#### **Области системы для кеширования**

* **API интернет-магазина**
  + Каталог товаров (список категорий, описание, изображения).
  + Часто запрашиваемые данные о популярных товарах (рейтинг, остатки на складе).
* **CRM**
  + Информацию о клиентах, с которыми активно работают менеджеры (имена, последние заказы, контактные данные).
  + Часто используемые справочники (статусы заказов, типы операций).
* **MES**
  + Часто запрашиваемые данные о текущем состоянии заказов (статус производства, сроки выполнения).

# Мотивация

Кеширование позволяет повысить производительность системы, сократить время отклика для пользователей и уменьшить нагрузку на базу данных и другие ключевые сервисы.

### **Преимущества внедрения**

* **Скорость работы системы:** Уменьшение времени ответа API и интерфейсов пользователя.
* **Устойчивость к нагрузкам:** Снижение количества запросов к базам данных и микросервисам.
* **Экономия ресурсов:** Сокращение объёма вычислений и операций ввода-вывода на серверах.
* **Лучшая масштабируемость:** Возможность обслуживать больше пользователей без существенного увеличения нагрузки на систему.

# Предлагаемое решение

Для системы предлагается внедрить серверное кеширование, так как оно обеспечивает контроль над данными, подходящее время их обновления, и позволяет избежать избыточной нагрузки на клиенты. Клиентское кеширование (например, с использованием HTTP-заголовков Cache-Control) может быть применено для статического контента (изображений, CSS, JavaScript), но для динамических данных требуется более сложное управление, доступное только на серверной стороне.

Для различных частей системы будут использованы разные паттерны кеширования в зависимости от их характера и требований.

**Cache-Aside**

* **Где использовать:** Каталог товаров, справочные данные (статусы заказов, производственные операции).
* **Почему:** Этот паттерн позволяет гибко управлять данными — если они отсутствуют в кеше, приложение загружает их из базы данных и сохраняет в кеш. Это снижает вероятность того, что кеш будет переполнен редко используемыми данными.
* **Преимущества:**
  + Простота реализации.
  + Минимальные затраты на поддержание кеша.
* **Почему не другие паттерны:**
  + **Write-Through:** Подходит для данных, которые часто обновляются, но справочные данные редко меняются.
  + **Refresh-Ahead:** Усложняет управление данными, увеличивает нагрузку из-за проактивного обновления кеша.

#### **Write-Through**

* **Где использовать:** MES - Данные заказов в реальном времени.
* **Почему:** Этот паттерн позволяет синхронизировать кеш и базу данных, обеспечивая консистентность данных. Например, при изменении остатков на складе данные сразу обновляются в кеше.
* **Преимущества:**
  + Высокая консистентность между кешем и базой данных.
  + Уменьшение числа обращений к базе.
* **Почему не другие паттерны:**
  + **Cache-Aside:** Важные данные могут отсутствовать в кеше, что увеличит задержки.
  + **Refresh-Ahead:** Избыточное обновление кеша для редко запрашиваемых данных.

# Чтение списка заказов

На самом деле, мне сложно представить как можно закешировать СПИСОК заказов. Что, если их там тысячи, необходима сортировка, пагинация и фильтрация? Для нормальной работы этого списка нужно скорее оптимизировать БД, нежели кеширование. Но если всё-таки нужно предложить какое-то решение с кешированием, то вот оно:

* Кешируются только новые / активные заказы. Сделаем допущение, что их относительно небольшое количество, и мы можем их фильтровать и сортировать в памяти.
* При запросе списка заказов сервер обращается к кешу (в памяти или Redis). Получает полный список и фильтрует/сортирует, применяет пагинацию, и отдаёт клиенту.
* Заказы при обновлении статуса / при создании заказа - обновляют кеш
* Такой кеш необходимо “прогреть” при старте приложения
* Инвалидировать такой кеш не нужно, т.к. по сути он служит источником данных для пользователя, фоном обновляясь по событию изменения заказа.