForeignKey(CustomUser, on_dele
    created_at = models.DateTimeField(default=now)
    updated_at = models.DateTimeField(auto_now_add=

    xydownik
    def __str__(self):
        return str(self.user_id)

    xydownik
    class Meta:
        indexes = [
            models.Index(fields=['user_id']),
        ]
```

```python
class OrderItem(models.Model):
    order_id = models.ForeignKey(Order, on_delete=models.CAS
    product_id = models.ForeignKey(Product, on_delete=models
    quantity = models.IntegerField(default=0)
    price = models.DecimalField(max_digits=10, decimal_place
    created_at = models.DateTimeField(default=now)
    updated_at = models.DateTimeField(auto_now_add=True)

    xydownik
    def __str__(self):
        return f"{self.order_id} - {self.product_id}"

    xydownik
    class Meta:
        indexes = [
            models.Index(fields=['order_id', 'product_id']),
        ]
```

```python
class Order(models.Model):
    user_id = models.ForeignKey(CustomUser, on_delete=models.CASCADE)
    order_status = models.CharField(max_length=255)
    total_amount = models.DecimalField(max_digits=10, decimal_places=2)
    created_at = models.DateTimeField(default=now)
    updated_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.order_status

    class Meta:
        indexes = [
            models.Index(fields=['user_id']),
            models.Index(fields=['order_status']),
        ]
```

```python
class Product(models.Model):
    name = models.CharField(max_length=255)
    description = models.TextField(default='No description')
    price = models.DecimalField(max_digits=10, decimal_places=2)
    stock_quantity = models.IntegerField(default=0)
    category_id = models.ForeignKey(Category, on_delete=models.CASCADE)
    created_at = models.DateTimeField(default=now)
    updated_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

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

```python
class CustomUser(AbstractBaseUser, PermissionsMixin):
    username = models.CharField(max_length=150, unique=True)
    email = models.EmailField(unique=True)
    first_name = models.CharField(max_length=150, blank=True)
    last_name = models.CharField(max_length=150, blank=True)
    created_at = models.DateTimeField(default=now)
    updated_at = models.DateTimeField(auto_now=True)

    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)

    objects = CustomUserManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['username']

    def __str__(self):
        return self.email

    class Meta:
        indexes = [
            models.Index(fields=['email']),
            models.Index(fields=['username']),
        ]
```

[Appendices 11] ORM

```python
queryset = Review.objects.select_related('product_id', 'user_id')
```

```python
queryset = Order.objects.select_related('user_id')
```

```python
queryset = Product.objects.select_related('category_id')
```