**Монолитное приложение:** Интернет-магазин (**E-commerce** платформа).

**Основные возможности приложения:**

* **Управление пользователями**: Регистрация, авторизация, профили пользователей.
* **Каталог товаров**: Просмотр категорий, фильтрация товаров, поиск.
* **Корзина**: Добавление товаров в корзину, подсчет стоимости.
* **Оформление заказа**: Оплата, обработка заказа.
* **Управление заказами**: Слежение за заказом, история заказов.
* **Административная панель**: Управление товарами, категориями, пользователями, заказами.

**Основные компоненты монолитного приложения:**

1. **User Management** (управление пользователями)
2. **Product Catalog** (каталог товаров)
3. **Shopping Cart** (корзина)
4. **Order Processing** (обработка заказов)
5. **Payment Processing** (обработка платежей)
6. **Admin Panel** (панель администратора)

**Шаг 2: Выявление узких мест и потенциальных проблем**

**Проблемы и узкие места, возникающие при высокой нагрузке:**

1. **Обработка заказов**: При большом количестве заказов увеличивается нагрузка на сервис оформления и обработки заказов.
2. **Платежная система**: Платежи могут быть узким местом, особенно при использовании сторонних API для обработки транзакций.
3. **Каталог товаров и поиск**: При большом ассортименте и количестве пользователей поиск и фильтрация товаров могут замедлить работу приложения.
4. **Корзина покупок**: Постоянные изменения корзины у множества пользователей могут привести к перегрузке базы данных.

**Шаг 3: План по разделению монолита на микросервисы**

**Цель:** Обеспечить гибкость, масштабируемость, а также независимую обработку отдельных частей системы.

1. **Определение модулей и сервисов:**
   * **User Service**: Управление пользователями, профилями, авторизацией и регистрацией.
   * **Catalog Service**: Управление товарами и категориями, а также поддержка поиска и фильтрации.
   * **Cart Service**: Корзина покупок, позволяющая добавлять, удалять и изменять товары.
   * **Order Service**: Обработка и управление заказами.
   * **Payment Service**: Обработка платежей, интеграция с платежными шлюзами.
   * **Admin Service**: Управление и мониторинг других сервисов через административную панель.
2. **План по миграции сервисов:**
   * **Шаг 1**: Выделение **User Service** и **Catalog Service**.
     + Эти сервисы относительно независимы, и их выделение минимально повлияет на работу приложения.
     + **Технологии**: Поддержка REST API или gRPC для обмена данными с другими сервисами.
   * **Шаг 2**: Выделение **Cart Service**.
     + Корзина активно используется, поэтому ее вынесение позволит снизить нагрузку на центральную базу данных.
     + **Решение**: Использование Redis или аналогичных решений для хранения данных корзины в памяти.
   * **Шаг 3**: Выделение **Order Service** и **Payment Service**.
     + Разделение заказа и платежей позволит масштабировать процессы независимо, например, для повышения отказоустойчивости при высокой нагрузке.
     + **Решение**: Использование очередей сообщений (например, RabbitMQ) для обработки заказов асинхронно.
   * **Шаг 4**: Выделение **Admin Service**.
     + Этот сервис интегрируется с другими микросервисами и предоставляет административный интерфейс.
     + **Решение**: Подключение с использованием REST API для доступа к каждому сервису.
3. **Настройка взаимодействия между сервисами:**
   * Использование API Gateway для маршрутизации запросов к нужным сервисам.
   * Настройка очередей сообщений для асинхронного взаимодействия, особенно для **Order** и **Payment** сервисов.
4. **Настройка баз данных:**
   * Каждый сервис имеет собственную базу данных для автономной работы, исключая корзину, где данные хранятся временно.
   * **Каталог и пользователи** могут оставаться на общем сервере базы данных с разграничением прав.

**Шаг 4: Размещение кода и документации на GitHub**

1. Разместите базовый код приложения и документацию (README) о процессе миграции и архитектуре приложения.
2. Укажите описание API для каждого микросервиса и примеры его использования.