Улучшение производительности

Цель: Оптимизировать код формирования отчета о продуктах для улучшения производительности.

Проанализирован и оптимизирован код, отвечающий за формирование отчета о продуктах. Были применены следующие меры:

1. Использование select_related и prefetch_related:
    - Был добавлен select_related('category_id') для эффективной выборки связанной категории продукта.
    - Был добавлен prefetch_related('tags') для предварительной загрузки тегов продуктов.
    - Это позволило снизить количество SQL-запросов и избежать проблемы N+1.
2. Использование only() для выбора необходимых полей:
    - Применена функция only() с указанием только тех полей, которые используются при формировании отчета.
    - Это позволило сократить объем извлекаемых данных из базы данных.

Результаты:

- Код был оптимизирован с использованием select_related, prefetch_related и only() для улучшения производительности.
- Количество SQL-запросов снижено, что ускорило загрузку данных и сделало процесс формирования отчета более эффективным.

Безопасность и использование JWT-токенов для регистрации и авторизации

Цель: Обеспечить безопасность системы через использование JWT-токенов для регистрации и авторизации пользователей.
Применена мера безопасности permission_classes = [IsAuthenticated] в представлениях.

JWT (JSON Web Token) и его роль в безопасности:
JWT является стандартом для представления утверждений между двумя сторонами в виде JSON-объекта. Он широко используется в системах аутентификации и авторизации для обмена информацией между клиентом и сервером без необходимости хранения состояния на сервере.

Шаги, принятые для обеспечения безопасности:

1. Регистрация и выдача токенов:
    - При успешной регистрации пользователя выдается JWT-токен, который содержит уникальную информацию о пользователе.
    - JWT-токен включает в себя информацию о правах доступа и сроке действия.
2. Аутентификация и авторизация:
    - Пользователь отправляет полученный токен с запросом на получение данных и товарах.
    - Сервер проверяет подлинность токена и права доступа пользователя.
    - При использовании permission_classes = [IsAuthenticated] в представлениях Django REST framework, только аутентифицированные пользователи с действительными токенами могут получить доступ к защищенным ресурсам.
3. Обновление токенов и безопасное хранение:
    - Для обеспечения безопасности, токен для доступа к данным имеет ограниченное время действия (15 минут).
    - Для продления сессии пользователь может запросить новый токен перед истечением срока действия.

Результаты:

- Использование JWT-токенов предоставило безопасный и эффективный механизм для аутентификации и авторизации пользователей.
- Мера безопасности permission_classes = [IsAuthenticated] в представлениях Django REST framework обеспечила защиту доступа к конкретным ресурсам только аутентифицированными пользователями.

Кеширование и инвалидация кеша в системе

Цель: Реализовать кеширование данных и инвалидацию кеша при изменении и удалении товаров в системе с помощью Redis.

Методы для реализации кеширования и инвалидации кеша:

1. Использование сигналов для инвалидации кеша:
    - Созданы сигналы @receiver(post_save, sender=Product) и @receiver(post_delete, sender=Product), связанные с функцией product_change_handler.
    - При сохранении или удалении объекта Product, функция product_change_handler очищает кеш с помощью cache.clear().
2. Кеширование данных в представлении:
    - В представлении ProductListAPIView использованы select_related и prefetch_related для оптимизации запроса к базе данных.
    - Применен декоратор @cache_page(60*15) для кеширования результатов представления на 15 минут.

Результаты:
Реализация сигналов и кеширования позволила эффективно управлять кешем данных и инвалидировать его при изменениях.

Использовании Docker Compose

Цель: Организовать развертывание приложения с использованием Docker Compose для создания и управления контейнерами.

Dockerfile:

- Основной образ: python:3.10
- Установлены переменные окружения для предотвращения создания байт-кода и буферизации вывода Python.
- Установлены зависимости, включая libpq-dev.
- Содержимое текущей директории скопировано в рабочую директорию контейнера.
- Установлены зависимости из requirements.txt.

docker-compose.yml:

- Используется версия 3.5 с помощью version: '3.5'.
- Определены следующие сервисы:

1. db:
    - Используется образ postgres для запуска контейнера PostgreSQL.
    - Данные сохраняются в локальной директории ./data/db.
    - Установлены переменные окружения для базы данных: POSTGRES_DB, POSTGRES_USER и POSTGRES_PASSWORD.
2. redis:
    - Используется образ redis:alpine для запуска контейнера Redis.
    - Порт 6379 контейнера проксируется на хостовый порт 6379.
3. web:
    - Собирается образ из текущей директории (build: .) с использованием Dockerfile.
    - Зависимость от сервиса db.
    - Запускается команда для применения миграций и запуска сервера Django.
    - Директория проекта монтируется в контейнер.
    - Порт 8000 контейнера проксируется на хостовый порт 8000.
    - Установлены переменные окружения для базы данных и Redis.

Результат и преимущества: Использование Docker Compose позволяет легко развертывать множество связанных сервисов с единым описанием.

Предоставляется изолированное окружение для каждого сервиса, что обеспечивает надежность и предотвращает конфликты между зависимостями.