High-Level Design for distributed online store system prototype

v. 1.0

1. Вступ

Цей документ описує високорівневу архітектуру (HLD) розподіленої системи інтернет-магазину. Система дотримується мікросервісної архітектури і розроблена для підтримки високої доступності та ефективного розподілу навантаження.

2. Архітектура системи

2.1 Компоненти системи

Платформа інтернет магазину складається з кількох сервісів, що працюють разом для забезпечення перегляду продуктів, керування кошиком, обробки замовлень та повідомлень.

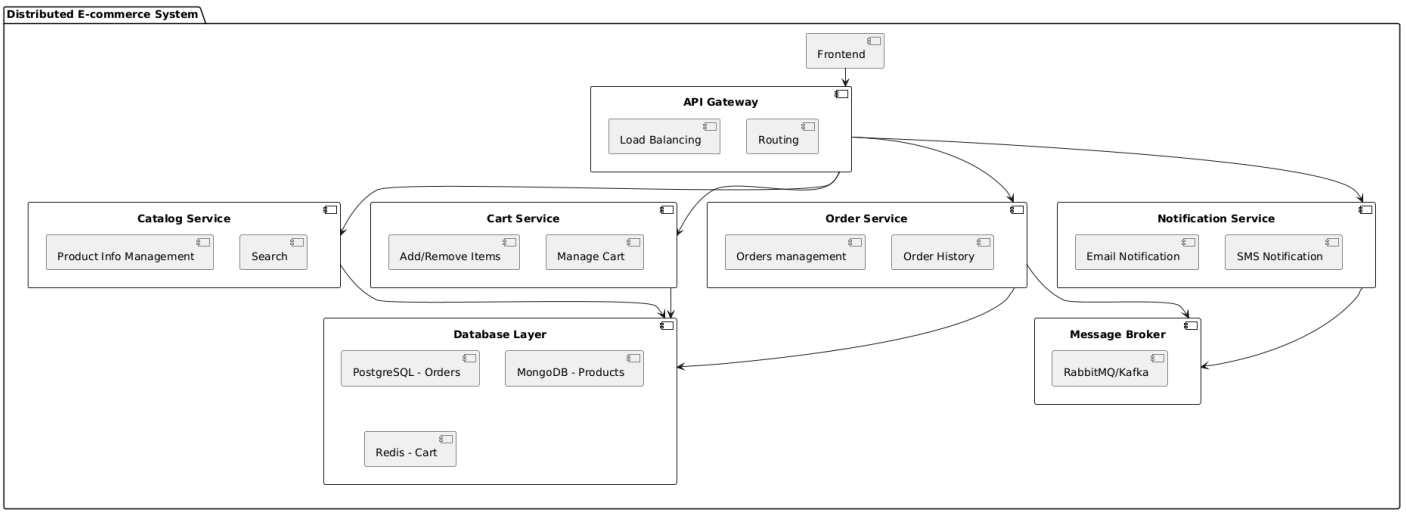
Система складається з таких основних сервісів:

1. **Сервіс каталогу товарів**: керує списками продуктів, категоріями та функціями пошуку.
    a. **Технології**:
        - Node.js (Express)
        - **База даних**: MongoDB (для гнучкого зберігання структурованих та неструктурованих даних).
    b. **API**:
        - GET /products`: Повертає JSON-масив із полями `{id, name, price, description}`.
        - GET /products/{id}`: Повертає JSON-об’єкт із деталями товару (статус 200 або 404).
2. **Сервіс кошика:** дозволяє додавати/видаляти продукти з кошика та ініціює оформлення замовлення.
    a. **Технології**:
        - Node.js (Express)
        - **База даних**: Redis (для швидкого тимчасового зберігання даних).
    b. **API**:
        - `POST /cart/ add` — додати товар у кошик. Приймає `{productId}` і додає товар (статус 200 або 400).
        - `DELETE /cart/{id}` — видалити товар з кошика (статус 200 або 404).

        - `GET /cart` — повертає JSON-масив ідентифікаторів товарів.
        - POST /cart/checkout: Ініціює оформлення замовлення (статус 200 або 503).
3. **Служба замовлення:** обробляє створення замовлень.
    a. **Технології**: Node.js (Express).
        - **База даних**: PostgreSQL (для надійного зберігання транзакційних даних).
    b. **API**:
        - `POST /orders` — створити замовлення. Приймає {items: [id1, id2, ...]} і створює замовлення (статус 200 або 500).
        - `GET /orders/{id}` — отримати деталі замовлення. Повертає JSON-об'єкт із деталями замовлення (статус 200 або 404).
4. **Сервіс нотифікацій**: надсилає сповіщення електронною поштою/SMS про оновлення статусу замовлення.
    a. **Технології**: Node.js.
        - **Черга повідомлень**: RabbitMQ (для асинхронної обробки повідомлень).
    b. **API**:
        - `POST /notifications` — відправити сповіщення (викликається через чергу).

2.2 Діаграма архітектури

Показує взаємодію Catalog, Cart, Order і Notification Service із MongoDB, Redis, PostgreSQL і RabbitMQ.



3. Технологічний стек

- **Мови програмування**: Node.js 20.x.
- **Бази даних**:
    - PostgreSQL — для надійного зберігання замовлень (обґрунтування: транзакційна цілісність).
    - Redis — для швидкого доступу до даних кошика (обґрунтування: висока продуктивність).
    - MongoDB — для каталогу товарів.
- **Інструменти**:
    - Docker — для контейнеризації.
    - Kubernetes — для оркестрації.
    - RabbitMQ — для асинхронної комунікації.
    - Prometheus + Grafana — для моніторингу метрик (запити, помилки).
    - Nginx — для балансування навантаження.

Бібліотеки:

- async-retry: для повторних спроб.
- circuit-breaker-js: для захисту від збоїв.

4. Сценарій роботи

4.1 Основний сценарій

1. **Користувач отримує список товарів (GET /products)**
    - **Опис**: користувач через Frontend надсилає запит до Catalog Service, щоб отримати список товарів.
    - **Технічна деталь**: Catalog Service звертається до MongoDB, щоб отримати дані про товари.
    - **Результат**: повертається JSON-масив товарів (статус 200). Користувач бачить список товарів і може вибрати, що додати в кошик.
2. **Додає товар у кошик (POST /cart/add)**
    - **Опис**: користувач додає обраний товар у кошик через запит до Cart Service.
    - **Технічна деталь**: Cart Service записує товар у Redis.
    - **Результат**: повертається підтвердження (статус 200). Товар додається в кошик, і користувач може продовжити покупки або оформити замовлення.
3. **Оформляє замовлення (POST /cart/checkout)**
    - **Опис**: користувач ініціює оформлення замовлення через Cart Service, який обробляє цей запит.

- **Підетапи**:
  1. **Cart Service викликає Order Service з retry і Circuit Breaker**
     - **Технічна деталь:** Cart Service використовує бібліотеки async-retry для повторних спроб і circuit-breaker-js для захисту від перевантаження.
     - **Результат:** якщо Order Service недоступний, Cart Service намагається повторити запит, а якщо збої тривають, Circuit Breaker зупиняє запити.

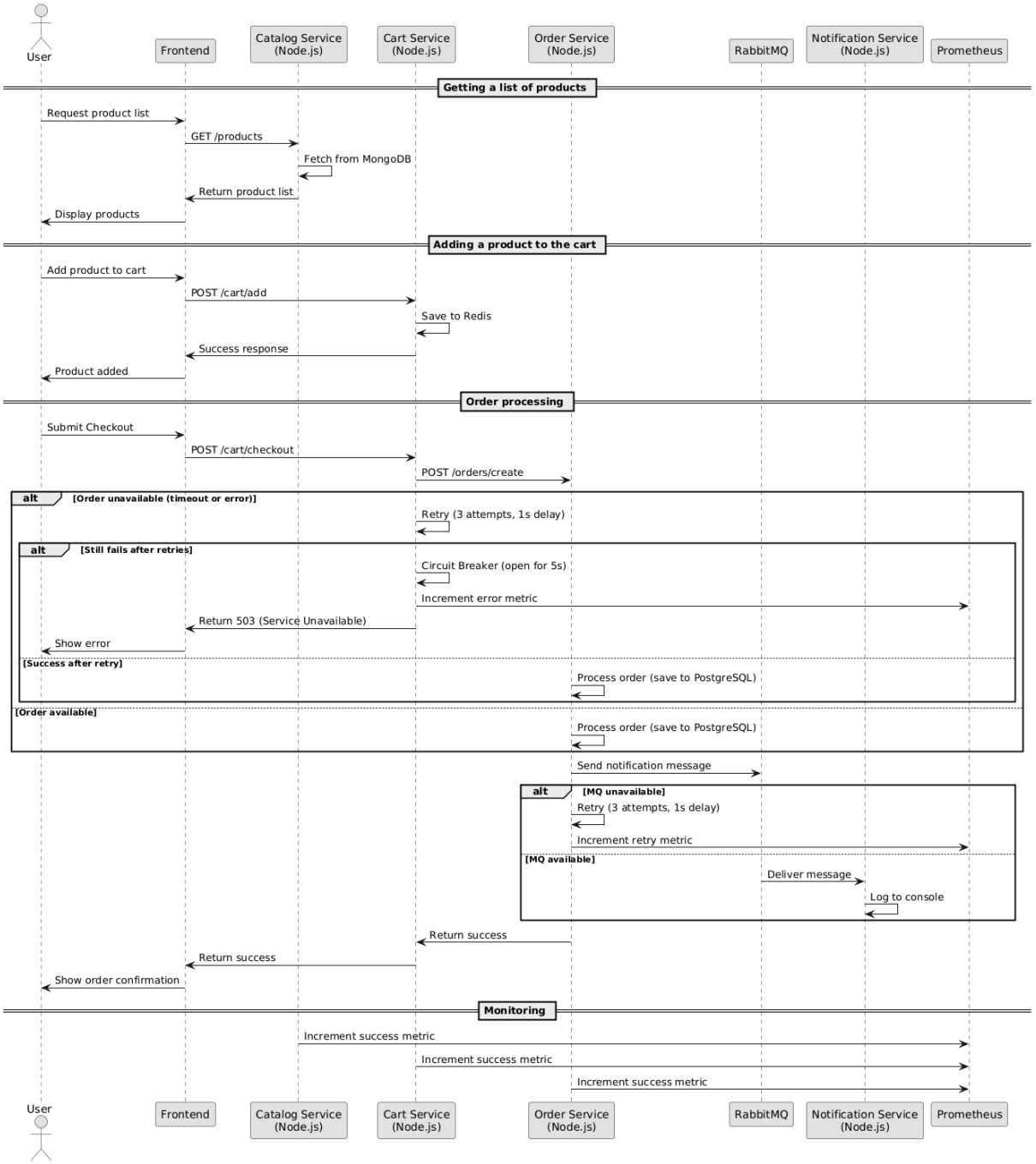
  2. **Order Service записує замовлення і надсилає повідомлення в RabbitMQ (з retry):**

     - **Технічна деталь:** Order Service зберігає замовлення в PostgreSQL і надсилає повідомлення в RabbitMQ, використовуючи retry (наприклад, 3 спроби), якщо брокер недоступний.
     - **Результат:** замовлення збережено, повідомлення надіслано.

  3. **Notification Service виводить повідомлення в консоль**

     - **Технічна деталь:** Notification Service слухає RabbitMQ і просто логує повідомлення (спрощена реалізація).
     - **Результат:** система підтверджує, що нотифікація отримана (у консолі).

Процес оформлення замовлення з урахуванням можливих збоїв і механізмів їх обробки:



Сценарій: Повний цикл взаємодії користувача з системою.

Потік:

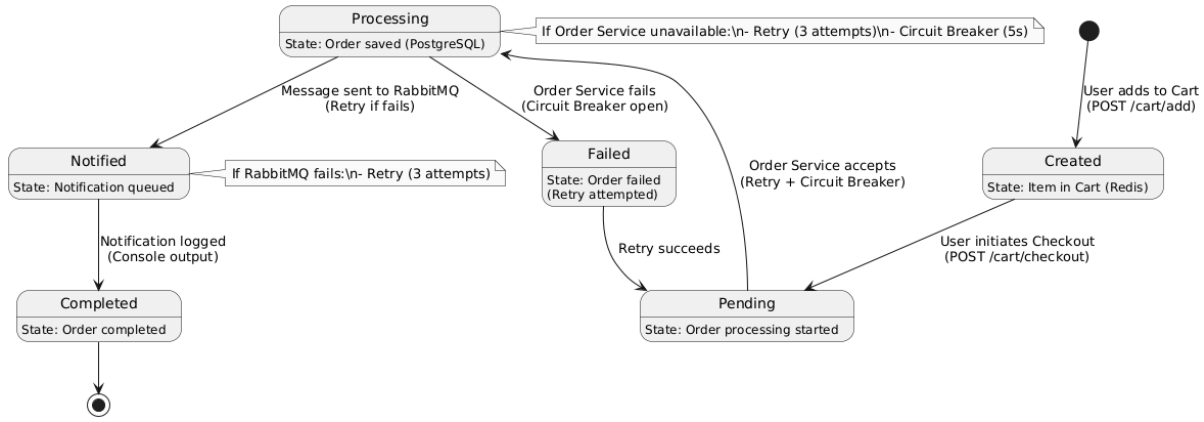
1. **Отримання списку товарів**:
    - User запитує товари через Frontend.
    - Catalog Service повертає дані з MongoDB.
    - Користувач бачить список товарів.
2. **Додавання товару в кошик**:
    - User додає товар у кошик через Frontend.

- Cart Service зберігає дані в Redis.
- Користувач отримує підтвердження.

3. **Оформлення замовлення**:
   - User ініціює checkout.
   - Cart Service викликає Order Service із retry і Circuit Breaker.
   - Order Service зберігає замовлення (PostgreSQL) і надсилає повідомлення в RabbitMQ (з retry).
   - Notification Service виводить повідомлення в консоль.
   - Prometheus фіксує метрики (успіх/помилки) для всіх сервісів.

Результат: користувач завершує покупку або отримує повідомлення про помилку.

Діаграма станів замовлення



Стани:

1. **Created**: Замовлення починається, коли користувач додає товар у кошик (зберігається в Redis через Cart Service).
2. **Pending**: Користувач ініціює checkout (POST /cart/checkout), і процес обробки починається.
3. **Processing**: Order Service приймає замовлення, зберігає його в PostgreSQL і активує механізми retry і Circuit Breaker.
4. **Failed**: Замовлення не вдалося обробити (наприклад, через збій Order Service або відкритий Circuit Breaker).
5. **Notified**: Повідомлення про замовлення відправлено в RabbitMQ (з retry при збоях).
6. **Completed**: Notification Service обробляє повідомлення (виводить у консоль), і замовлення вважається завершеним.