

Программный проект

FoodHelper

Поиск готовой еды по КБЖУ

Выполнил:
студент группы БПМИ238
Майорова Марина
Александровна

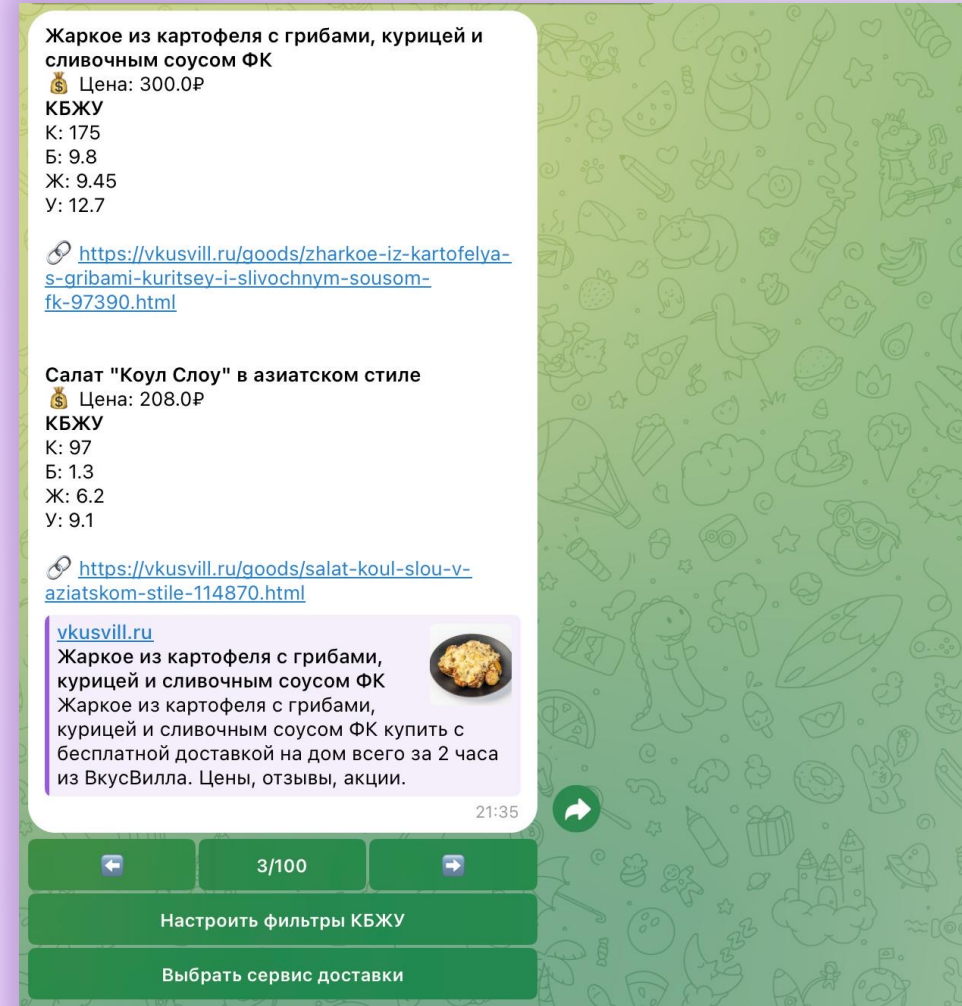
Принял руководитель проекта:
Иванов Андрей Александрович
Доцент
Базовая кафедра Т-Банка



Telegram-бот



Пример карточки товара





Актуальность

664 млрд ₽ **56%**

рынок доставки еды в
России +27% за год

Data Insight, 2023)

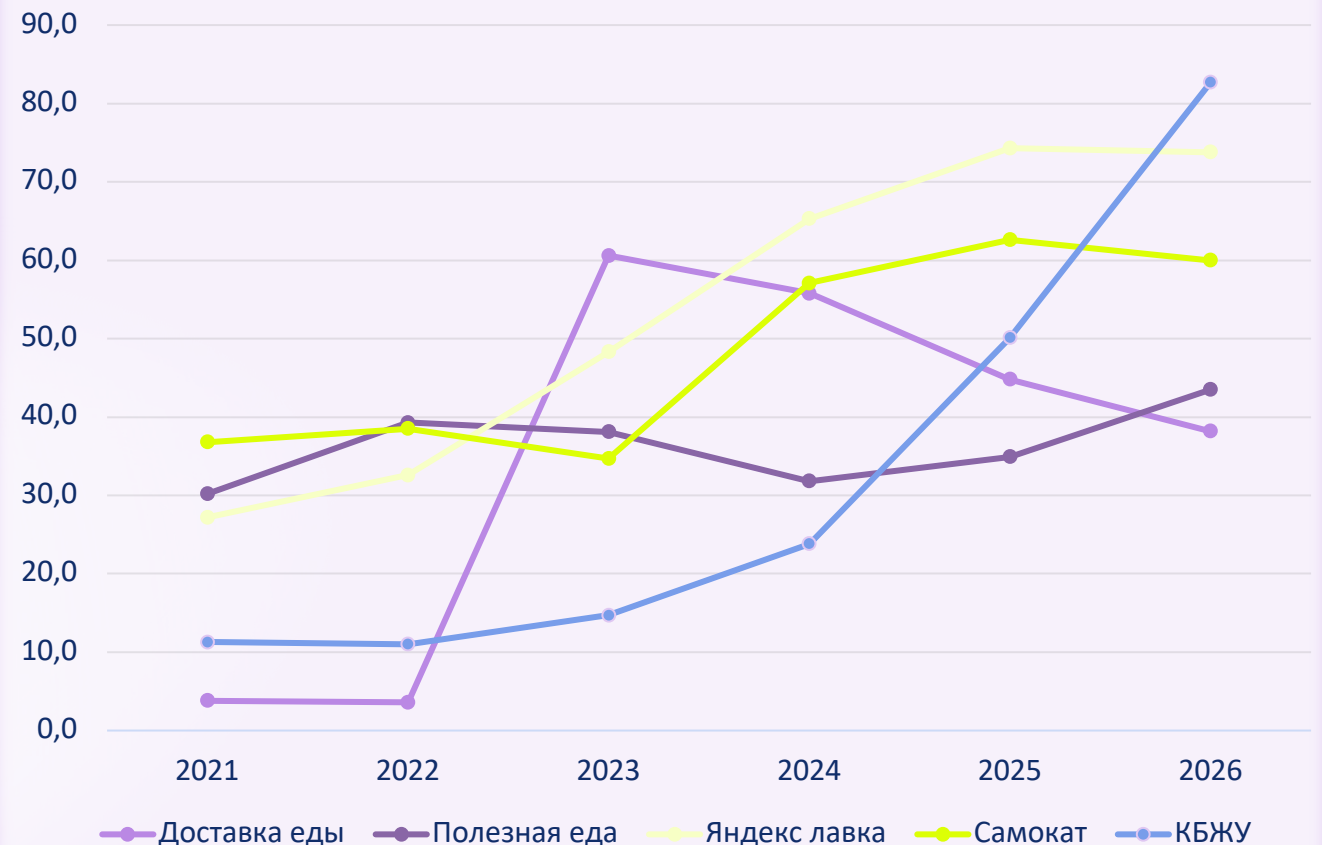
россиян следят за
питанием

(ВЦИОМ, 2023) +7
п.п. за 9 лет

Проблема

- в доставках есть карточки товаров, но **нет удобного поиска по КБЖУ**
- пользователю приходится **вручную** сравнивать блюда
- особенно неудобно искать сразу по **нескольким** сервисам

Спрос на доставку и полезную еду

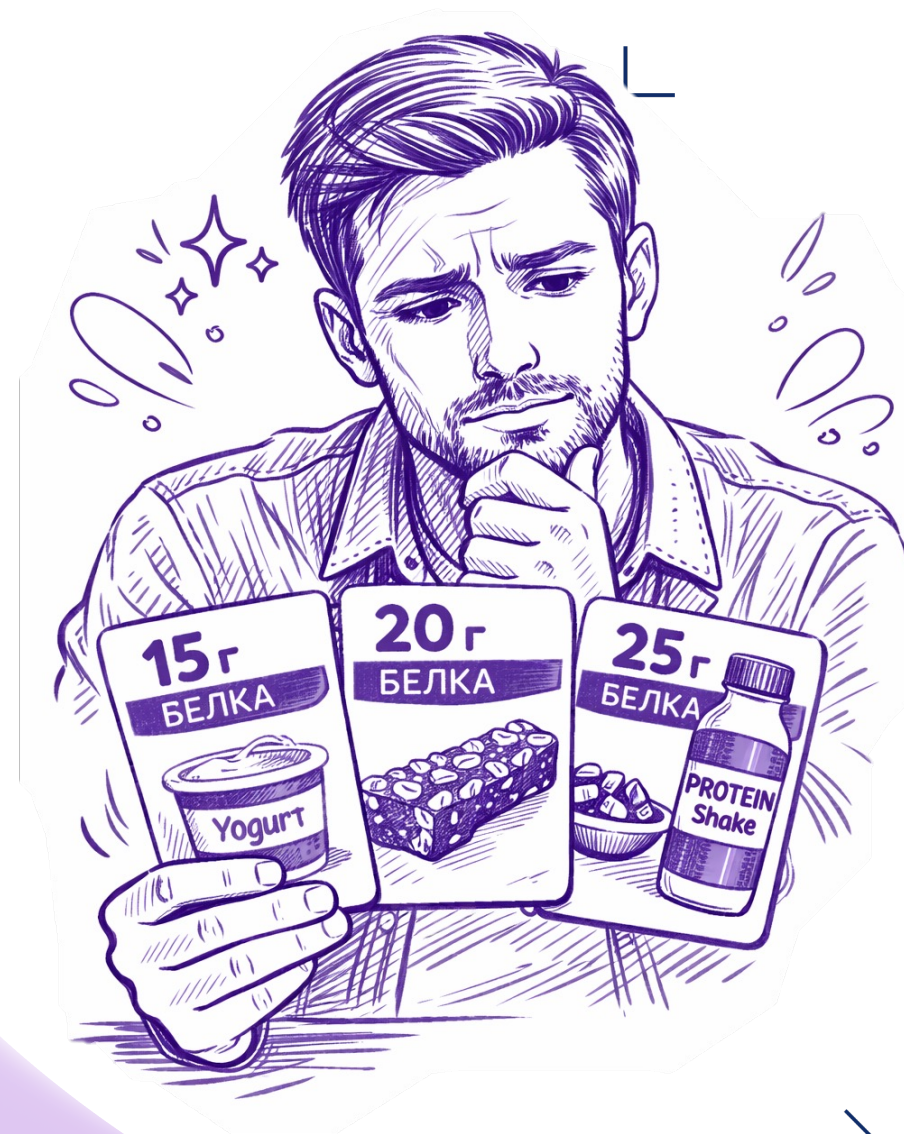


Цель

Предоставить доступ к поиску товаров в соответствии с КБЖУ через backend API, пригодный для разных клиентов.

Задачи работы

1. Спроектировать БД под фильтрацию и поиск
2. Реализовать API поиска и импорта
3. Сделать ранжирование текстового запроса
4. Настроить инфраструктуру
5. Обеспечить мониторинг и документацию API
6. Покрыть ключевые сценарии тестами



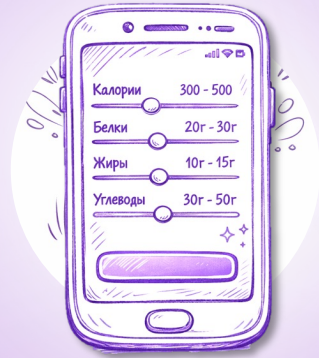
Анализ существующих аналогов

FoodHelper закрывает нишу, которую не занимают ни трекеры калорий, ни сервисы доставки по отдельности.

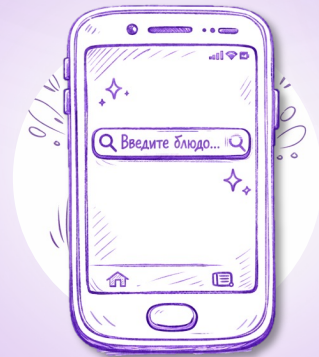
Критерий	MyFitnessPal / YAZIO	ВкусВилл / Яндекс Еда	FoodHelper
Фильтрация по диапазонам КБЖУ	—	—	✓
Актуальный каталог доставки	—	✓	✓
Прямая ссылка на покупку	—	✓	✓
Мультисервисный поиск	—	—	✓
Telegram-интерфейс	частично / —	частично	✓
Трекинг суточного рациона	✓	—	—
Доступность в России	ограниченно / ✓	✓	✓



Задать диапазоны
КБЖУ



Поиск
по названию



Выбрать
службы доставки



Карточки: цена,
КБЖУ, ссылка

Основные функции системы

- **КБЖУ** — фильтрация по калориям, белкам, жирам и углеводам одновременно: min и max.
- **Multi** — мультисервисность: поиск сразу по нескольким службам доставки.
- **Text** — текстовый поиск: по названию и составу блюда с AND-семантикой токенов.
- **Card** — карточка товара: название, цена, КБЖУ, служба доставки, прямая ссылка на покупку.
- **Auto** — автообновление каталога через парсер по расписанию.
- **2 Clients** — Telegram-бот + Web SPA через единый REST API.



Архитектура системы

Центральный элемент

FoodHelper API — единая точка доступа для всех клиентов. Обрабатывает запросы поиска, управляет импортом данных и предоставляет метрики.

Клиенты

Telegram Bot — интерактивный поиск через мессенджер

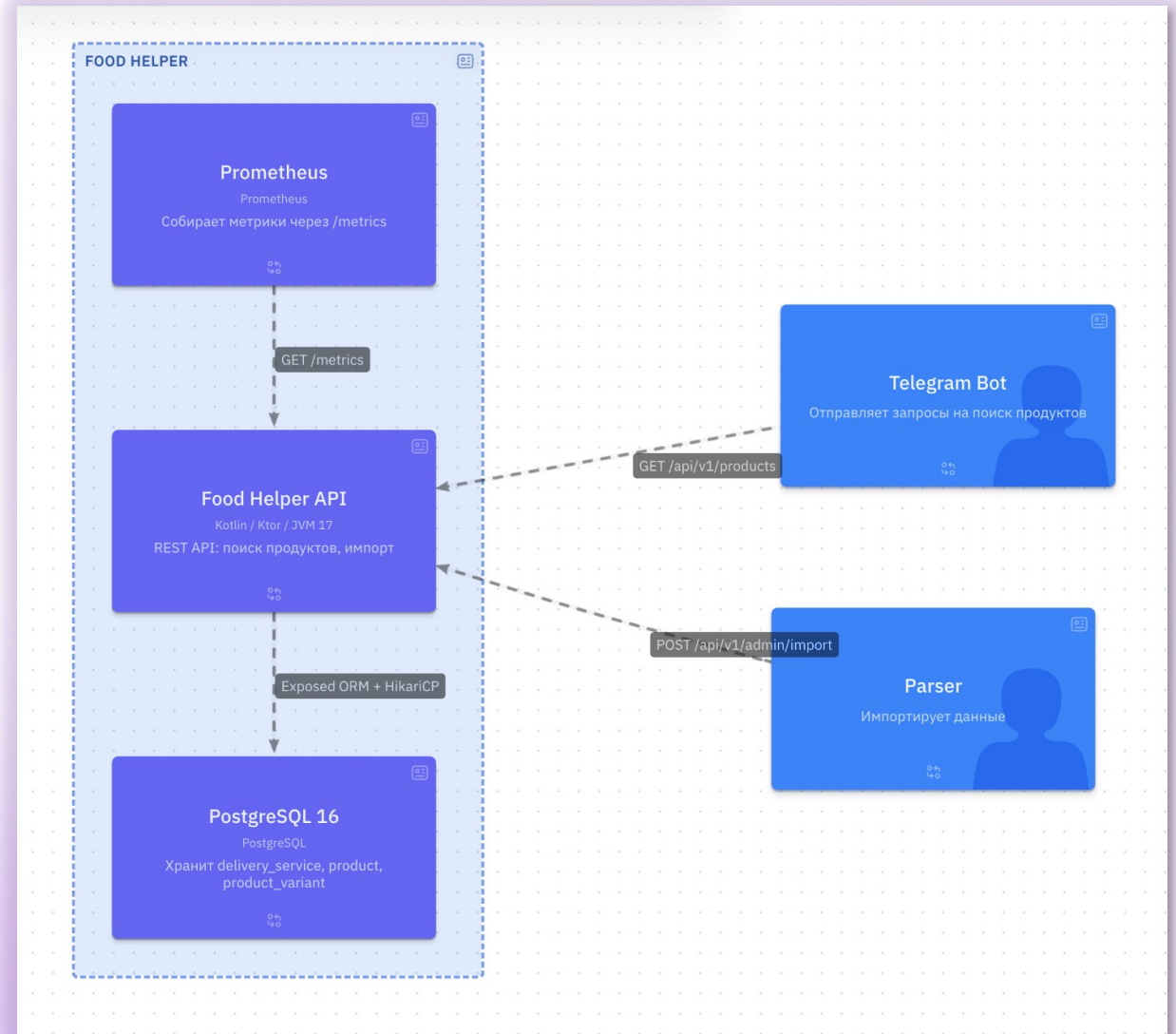
Web SPA — веб-интерфейс

Инфраструктура

Parser — автообновление каталога по расписанию

PostgreSQL — хранение товаров и нутриентных данных

Prometheus + Grafana — мониторинг и алертинг





Основные endpoints backend API

Public API

- GET /api/v1/products — поиск и фильтрация
- GET /api/v1/delivery-services — список сервисов
- /health, /swagger

Admin API → protected by X-API-Key

- POST /api/v1/admin/import — импорт товаров от парсера
- POST /api/v1/admin/delivery-services — создание службы доставки
- PUT /api/v1/admin/delivery-services/{id} — обновление службы доставки
- DELETE /api/v1/admin/delivery-services/{id} — удаление службы доставки

Пример запроса

GET /api/v1/products?q=куриное филе&minProtein=15&maxCalories=400

```
items: [
  {
    "id": 309,
    "name": "Куриное филе в хрустящей корочке с кисло-сладким соусом",
    "url": "https://vkusvill.ru/goods/kurinoe-file-v-khrustyashchey-korochke-s-kislo-sladkim-sou-93949.html",
    "price": 260,
    "currency": "RUB",
    "deliveryService": {
      "id": 2,
      "code": "VKUSVILL",
      "name": "Вкусвилл",
      "siteUrl": "https://vkusvill.ru",
      "logoUrl": null,
      "active": true
    },
    "variants": [
      {
        "id": 3841,
        "manufacturer": "ООО КУЛИНАРНАЯ ФАБРИКА \"КОРОЛЕВСКИЙ ВКУС\"",
        "composition": "куриное филе в хрустящей корочке (филе куриной грудки охлажденное, соль пищевая, панировка оригинальная (концентрат пищевой) (мука пшеничная хлебопекарная высший сорт, кукурузный крахмал, соль пищевая, специи молотые (перец черный, паприка), разрыхлитель -гидрокарбонат натрия (сода пищевая), уксус пищевой сухой), масло подсолнечное рафинированное дезодорированное), соус (сахар, вода питьевая, перец чили свежий, соль пищевая, чеснок свежий, загуститель -кукурузный крахмал, регулятор кислотности -уксусная кислота ледяная, загуститель -ксантановая камедь).",
        "weight": 200,
        "nutrients": {
          "calories": 230,
          "protein": 17.0,
          "fat": 9.6,
          "carbs": 19.0
        }
      }
    ]
  }
]
```



Ранжирование поисковой выдачи



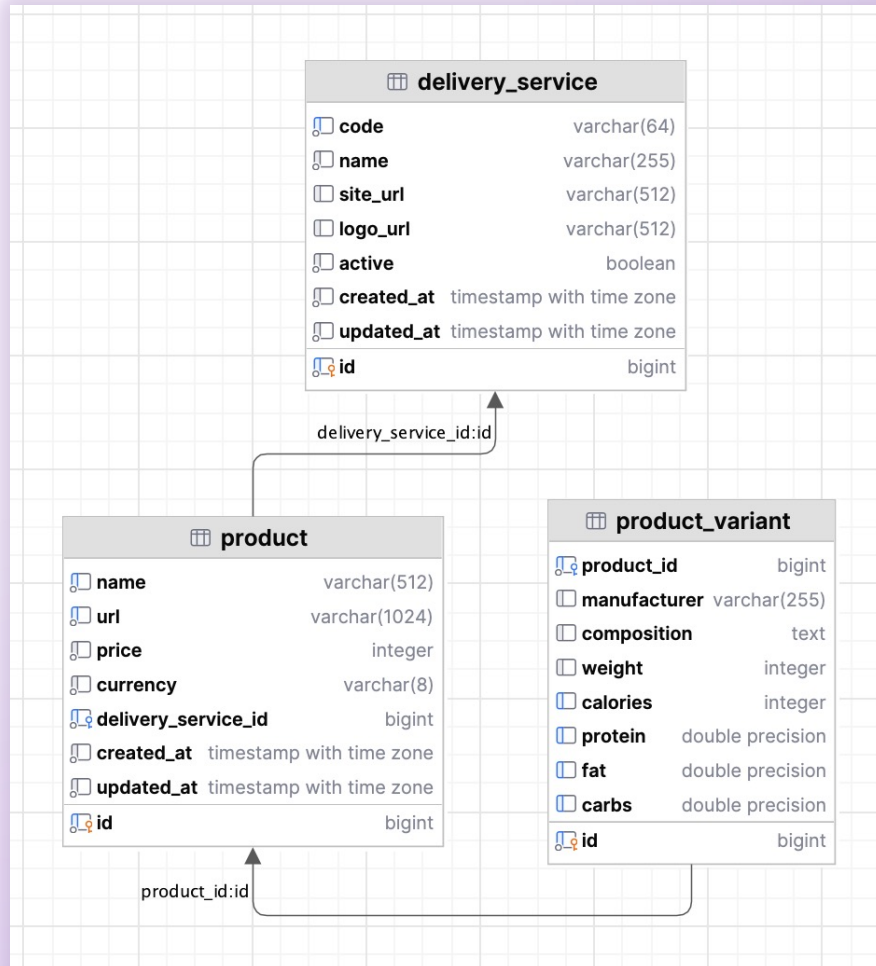
Алгоритм обеспечивает релевантную выдачу: совпадения в названии ценятся выше, чем в составе продукта.

Шкала рангов

Ранг	Условие	Пример
100	Полная фраза в name	<u>Куриное филе</u> на гриле
80	Все токены в name	<u>Филе куриное</u> нежное
50	Часть в name, часть в composition	<u>Филе</u> на гриле + « <u>куриное</u> мясо»
30	Все токены только в composition	Боул протеиновый + « <u>куриное филе</u> »



Модель данных и выбор PostgreSQL



Нормализация (3НФ)

- Данные служб доставки вынесены в отдельную таблицу
- Нутриентные варианты — в `product_variant`
- Исключено дублирование сведений о сервисах и вариантах

Производительность

- GIN + `pg_trgm` для полнотекстового поиска
- B-tree и составные индексы для диапазонов КБЖУ
- Уникальный ключ для идемпотентного импорта (`delivery_service_id, url`)
- Upsert через `ON CONFLICT DO UPDATE`

+ Swagger + мониторинг + health

Monitoring



Prometheus + Grafana

latency, RPS

Пустые результаты поиска
импорт товаров

Swagger



OpenAPI 3.0

Интерактивное

тестирование

Единый контракт для
клиентов

Reliability

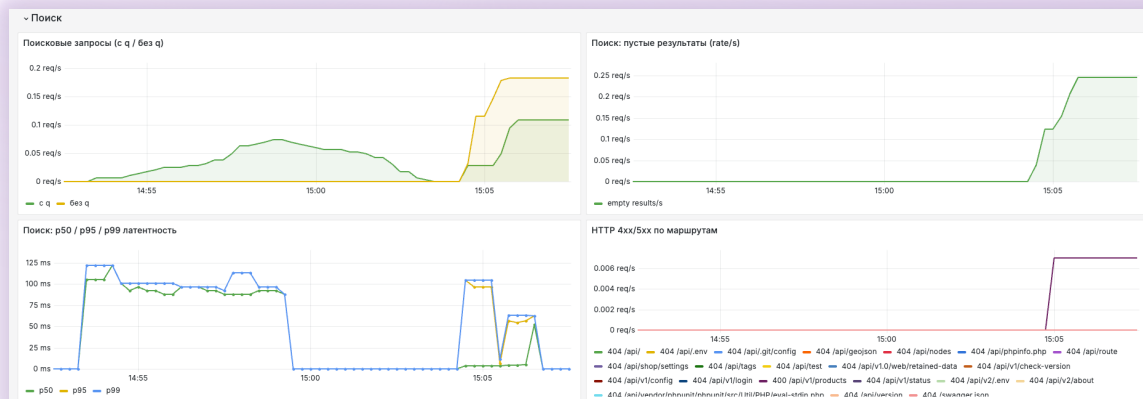


/health

/ready

requestId

Единый формат ошибок



```
{
  "requestId": "a764ee23-5213-4488-811f-c4b00ae26dcc",
  "code": "VALIDATION_ERROR",
  "message": "calories must be >= calories(min)",
  "status": 400,
  "path": "/api/v1/products",
  "timestamp": "2026-04-12T10:50:26.324171714Z",
  "details": null,
  "fieldErrors": null
}
```


Тестовое покрытие

Тест-класс	Тестов	Что проверяется
ImportApiTest.kt	12	HTTP-интеграция импорта, аутентификация, ошибки
QueryTokenizerTest.kt	11	Парсинг и токенизация query-параметров
ProductImportServiceTest.kt	24	Валидация батчей, дедупликация, коды ошибок
RankingTest.kt	12	Алгоритм ранжирования (100 / 80 / 50 / 30)
VariantFilteringTest.kt	12	Фильтрация по диапазонам КБЖУ

71

тест

**unit +
интеграционные**

0

failures

Результаты



REST API (ядро)

- GET /api/v1/products
- Алгоритм ранжирования
- Идемпотентный импорт батчами до 500 товаров
- OpenAPI 3.0 + Swagger UI
- 71 автотест, 0 failures



Инфраструктура

- Docker Compose: 7 сервисов, запуск одной командой
- Prometheus + Grafana мониторинг
- Планировщик парсера по расписанию
- Развёртывание на VPS Ubuntu 22.04



Каталог и данные

- 1000 товаров из ВкусВилл и Яндекс Еды
- Автообновление каталога каждые 6–24 часа
- 7 Liquibase-миграций, схема под версионным контролем



Дополнительно

- Web SPA — второй клиент к тому же API
- Подтверждена универсальность API-контракта



Направления дальнейшего развития

Расширение каталога

1

подключение новых служб доставки через admin API
без изменений кода backend

2

Персонализация

профиль пользователя
цели по КБЖУ
история запросов

3

Уведомления

подписка на новые товары
уведомления по сохранённым фильтрам

4

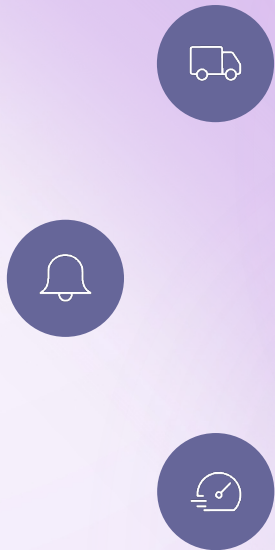
Трекинг рациона

интеграция с дневником питания
добавление блюда прямо из бота

5

Производительность

Redis для кэширования
Elasticsearch при росте каталога

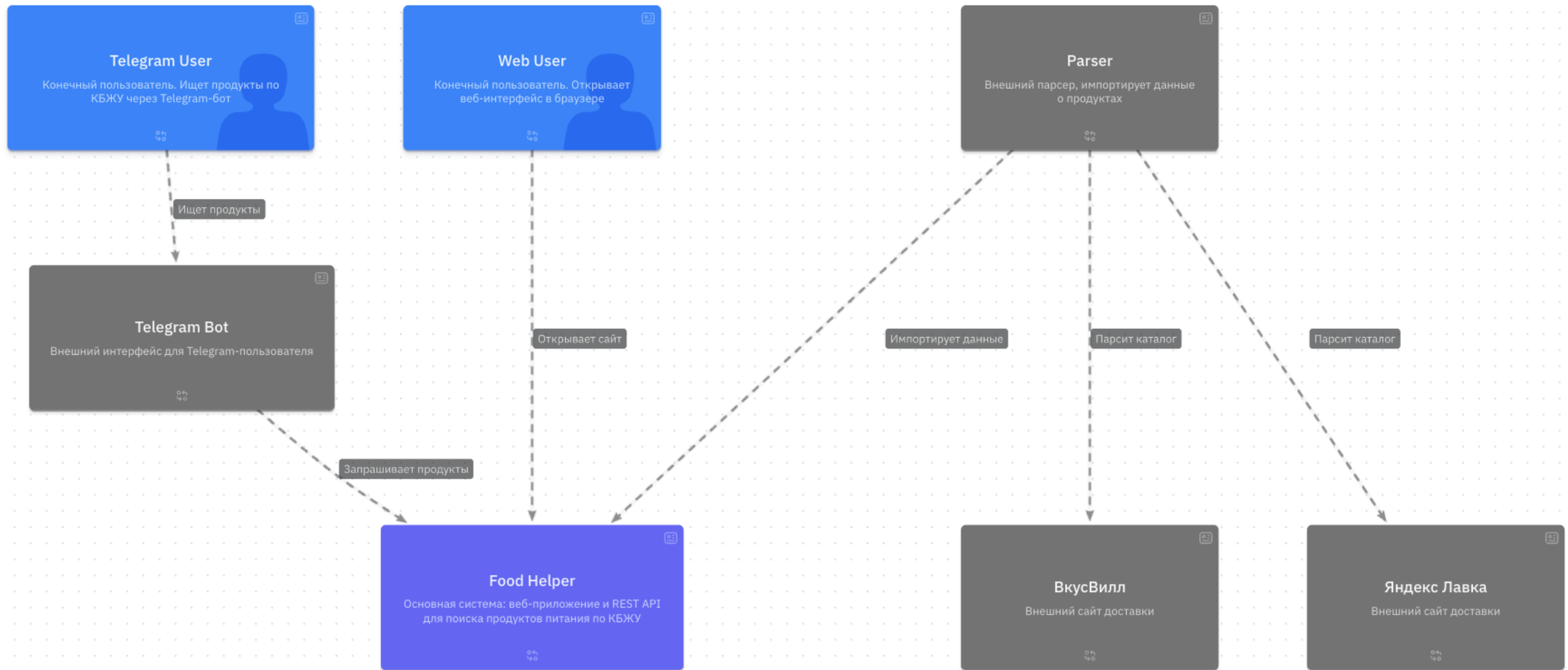


ОКН





Приложения. C4 L1 System Context



+ Приложения. Примеры запросов

```
curl -X 'GET' \
```

```
'http://158.160.201.194/api/v1/delivery-services' \
-H 'accept: application/json'
```

```
[
  {
    "id": 1,
    "code": "YANDEX_EATS",
    "name": "Яндекс Еда",
    "siteUrl": "https://eda.yandex.ru",
    "logoUrl": null,
    "active": true
  },
  {
    "id": 2,
    "code": "VKUSVILL",
    "name": "Вкусвилл",
    "siteUrl": "https://vkusvill.ru",
    "logoUrl": null,
    "active": true
  }
]
```

```
curl -X 'GET' \
```

```
api/v1/products?deliveryServiceIds=1%2C2&page=0&size=20&sort=name%2Ca
sc&maxCalories=100'
```

```
{
  "items": [
    {
      "id": 482,
      "name": "Борщ \\"Вегетарианский\\"",
      "url": "https://vkusvill.ru/goods/borshch-vegetarianskiy-20171.html",
      "price": 180,
      "currency": "RUB",
      "deliveryService": {
        "id": 2,
        "code": "VKUSVILL",
        "name": "Вкусвилл",
        "siteUrl": "https://vkusvill.ru",
        "logoUrl": null,
        "active": true
      },
      "variants": [
        {
          "id": 4078,
          "manufacturer": "ООО \\"ГЛОБАЛ ФУД\\"",
          "composition": "вода питьевая, свекла столовая свежая,картофель свежий, пассеровка томат
ная (вода питьевая, томатная паста концентрированная (томаты свежие, вода питьевая, соль пищевая),
масло подсолнечное рафинированное дезодорированное, уксус столовый 9%), капуста белокочанная свежа
я, фасоль консервированная (фасоль красная,вода питьевая,соль пищевая ), морковь столовая свежая,
лук репчатый свежий, сахар, сок свекольный (свекла столовая свежая), приправа овощная универсальна
я (соль пищевая, сахар, экстракт дрожжей, лук репчатый молотый, морковь столовая сушеная, пастерна
к корень сушеный, петрушка сушеная, куркума молотая, чеснок сушеный, перец черный молотый),чеснок
сушеный гранулированный , соль пищевая, лавровый лист, перец черный молотый.",
          "weight": 390,
          "nutrients": {
            "calories": 87,
            "protein": 0.7,
            "fat": 1.4,
            "carbs": 17.9
          }
        },
        {
          "id": 4079,
```

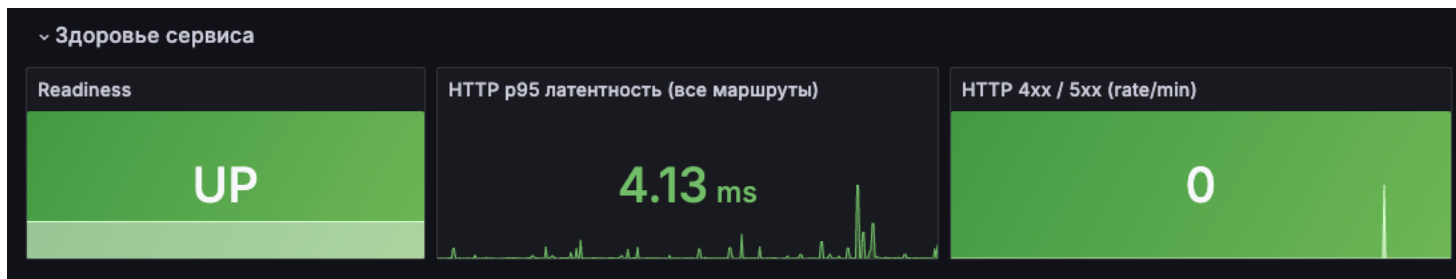
```
curl -X 'POST' \
```

```
/api/v1/admin/import
-H 'accept: application/json' \
-H 'X-Api-Key: ' \
-H 'Content-Type: application/json' \
-d '{
  "deliveryServiceCode": "VKUSVILL",
  "items": [
    {
      "name": "Молоко 1л",
      "url":
"https://vkusvill.ru/goods/12345",}
  ]
}
```

```
{ "requestId": "a32eb237-8b5a-41af-907c-
5895702c54d7",
  "code": "BAD_REQUEST",
  "message": "Failed to convert request body to
class org.example.api.dto.ParserImportRequest",
  "status": 400,
  "path": "/api/v1/admin/import",
  "timestamp": "2026-04-
12T19:15:35.480725126Z",
  "details": null,
  "fieldErrors": null
}
```




Мониторинг. Grafana



Readiness

- показывает, доступен ли backend
- UP = сервис готов принимать запросы
- DOWN = сервис недоступен

HTTP 4xx / 5xx

- 4xx — ошибки запроса
- 5xx — внутренние ошибки сервиса
- позволяют быстро заметить сбой

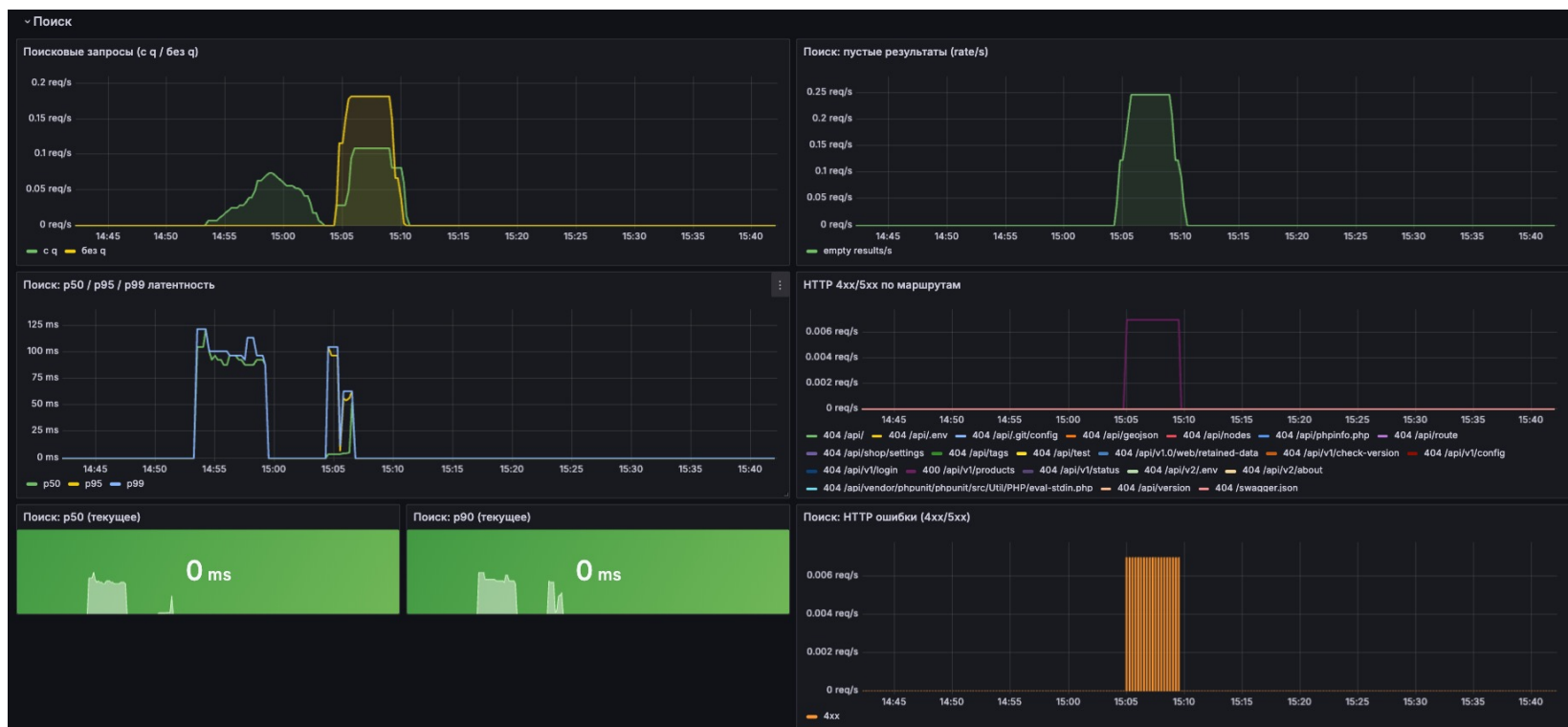
HTTP p95 latency

- показывает задержку для большинства запросов
- помогает отслеживать деградацию API
- важнее среднего времени ответа





Мониторинг. Grafana



Латентность поиска

- p50 — обычный сценарий
- p95 / p99 — тяжёлые запросы
- помогает отслеживать производительность `/api/v1/products`

Ошибки по маршрутам

- локализуют проблемный endpoint
- помогают быстро найти источник сбоя

Текущая задержка поиска

- быстрый индикатор производительности
- позволяет сразу увидеть деградацию

Поисковые запросы: с q / без q

- показывает пользовательские сценарии поиска
- помогает понять, как используется API

Пустые результаты

- показывает долю неуспешных поисков
- помогает оценивать качество каталога и фильтров





Мониторинг. Grafana



Качество входных данных

- топ кодов ошибок
- дубли URL в батче

Активность импорта

- запросы по службам доставки

Результат обработки

- created / updated / failed
- created / updated по сервисам

Нагрузка и производительность

- размер батча: p50 / p75 / p90 / p95
- латентность обработки батча: p50 / p95

Латентность импорта

- показывает скорость обработки одного батча
- помогает отслеживать деградацию backend или БД





Приложения. Миграции

```
databaseChangeLog:
  - changeSet:
      id: 006-unique-product-delivery-url
      author: ma-maiorova
      changes:
        - addUniqueConstraint:
            tableName: product
            columnNames: "delivery_service_id, url"
            constraintName: uk_product_delivery_service_url
```





Приложения. Докер

стек поднимается через `docker compose up -d`;
есть основной `compose`-стек и отдельные
`compose`-файлы для мониторинга и парсера;
чувствительные параметры передаются через `.env`,
а в репозитории лежит `.env.example`;
сервисы разделены по ролям: API, PostgreSQL,
Nginx, monitoring, parser

```
services:
  postgres: {8 keys}

  api:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: food_helper_api
    environment:
      JDBC_URL: jdbc:postgresql://postgres:5432/${POSTGRES_DB}
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      ADMIN_API_KEY: ${ADMIN_API_KEY}
    networks:
      - app_net
    depends_on:
      postgres:
        condition: service_healthy
    healthcheck:
      test: ["CMD-SHELL", "wget -q -O /dev/null http://127.0.0.1:8080/health 2>/dev/null || exit 1"]
      interval: 10s
      timeout: 5s
      retries: 5
      start_period: 90s
    restart: unless-stopped

  nginx: {7 keys}

  parser-scheduler: {8 keys}

networks: app_net

volumes: pgdata, nginx_logs, parser_data
```



Приложения. Конкурентное обновление таблицы

1. Источник обновления данных

- Каталог обновляется **внешним парсером**, который собирает товары с сайтов доставок
- Парсер не пишет напрямую в БД, а отправляет батчи в backend через `POST /api/v1/admin/import`

- Административный контур защищён заголовком `X-API-Key`

2. Как предотвращаются дубли

- Для таблицы `product` задано ограничение уникальности по паре **(delivery_service_id, url)**
- Один и тот же товар в рамках одной службы доставки определяется именно этой парой
- Поэтому повторная отправка уже существующего товара не создаёт новую запись

3. upsert

- Импорт реализован через **ON CONFLICT DO UPDATE**
- Если товар с тем же `(delivery_service_id, url)` уже существует:
 - запись **не вставляется повторно**
 - вместо этого выполняется **обновление существующей строки**
- Если товара ещё нет: создаётся новая запись

4. Поведение при ошибках

- Если ошибка на уровне всего запроса — например, не тот `X-API-Key` — батч отклоняется целиком
- Если в батче невалидны отдельные товары:
 - весь импорт не падает
 - API возвращает статистику `created / updated / failed`
- Это удобно для парсера и для мониторинга качества данных

Parser



POST /api/v1/admin/import



Unique key: (delivery_service_id, url)



Upsert: insert or update





Приложения. Корутины

Корутины

- лёгковесные единицы конкурентного выполнения
- позволяют писать асинхронный код как последовательный
- не требуют создавать отдельный поток ОС на каждую задачу

- backend в основном работает с I/O:
 - HTTP
 - БД
 - внешние сервисы
- при ожидании coroutine может приостановиться и не блокировать поток
- это повышает эффективность обработки запросов

- проект написан на **Kotlin / Ktor**
- Ktor изначально использует coroutine-модель
- suspend-функции хорошо подходят для серверной логики

Почему не virtual threads

- virtual threads — допустимая альтернатива
- но для Kotlin/Ktor coroutines более нативны:
 - встроены в язык
 - лучше интегрированы в стек

