Микросервисная архитектура приложения для подсчета калорий и трекинга тренировок

1. Фронтенд (React/Vue.js + TypeScript)

Фронтенд-приложение отвечает за взаимодействие с пользователем и визуализацию данных. Основные функции:

- Отображение дневника питания с возможностью добавления продуктов

- Трекер тренировок с выбором упражнений и вводом параметров

- Графики и статистика по потреблению калорий и активности

- Настройки профиля пользователя

Фронтенд общается исключительно через API Gateway, отправляя REST-запросы и получая данные в формате JSON. Для авторизации используется JWT-токен, который сохраняется в cookies.

2. API Gateway (Node.js + Express)

API Gateway является единой точкой входа для всех запросов от фронтенда. Его основные задачи:

- Маршрутизация запросов к соответствующим микросервисам

- Балансировка нагрузки между экземплярами сервисов

- Агрегация данных из нескольких сервисов для сложных запросов

- Обеспечение безопасности (валидация JWT-токенов, ограничение запросов)

Gateway знает обо всех сервисах в системе и управляет их взаимодействием. При получении запроса он определяет, какой сервис должен его обработать, и перенаправляет запрос.

3. Сервис аутентификации (Node.js + JWT)

Сервис отвечает за:

- Регистрацию новых пользователей с валидацией данных

- Аутентификацию по email/паролю

- Генерацию и валидацию JWT-токенов

- Управление сессиями пользователей

- Восстановление пароля через email

При регистрации сервис сохраняет данные пользователя в PostgreSQL и создает запись в Redis для быстрого доступа к данным сессии. Для хранения паролей используется bcrypt с солью.

4. Сервис пользователей (Node.js + PostgreSQL)

Основные функции:

- Хранение и обновление профильной информации (рост, вес, цели)

- Расчет базового метаболизма (BMR) и рекомендуемой нормы калорий

- Управление настройками пользователя

- Предоставление данных для статистики и отчетов

Сервис тесно интегрирован с сервисом аутентификации и использует ту же базу данных PostgreSQL, но работает с другими таблицами.

5. Сервис питания (Node.js + PostgreSQL)

Отвечает за все, что связано с учетом потребляемых калорий:

- Хранение базы продуктов и блюд

- Добавление и редактирование приемов пищи

- Расчет питательной ценности (калории, БЖУ)

- Поиск продуктов по названию и категории

- Генерация отчетов по потреблению нутриентов

Сервис использует PostgreSQL для хранения структурированных данных о продуктах и приемах пищи. Для сложных запросов (например, поиск по названию) используется полнотекстовый поиск PostgreSQL.

6. Сервис тренировок (Node.js + MongoDB)

Функционал:

- Хранение данных о тренировках и упражнениях

- Расчет сожженных калорий на основе типа активности и параметров пользователя

- Формирование программ тренировок

- Отслеживание прогресса (увеличение весов, повторений и т.д.)

- Интеграция с внешними трекерами (Google Fit, Apple Health)

MongoDB выбрана для этого сервиса из-за гибкой схемы данных, что позволяет легко хранить разнородные данные о различных типах тренировок.

7. Сервис уведомлений (Node.js + RabbitMQ)

Основные задачи:

- Отправка email-уведомлений (напоминания о приемах пищи, тренировках)

- Push-уведомления в мобильное приложение

- Напоминания о достижении целей или отклонении от плана

- Уведомления о новых функциях и акциях

Сервис работает асинхронно, получая сообщения через RabbitMQ от других сервисов. Для отправки email используется SendGrid или аналогичный сервис.

8. Брокер сообщений (RabbitMQ)

Обеспечивает асинхронное взаимодействие между сервисами:

- Передача сообщений о событиях (новая тренировка, изменение веса и т.д.)

- Очередь задач для сервиса уведомлений

- Логирование важных событий для мониторинга

9. Мониторинг (Prometheus + Grafana)

Система мониторинга собирает:

- Метрики производительности каждого сервиса

- Время ответа на запросы

- Количество ошибок

- Использование ресурсов (CPU, память, диски)

- Бизнес-метрики (активные пользователи, количество тренировок и т.д.)

Данные визуализируются в Grafana, что позволяет оперативно реагировать на проблемы и анализировать нагрузку.

Взаимодействие сервисов на примере типичных сценариев

Сценарий 1: Добавление приема пищи

1. Пользователь через фронтенд добавляет продукт в свой дневник питания

2. Фронтенд отправляет POST-запрос в API Gateway

3. Gateway проверяет JWT-токен и перенаправляет запрос в сервис питания

4. Сервис питания:

   - Ищет продукт в базе данных PostgreSQL

   - Создает запись о приеме пищи

   - Рассчитывает калории и нутриенты

   - Отправляет сообщение в RabbitMQ о новом приеме пищи

5. Сервис уведомлений получает сообщение и проверяет, не превышает ли пользователь дневную норму калорий

6. Если норма превышена, сервис уведомлений отправляет предупреждение

7. Сервис пользователей обновляет статистику пользователя

8. Обновленные данные возвращаются через Gateway к фронтенду

Сценарий 2: Завершение тренировки

1. Пользователь отмечает завершение тренировки во фронтенде

2. Фронтенд отправляет данные о тренировке в API Gateway

3. Gateway перенаправляет запрос в сервис тренировок

4. Сервис тренировок:

   - Сохраняет данные в MongoDB

   - Рассчитывает сожженные калории на основе веса пользователя и интенсивности тренировки

   - Отправляет сообщение в RabbitMQ о завершенной тренировке

5. Сервис уведомлений может отправить поздравление с завершением тренировки

6. Сервис пользователей обновляет статистику активности пользователя

7. Обновленные данные возвращаются пользователю

Заключение

Предложенная микросервисная архитектура обеспечивает гибкость, масштабируемость и отказоустойчивость приложения для подсчета калорий и трекинга тренировок. Разделение функционала на отдельные сервисы позволяет:

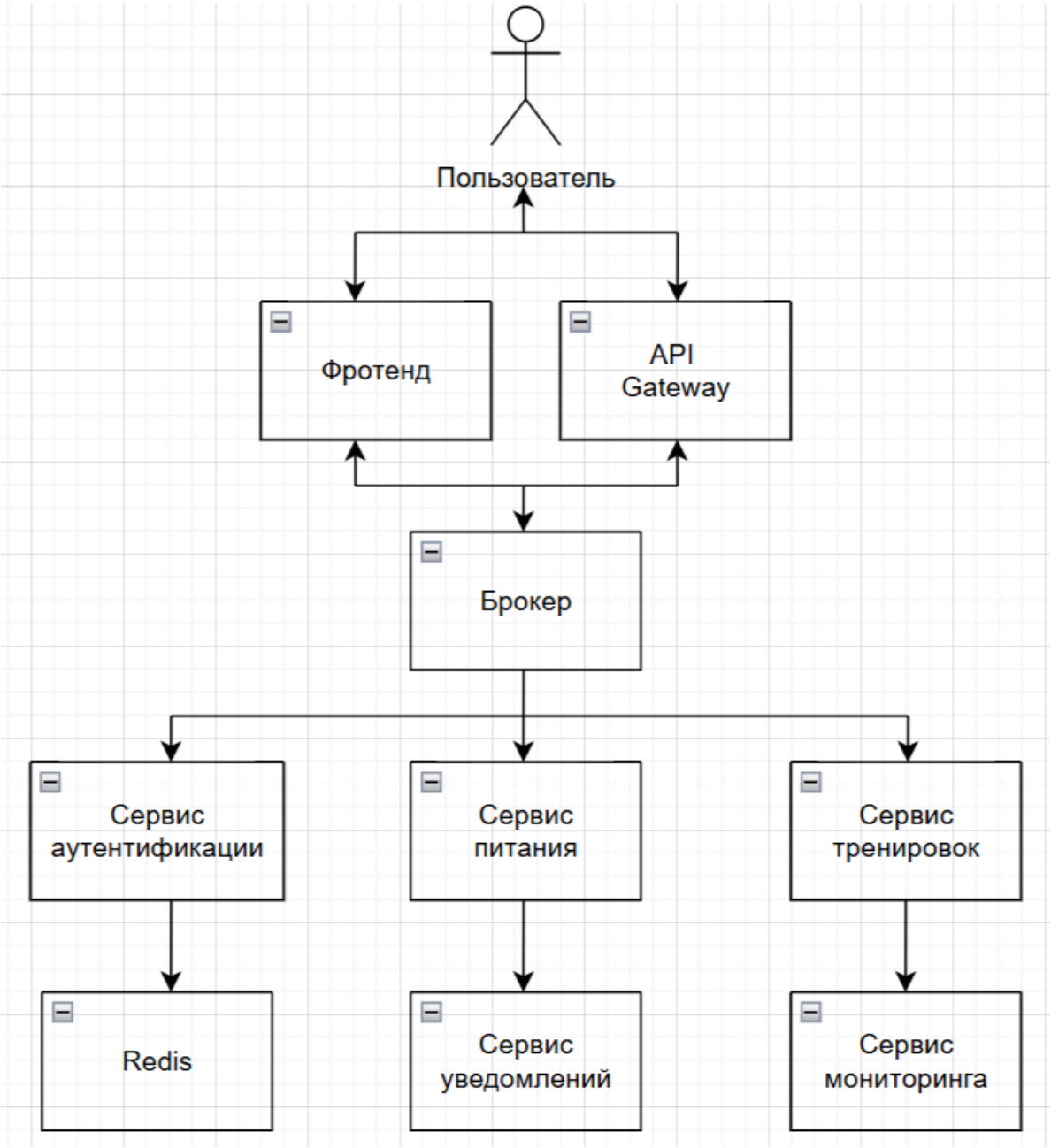
- Независимо масштабировать компоненты под нагрузкой

- Использовать оптимальные технологии для каждой задачи

- Легко добавлять новый функционал без переписывания всей системы

- Обеспечивать высокую доступность критических компонентов

Асинхронное взаимодействие через брокер сообщений уменьшает задержки и повышает отказоустойчивость системы. Мониторинг всех компонентов позволяет оперативно выявлять и устранять проблемы.


Пользователь
Фротенд
API
Gateway
Брокер
Сервис
аутентификации
Сервис
питания
Сервис
тренировок
Redis
Сервис
уведомлений
Сервис
мониторинга