Обо мне

1. Тимлид 7 лет опыта в IT
2. Последние 4 года работал в финтех
3. Сейчас в uchi.ru/vk

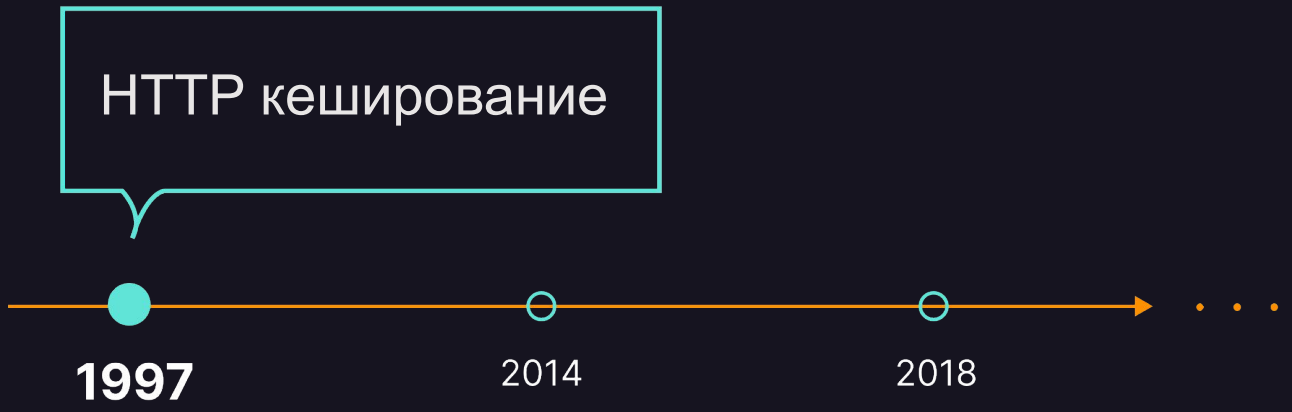
План доклада

01 Как работает HTTP-кеш

02 Service Workers для кеширования

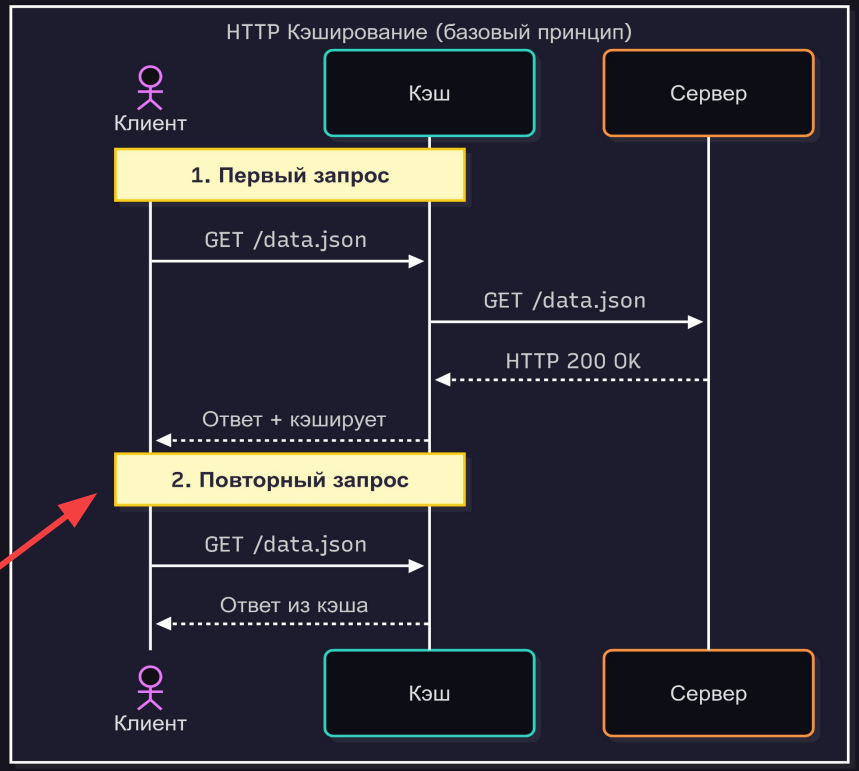
03 Современные клиентские библиотеки

04 Измерение метрик кеширования на клиенте


HTTP кеширование
1997
2014
2018

HTTP-кеш

Механизм временного хранения HTTP-ответов для их повторного использования



Ключевые компоненты HTTP кеша

01 Ключ

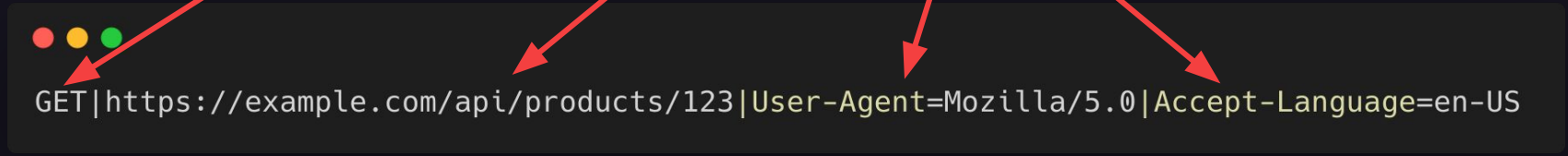
02 Место хранения

03 Состояние

04 Время жизни

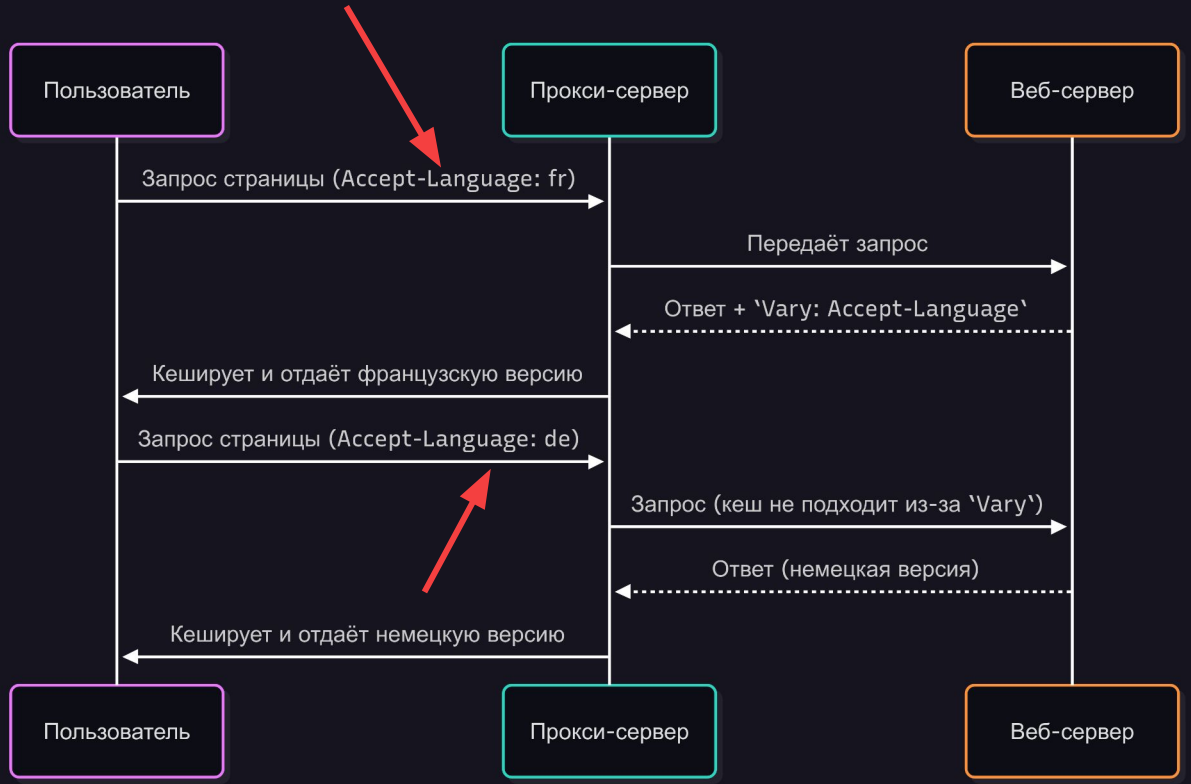
Ключ кеша (Cache Key)

Ключ = [Метод запроса] + [URL] + [Заголовки из Vary]

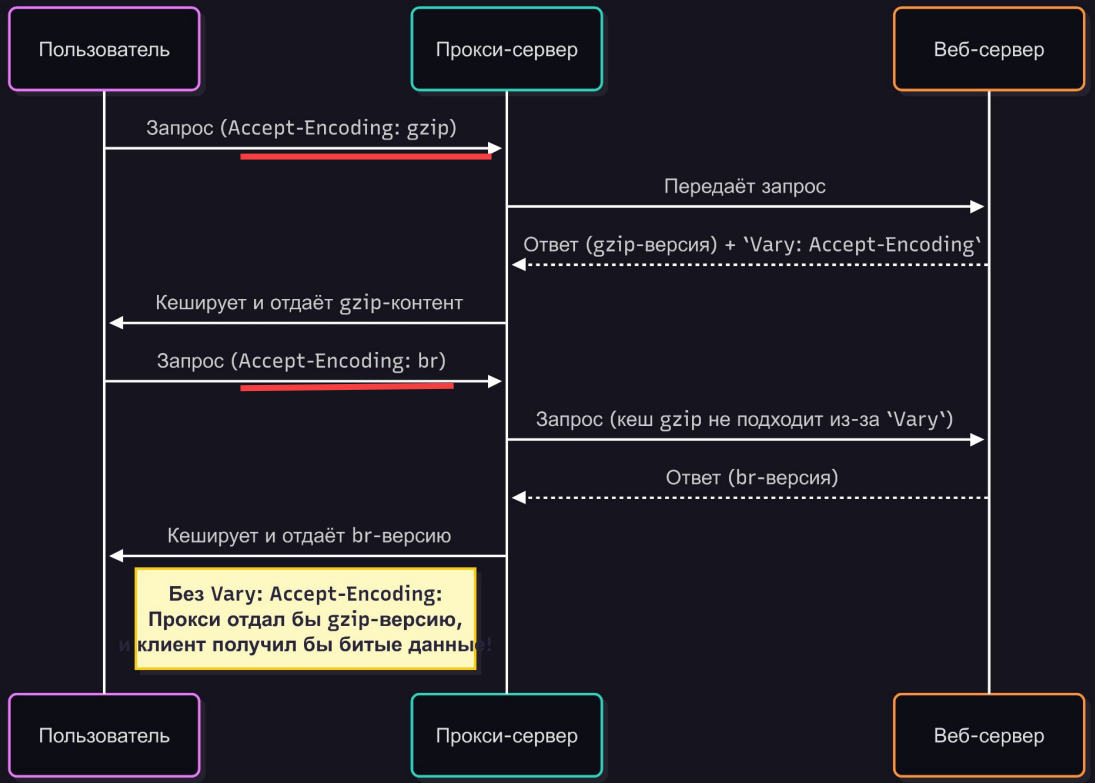


Заголовок **Vary** — ключевой механизм для управления вариативностью ресурсов в HTTP-кешировании

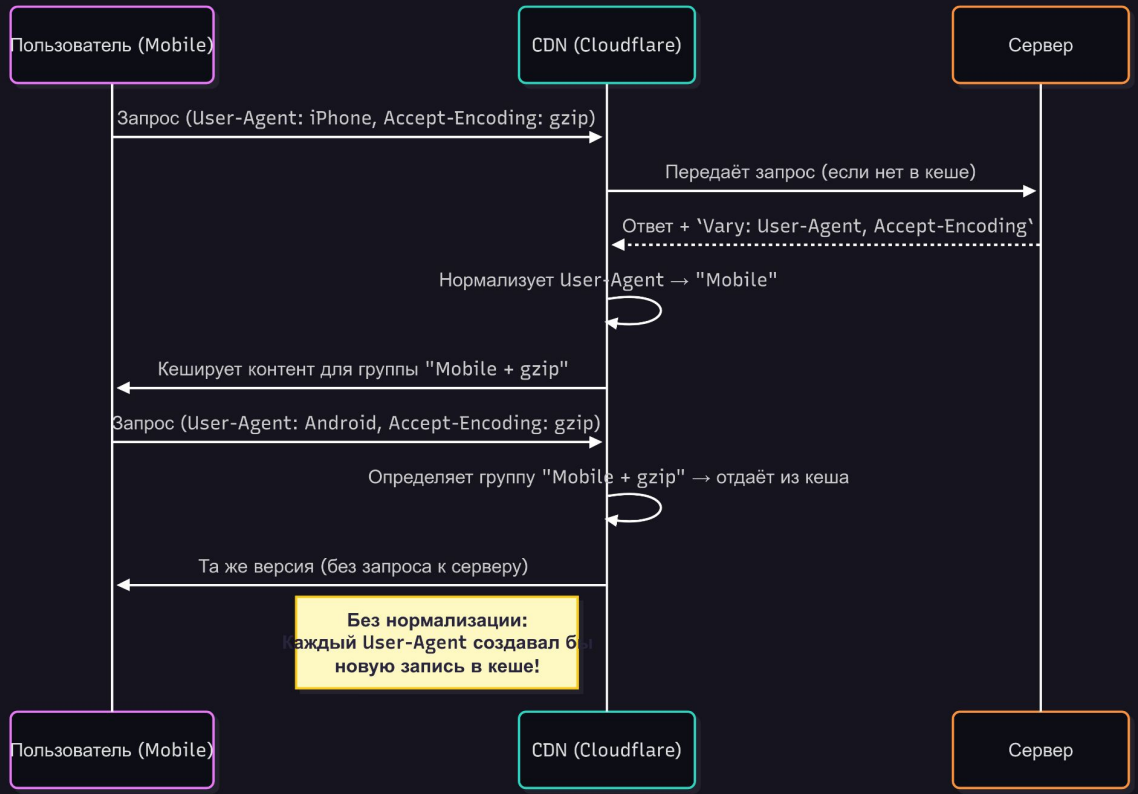
Vary для локализации контента



Vary — Контентная адаптация



Vary — Адаптивный контент



Нормализация User Agent

Исходный User-Agent	Нормализованное значение
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36	desktop-chrome-120
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/17.0 Safari/605.1.15	desktop-safari
Mozilla/5.0 (iPhone; CPU iPhone OS 17_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/17.2 Mobile/15E148 Safari/604.1	mobile-ios-safari

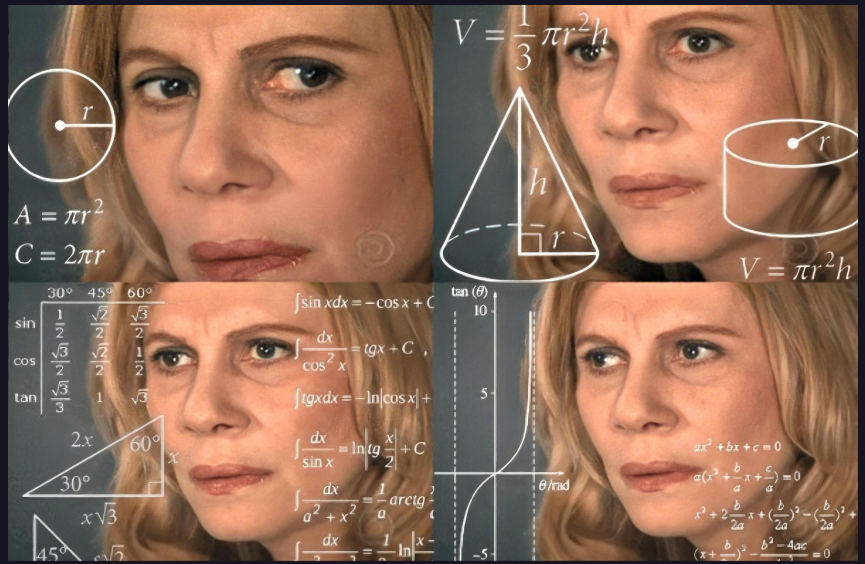
Директивы Cache-Control

5.2.1. Request Directives

- 5.2.1.1. max-age
- 5.2.1.2. max-stale
- 5.2.1.3. min-fresh
- 5.2.1.4. no-cache
- 5.2.1.5. no-store
- 5.2.1.6. no-transform
- 5.2.1.7. only-if-cached

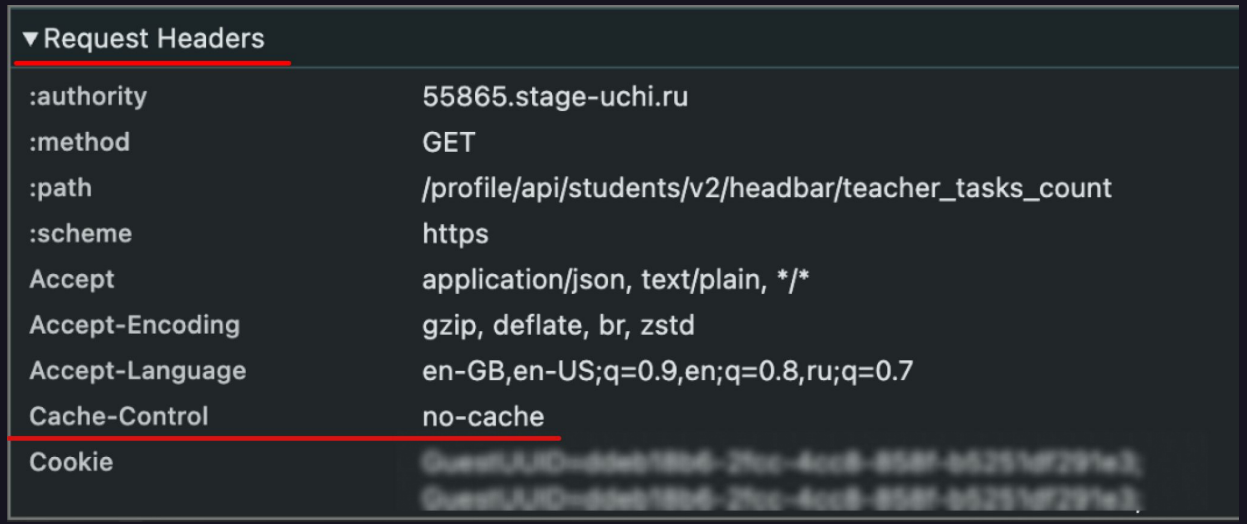
5.2.2. Response Directives

- 5.2.2.1. max-age
- 5.2.2.2. must-revalidate
- 5.2.2.3. must-understand
- 5.2.2.4. no-cache
- 5.2.2.5. no-store
- 5.2.2.6. no-transform
- 5.2.2.7. private
- 5.2.2.8. proxy-revalidate
- 5.2.2.9. public
- 5.2.2.10. s-maxage



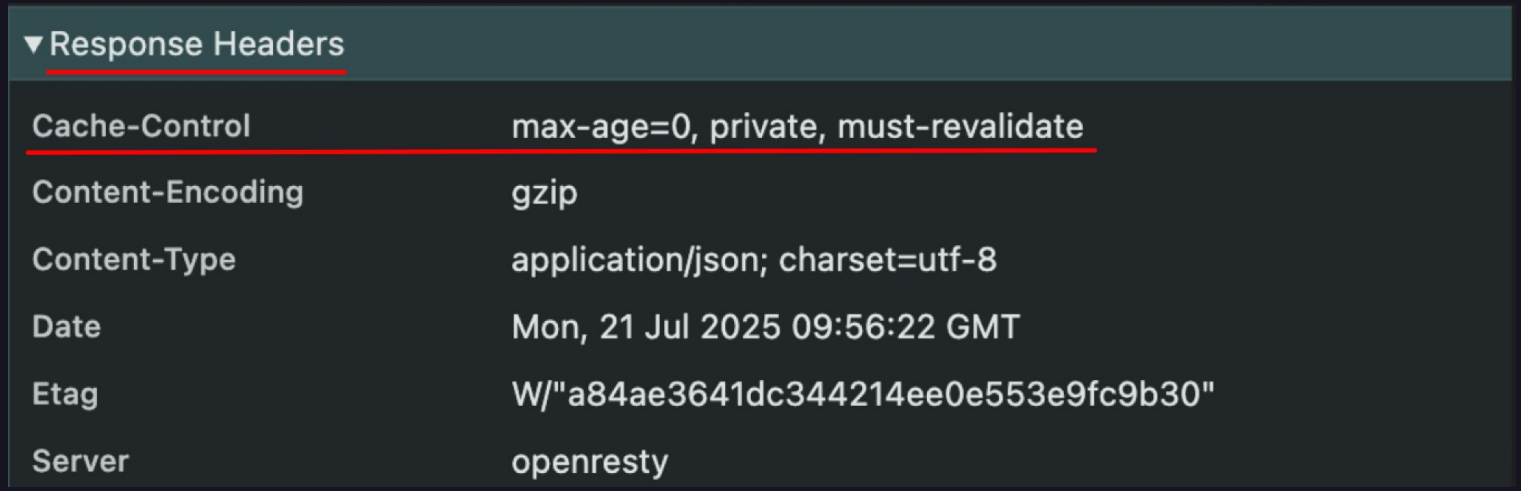
Директивы Cache-Control в запросе

В запросе - указывают **предпочтения** клиента по обработке кешированных данных



Директивы Cache-Control в ответе

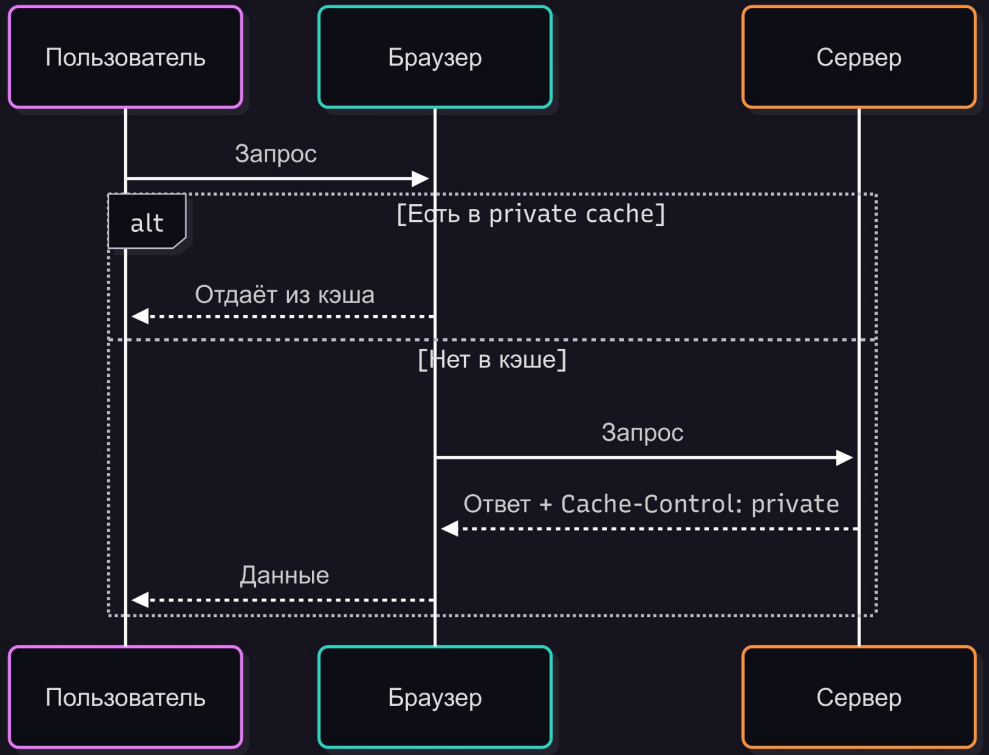
В ответе - определяют **правила** кеширования для ресурса на стороне кешей



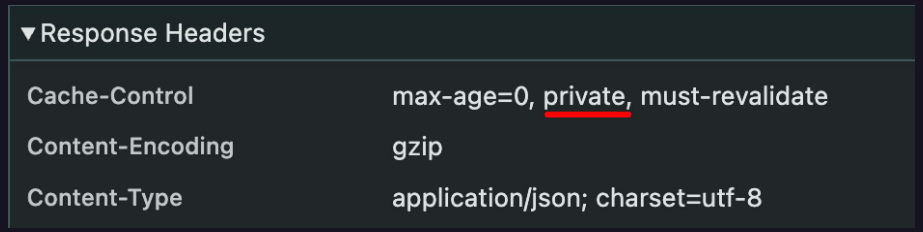
Полный контроль кеширования на клиенте невозможен из-за приоритета серверных правил

Где хранится http cache

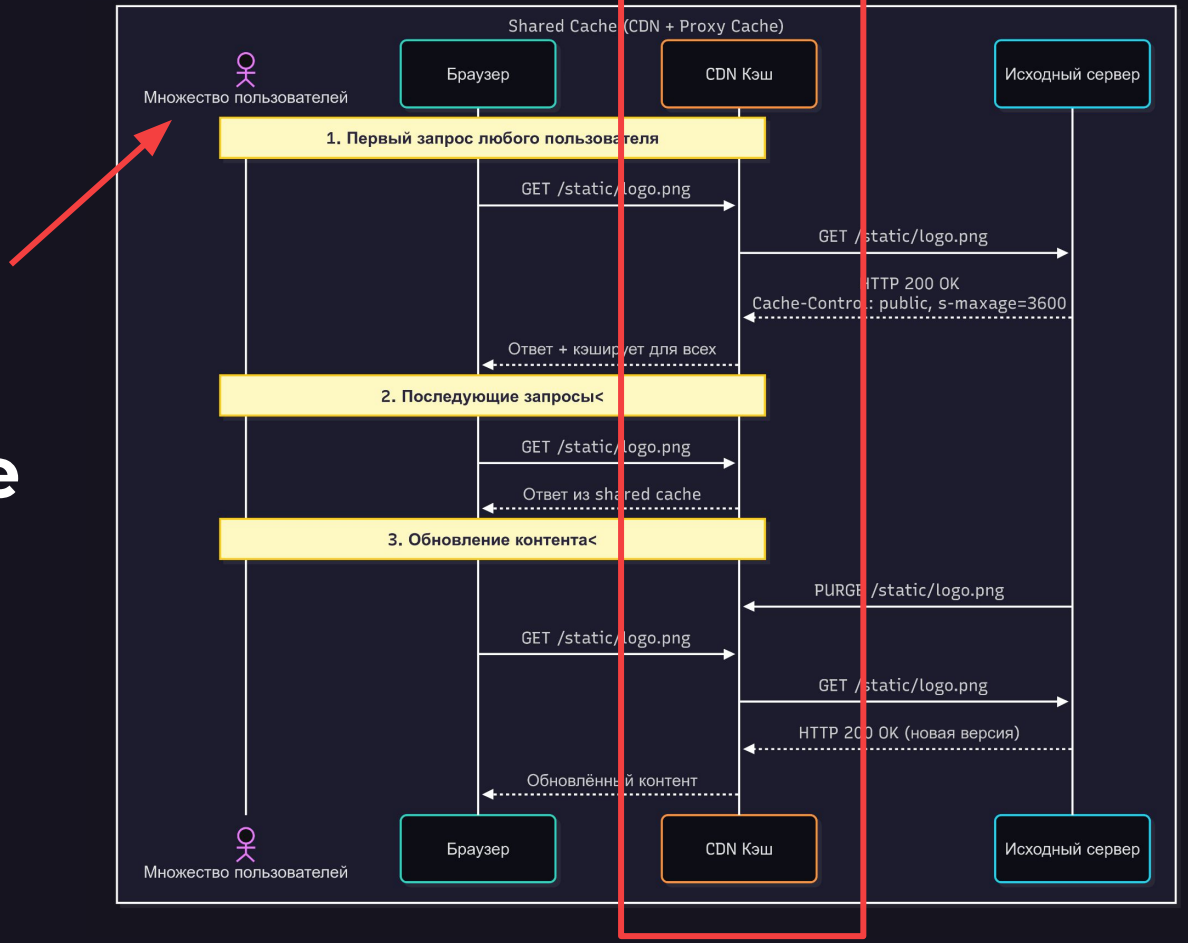
Private cache



Private cache



Shared cache



Shared Cache

01 Кеш у интернет-провайдера (ISP)

02 CDN - Геораспределённые edge-серверы

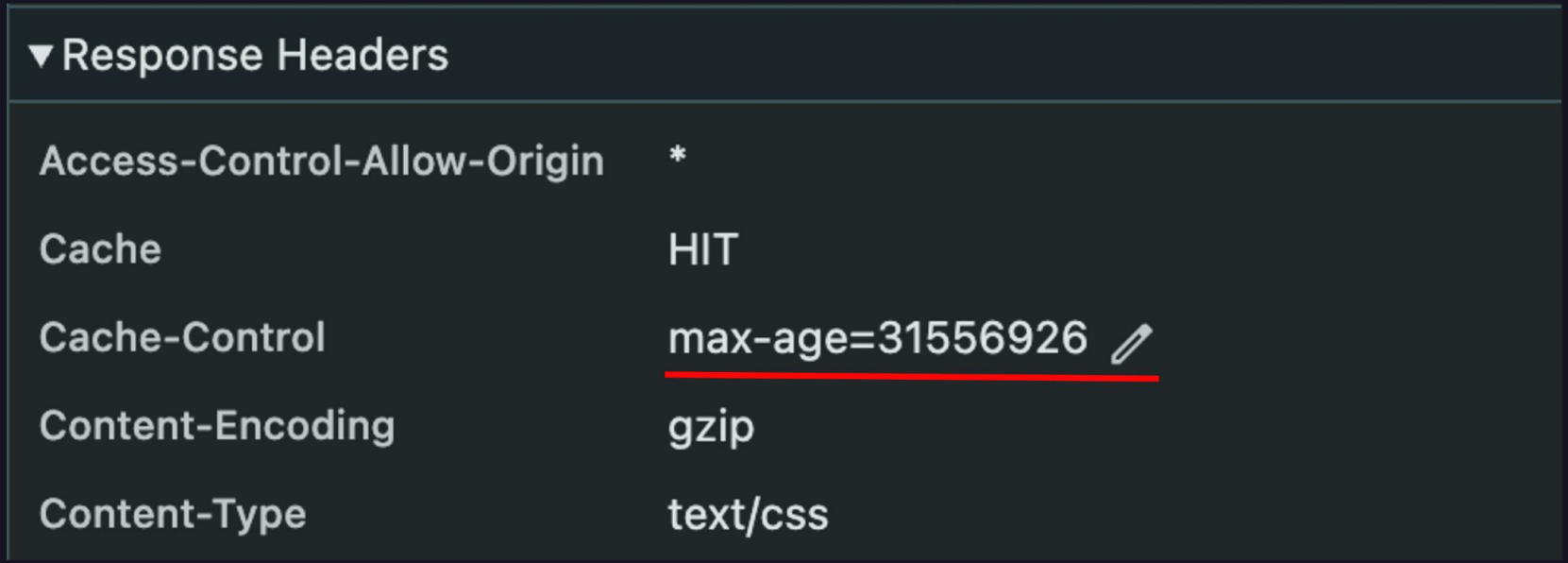
03 Серверы перед origin-сервером (например Nginx)

2 состояния кеша:

Fresh (свежий)

Stale (устаревший)

Директива max-age - показывает сколько времени ресурс считается свежим



Валидаторы

01

Last-Modified

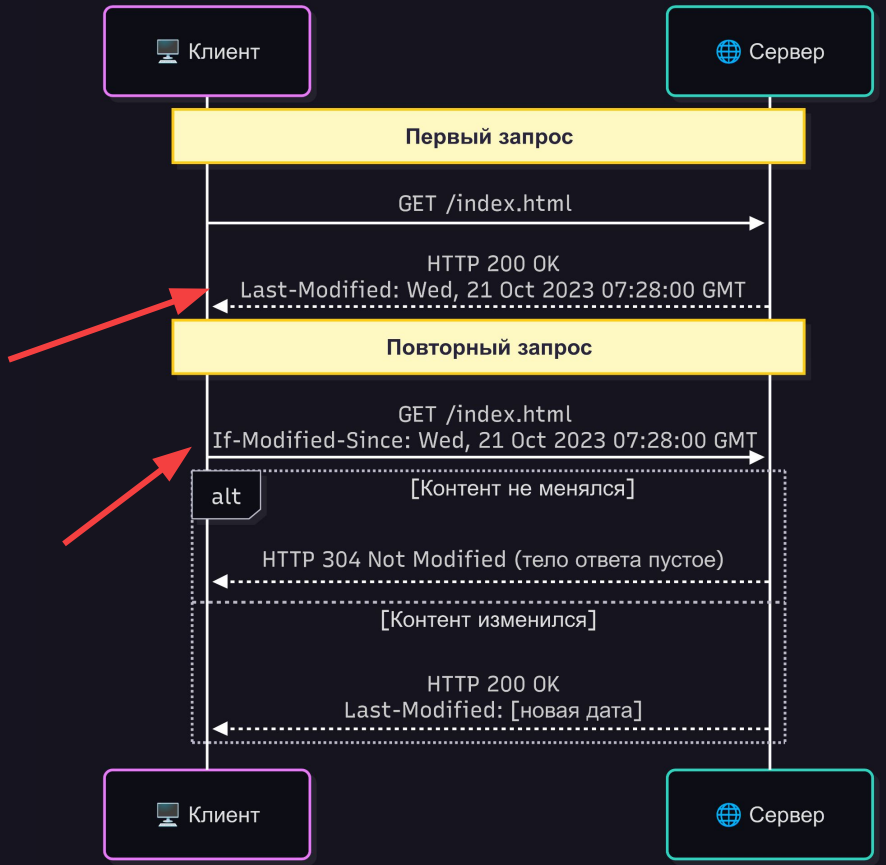
Дата последнего изменения

02

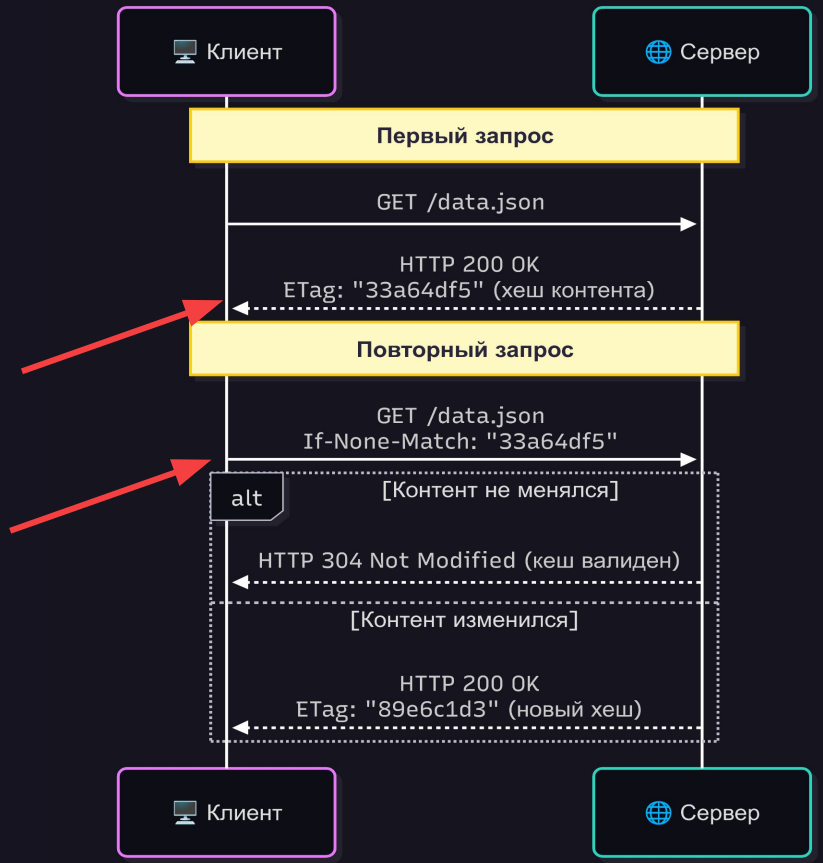
ETag

Идентификатор текущей версии ресурса

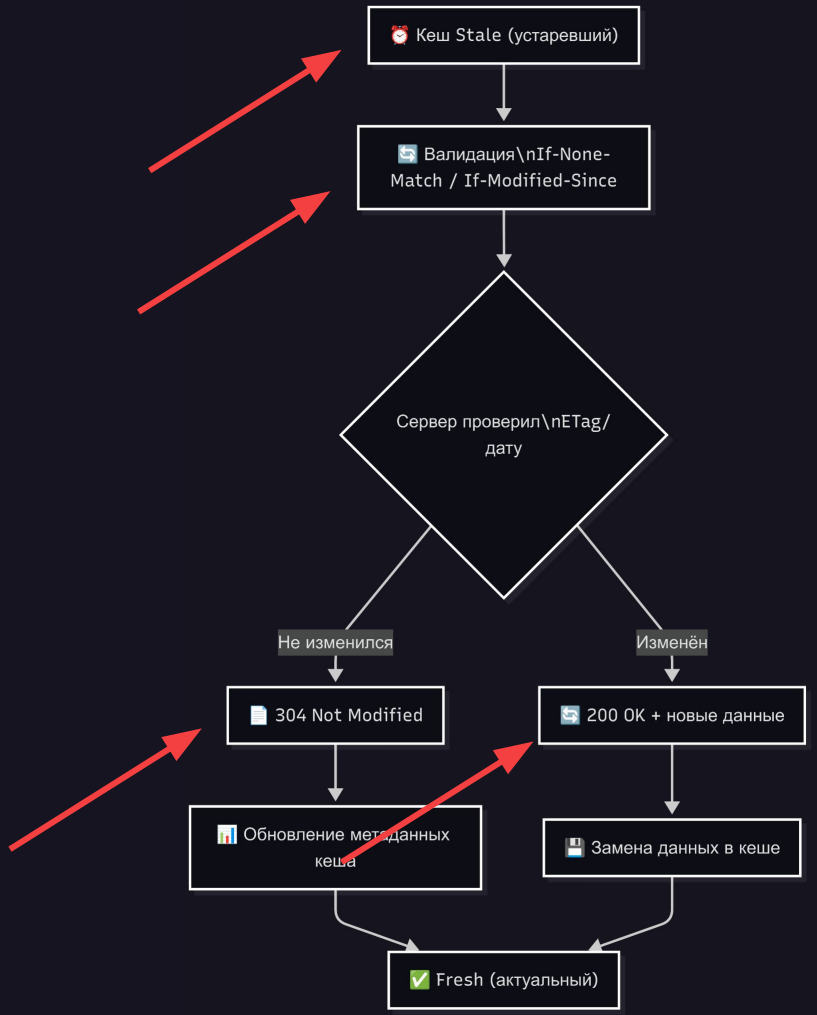
Last-Modified



ETag


Клиент
Сервер
Первый запрос
GET /data.json
HTTP 200 OK
ETag: "33a64df5" (хеш контента)
Повторный запрос
GET /data.json
If-None-Match: "33a64df5"
alt
[Контент не менялся]
HTTP 304 Not Modified (кеш валиден)
[Контент изменился]
HTTP 200 OK
ETag: "89e6c1d3" (новый хеш)
Клиент
Сервер

Валидация


⏰ Кеш Stale (устаревший)
🔄 Валидация\nIf-None-Match / If-Modified-Since
Сервер проверил\nETag/дату
Не изменился
Изменён
📄 304 Not Modified
🔄 200 OK + новые данные
📊 Обновление метаданных кеша
💾 Замена данных в кеше
✅ Fresh (актуальный)

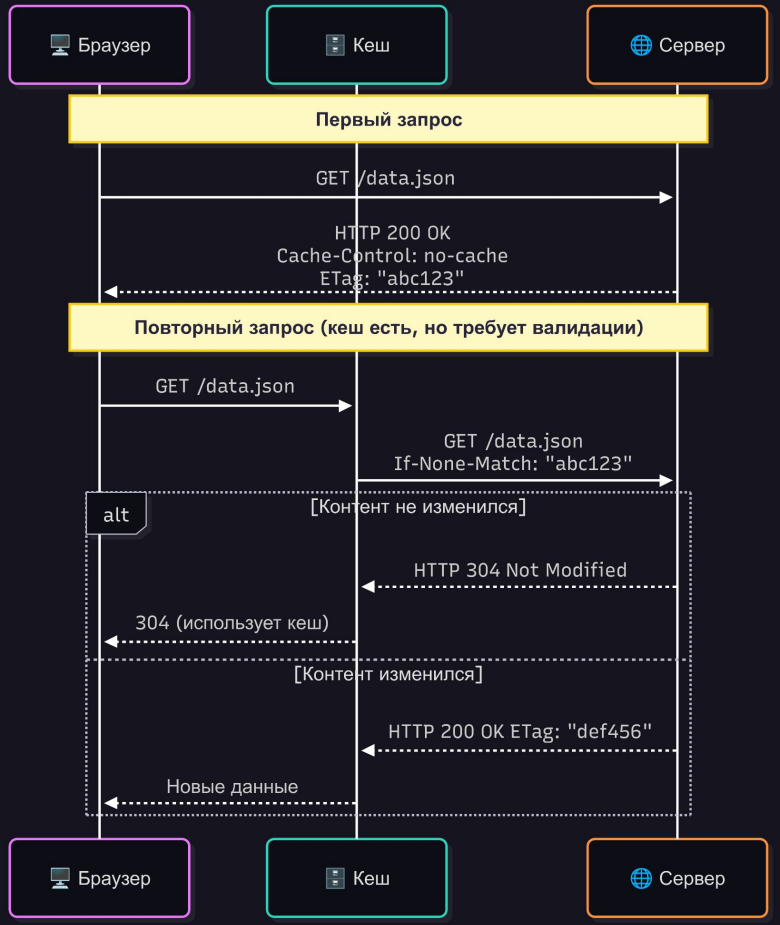
Зачем 2 валидатора ?

01 Гарантированно обнаруживает любые изменения

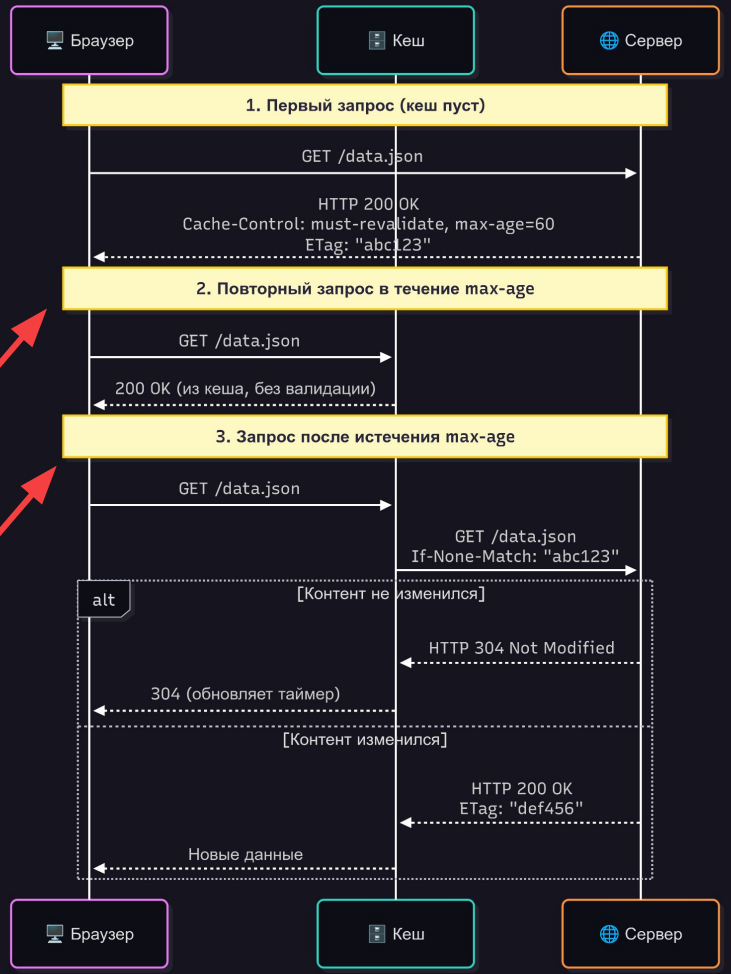
02 Исключения лишних загрузок

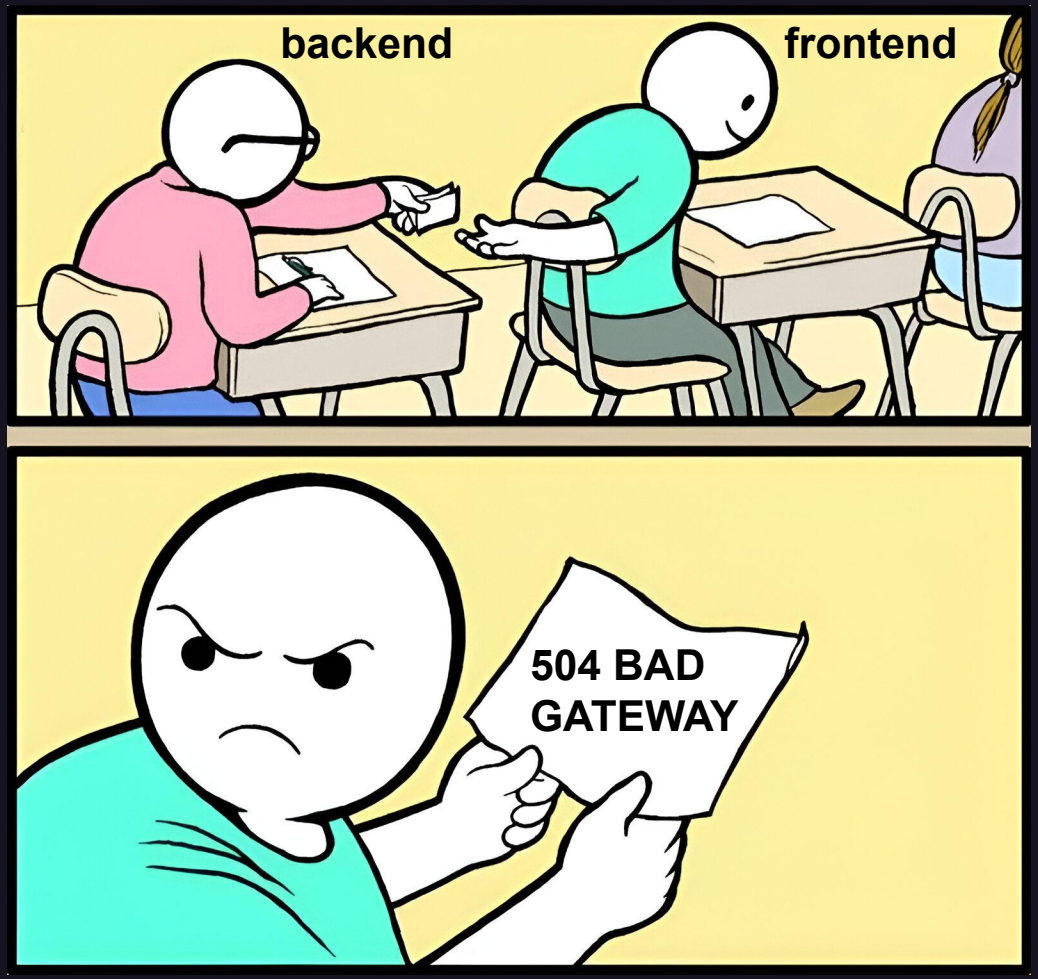
Зачем дополнительная валидация и почему нельзя проверять только max-age?

no-cache

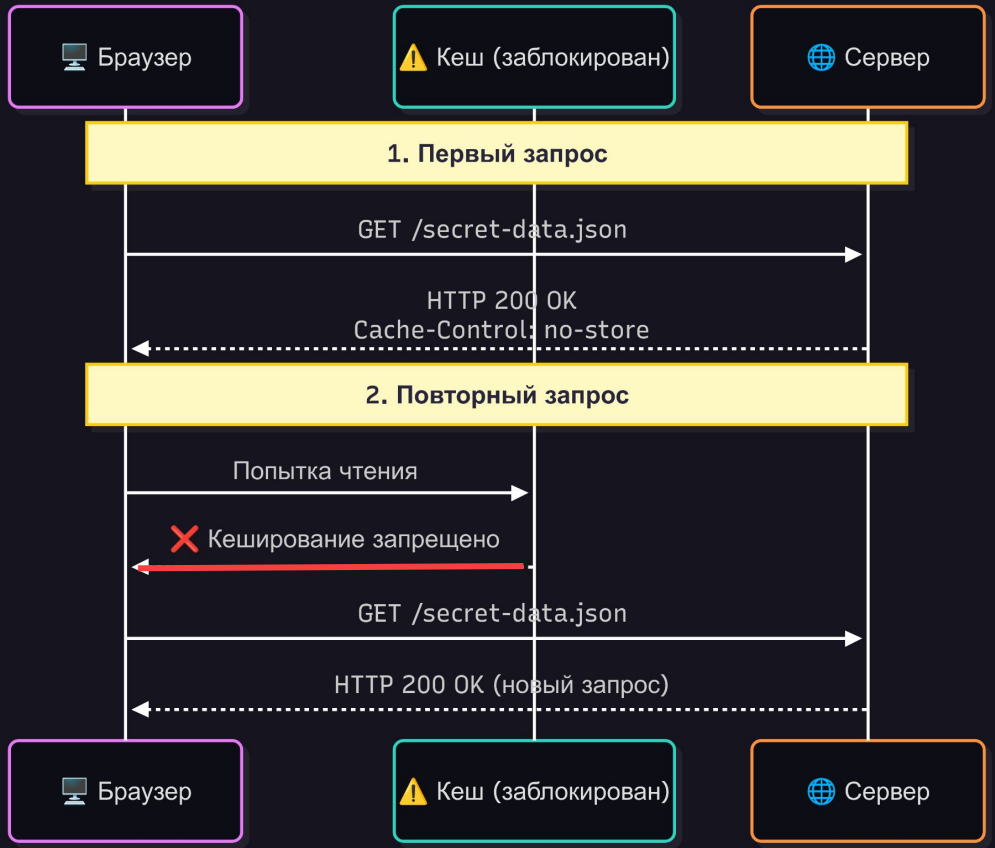

🖥️ Браузер
🗄️ Кеш
🌐 Сервер
Первый запрос
GET /data.json
HTTP 200 OK
Cache-Control: no-cache
ETag: "abc123"
Повторный запрос (кеш есть, но требует валидации)
GET /data.json
GET /data.json
If-None-Match: "abc123"
alt
[Контент не изменился]
HTTP 304 Not Modified
304 (использует кеш)
[Контент изменился]
HTTP 200 OK ETag: "def456"
Новые данные
🖥️ Браузер
🗄️ Кеш
🌐 Сервер

must-revalidate




backend
frontend
504 BAD
GATEWAY

no-store


🖥️ Браузер
⚠️ Кеш (заблокирован)
🌐 Сервер
1. Первый запрос
GET /secret-data.json
HTTP 200 OK
Cache-Control: no-store
2. Повторный запрос
Попытка чтения
❌ Кеширование запрещено
GET /secret-data.json
HTTP 200 OK (новый запрос)
🖥️ Браузер
⚠️ Кеш (заблокирован)
🌐 Сервер

Только для критических данных

Пароли, платежные данные

Персональные данные пользователей

Какое правило приоритетнее ?

сервер

Cache-Control: max-age=3600

VS

клиент

Cache-Control: max-age=0

сервер

Cache-Control: max-age=300, must-revalidate

VS

клиент

Cache-Control: no-cache

сервер

Cache-Control: no-store

VS

клиент

Cache-Control: max-age=300, must-revalidate

Где будет храниться кеш ?

сервер

Cache-Control: public, max-age=600

VS

клиент

Cache-Control: private

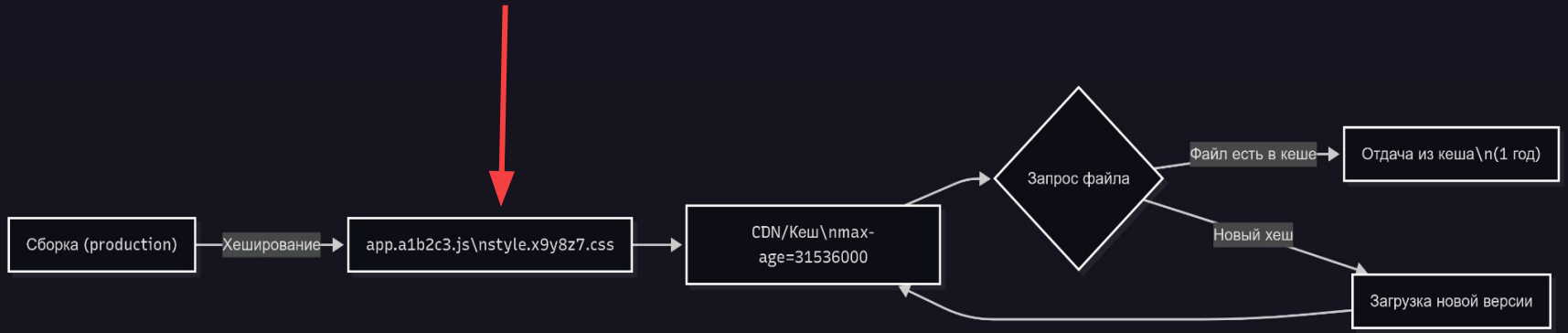
Best practices

01 Используйте длительные TTL для статики

02 Не используйте no-store без необходимости

03 Кешируйте персональный контент с указанием private

Длительные TTL для статики



Не используйте no-store без необходимости

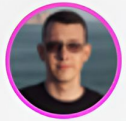
Дополнительная нагрузка на origin-сервер и БД

Увеличение времени ответа и latency

Влияет на BFCache

Смотреть запись

BFCache: как моментально загрузить предыдущую страницу



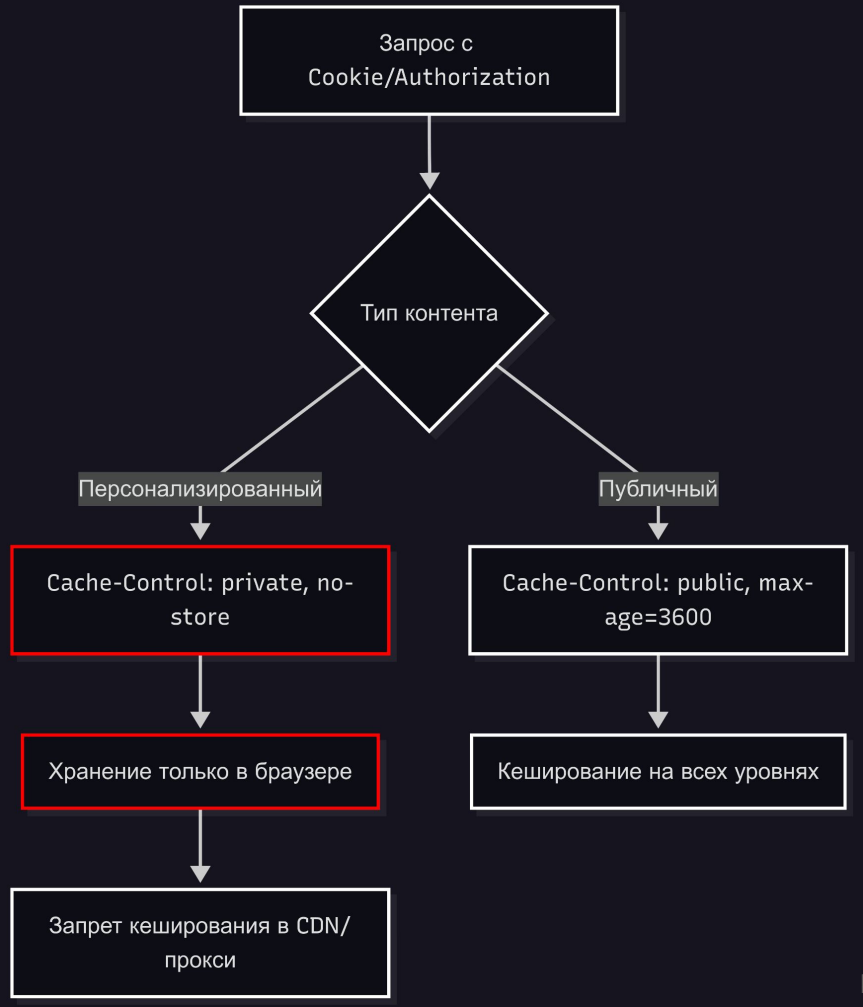
Илья Сидорчик
Яндекс Маркет

RU

Перформанс

Лучше используйте private + no-cache

персональный контент



Что делать, если все-таки сохранили чувствительные данные в shared cache ?

Кадр из фильма Такси 3

Так что же делать ?

01 Не решайте проблему в одиночку

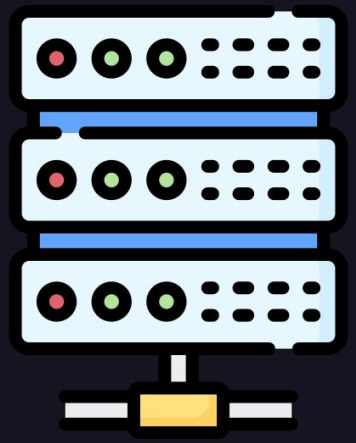
02 Определите точные ключи кеша

03 Точечно удалите кеш во всех shared cache

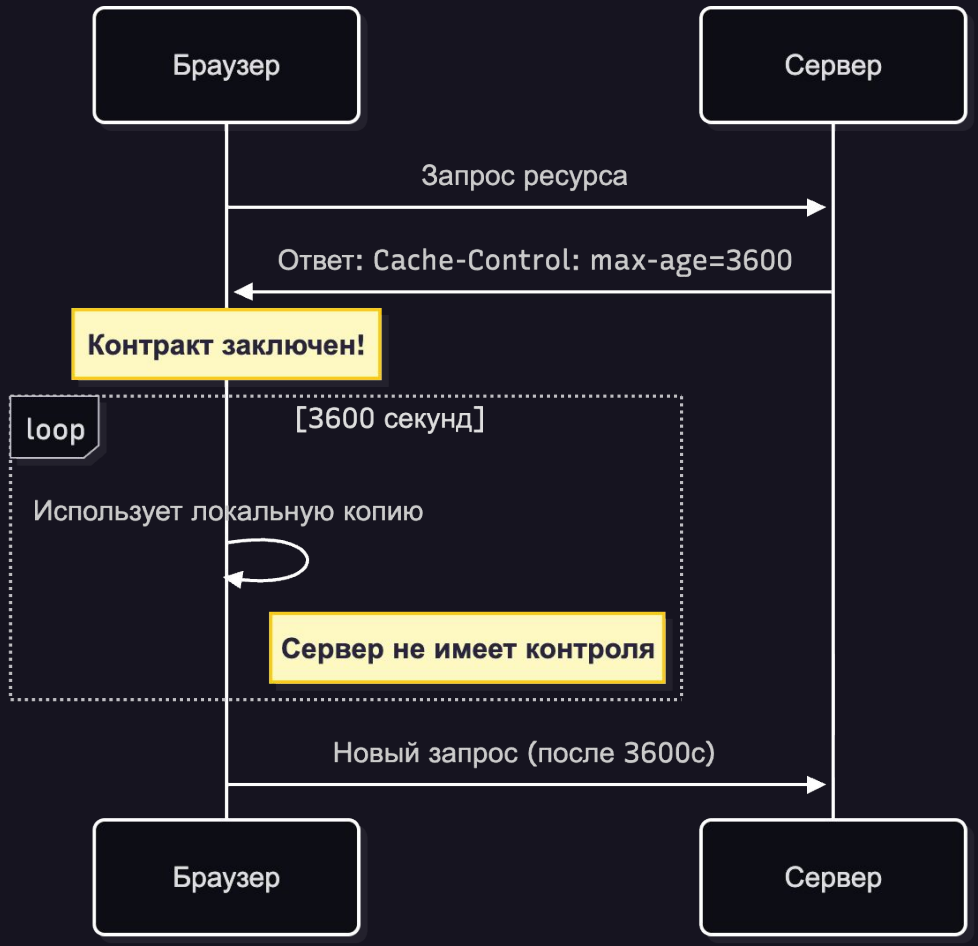
Ограничения http cache

01 Зависимость от серверных заголовков

02 Пока max-age не истек, вы не хозяин своему кешу



Cache-Control: max-age=3600


Браузер
Сервер
Запрос ресурса
Ответ: Cache-Control: max-age=3600
Контракт заключен!
loop
[3600 секунд]
Использует локальную копию
Сервер не имеет контроля
Новый запрос (после 3600с)
Браузер
Сервер

Ограничения http cache

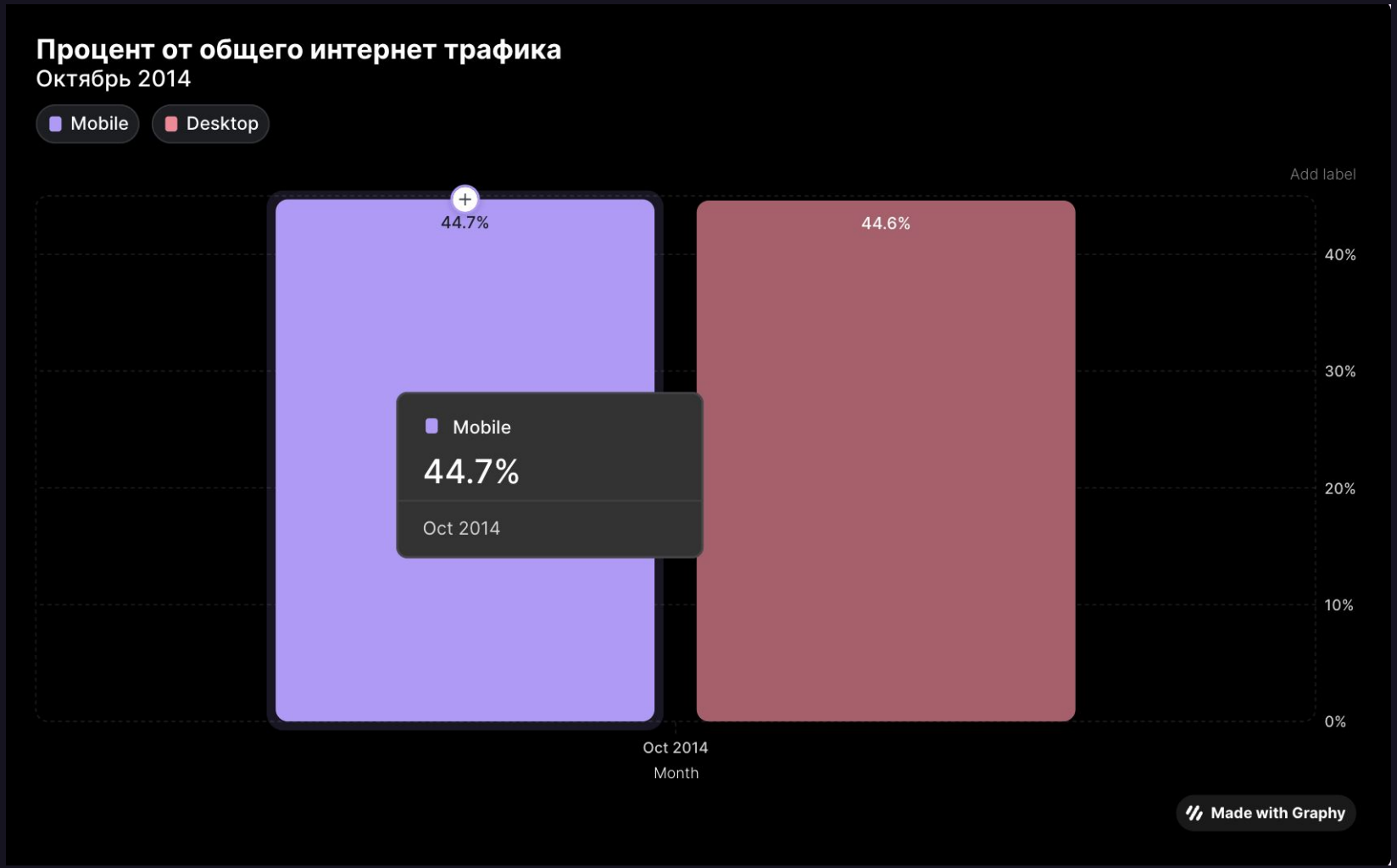
01 Зависимость от серверных заголовков

02 Пока max-age не истек, вы не хозяин своему кешу

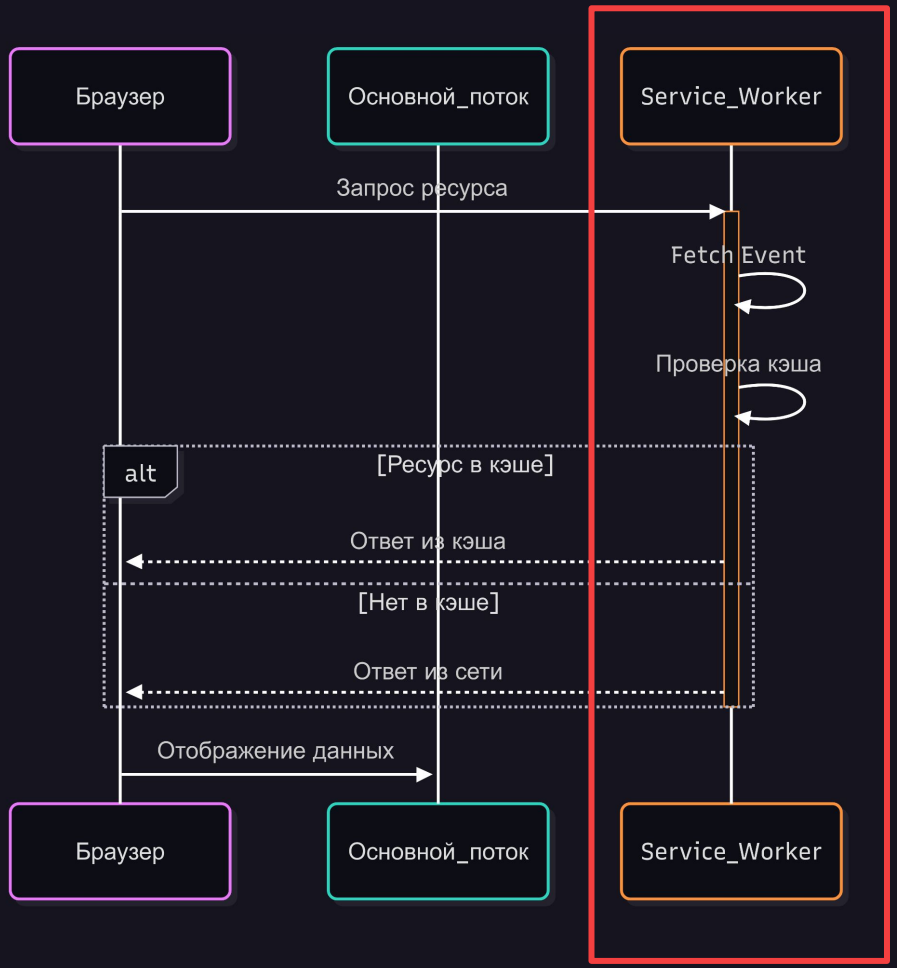
03 Невозможность контекстно-зависимых стратегий

04 Ограниченная обработка ошибок


Service Workers
1997
2014
2018


Процент от общего интернет трафика
Октябрь 2014
Mobile
Desktop
Add label
44.7%
44.6%
40%
30%
20%
10%
0%
Mobile
44.7%
Oct 2014
Oct 2014
Month
Made with Graphy

Service worker - это фоновый скрипт-прокси между приложением и сетью


Браузер
Основной_поток
Service_Worker
Запрос ресурса
Fetch Event
Проверка кэша
alt
[Ресурс в кэше]
Ответ из кэша
[Нет в кэше]
Ответ из сети
Отображение данных
Браузер
Основной_поток
Service_Worker

Как sw побеждает ограничения HTTP cache?

Зависимость от серверных заголовков

```
// Пример: Кэширование вопреки серверу
self.addEventListener('fetch', (event) => {
  if (event.request.url.includes('critical-data')) {
    event.respondWith(
      caches
        .match(event.request)
        .then((cached) => cached || fetchAndCache(event.request)) // Свои правила
    )
  }
})

async function fetchAndCache(request) {
  const response = await fetch(request)
  const cache = await caches.open('v1')
  cache.put(request, response.clone()) // Сохраняем, игнорируя `no-store`
  return response
}

```

Может анализировать контекст выполнения

```
// Разные стратегии для 4G/3G/оффлайн
event.respondWith(
  navigator.connection.effectiveType === '4g'
    ? networkFirst(event.request)
    : cacheFirst(event.request)
);

// Пример для медленных сетей
function cacheFirst(request) {
  return caches.match(request)
    .then(cached => cached || fetch(request));
}
```

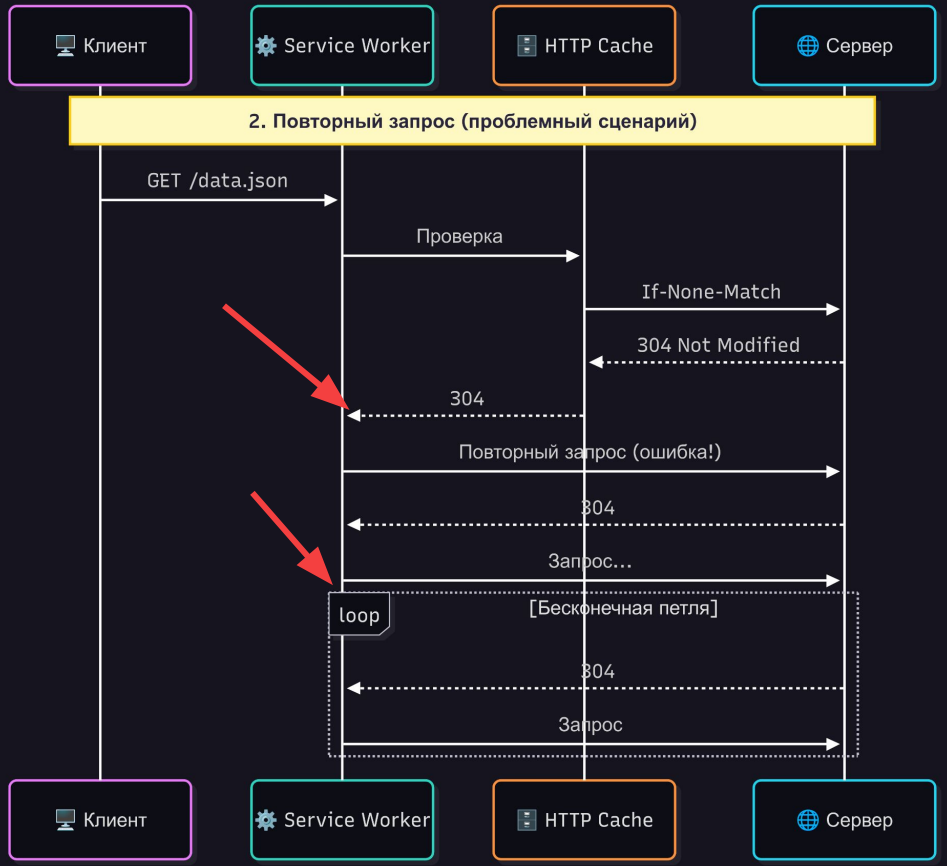
Обработка ошибок сети

```
    event.respondWith(
  fetchWithTimeout(event.request, 2000) // Пытаемся получить сеть
    .catch(() ⇒ caches.match(request)) // → Кэш
    .catch(() ⇒ caches.match('/offline.html')) // → Статичный fallback
    .catch(() ⇒ new Response('Оффлайн-режим')) // → Динамический fallback
);

function fetchWithTimeout(request, timeout) {
  return Promise.race([
    fetch(request),
    new Promise((_, reject) ⇒
      setTimeout(() ⇒ reject(new Error('Timeout')), timeout))
  ]);
}
```

Service Workers нужны если вы столкнулись с ограничениями HTTP cache

Частые ошибки при совместном использование SW и HTTP cache

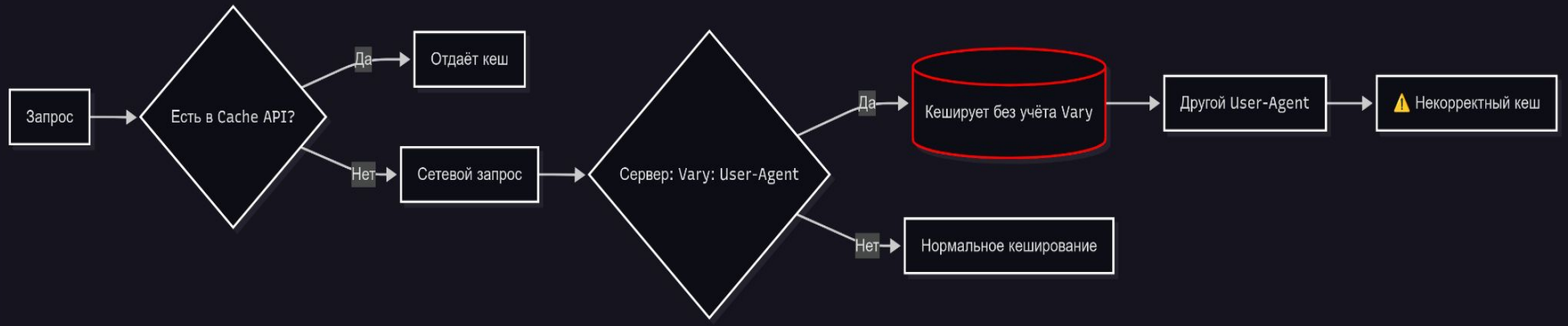
Бесконечная ревалидация (Validation Loop)



Решение: обрабатывать 304

```
self.addEventListener('fetch', event ⇒ {
  event.respondWith(
    (async () ⇒ {
      // 1. Проверяем кэш
      const cachedResponse = await caches.match(event.request);

      // 2. Делаем сетевой запрос
      try {
        const networkResponse = await fetch(event.request);

        // 3. Корректно обрабатываем 304
        if (networkResponse.status ⩵ 304 && cachedResponse) {
          return cachedResponse; // Используем кэшированную версию
        }

        // 4. Обновляем кэш при успешном ответе
        if (networkResponse.status ⩵ 200) {
          const cache = await caches.open('dynamic-cache');
          await cache.put(event.request, networkResponse.clone());
        }

        return networkResponse;
      } catch (error) {
        // 5. Fallback на кэш при ошибке сети
        return cachedResponse || new Response('Офлайн-контент');
      }
    })()
  );
});
```

Service Workers не учитывают Vary



Решение: учитывать Vary при формировании ключей

```
1 const varyHeaders = ['User-Agent', 'Accept-Language']
2 const cacheKey = `${e.request.url}-${varyHeaders
3   .map((h) ⟹ e.request.headers.get(h))
4   .join('-')}`
5
```

Выводы: Разделяйте ответственность

01 **Используйте** HTTP Cache — для статики с хешами

02 **Используйте** Service Workers — для динамики

Best practices

01 Используйте Workbox

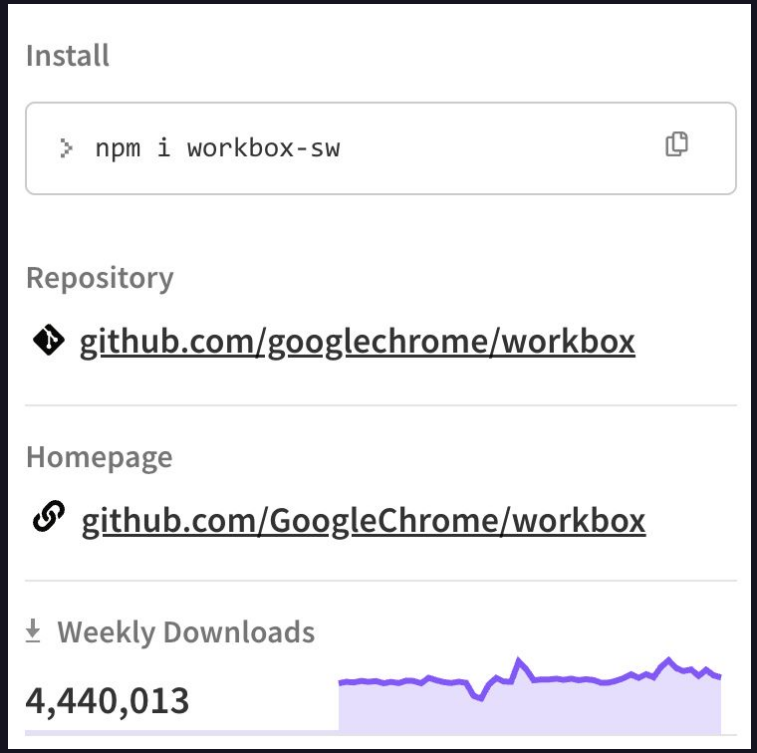
02 Выбирайте стратегии под тип контента

03 Разделяйте кеши под каждый тип контента

Best practices

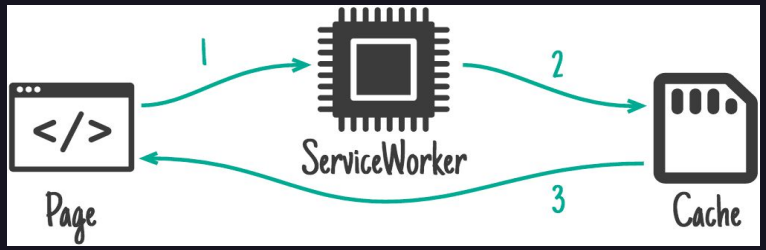
Workbox

Готовые решения для
сложных сценариев

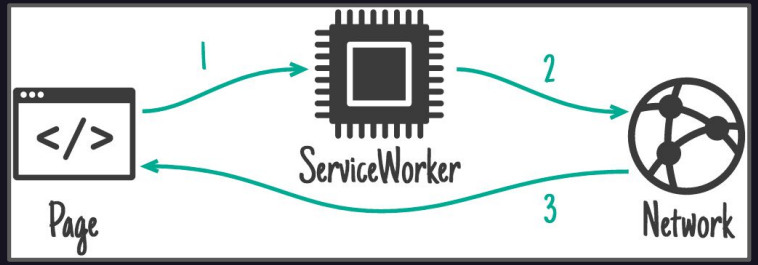


Стратегии в зависимости от контента

CacheFirst



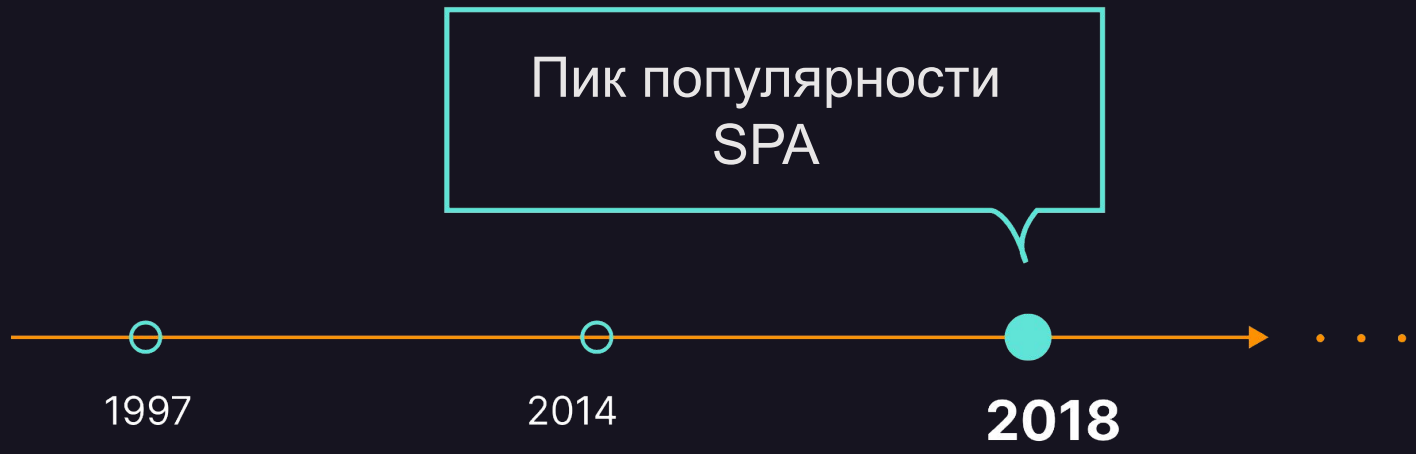
NetworkOnly



Подробнее про стратегии

Используйте раздельные кеши

```
// sw.js
import { registerRoute, Route } from 'workbox-routing';
import { CacheFirst, StaleWhileRevalidate } from 'workbox-strategies';

// Handle images:
const imageRoute = new Route(({ request }) => {
  return request.destination === 'image'
}, new StaleWhileRevalidate({
  cacheName: 'images'
}));

// Handle scripts:
const scriptsRoute = new Route(({ request }) => {
  return request.destination === 'script';
}, new CacheFirst({
  cacheName: 'scripts'
}));

// Handle styles:
const stylesRoute = new Route(({ request }) => {
  return request.destination === 'style';
}, new CacheFirst({
  cacheName: 'styles'
}));

// Register routes
registerRoute(imageRoute);
registerRoute(scriptsRoute);
registerRoute(stylesRoute);('beforeunload', this.sendToSentry);
  }
```


Пик популярности SPA
1997
2014
2018

Современное решение по управлению кешированием на клиенте

Транспортное кеширование

Кеширование как состояние

Axios Cache Interceptor

Разделение кеша и состояния упрощает архитектуру

Как это работает

01 Interceptor - Перехватывает запросы перед отправкой.

02 Создаёт ID на основе параметров запроса

03 На основе ID возвращает кеш или делает запрос

Пример

```
import axios from 'axios'
import { setupCache } from 'axios-cache-interceptor'

// Настройка кеширования
const api = setupCache(axios, {
  ttl: 60_000,  // 1 минута
  storage: localStorage,  // Сохранять между сессиями
})

// Запрос с кешированием
const { data } = await api.get('/api/posts', {
  cache: {
    ttl: 300_000, // Переопределить TTL для этого запроса
    interpretHeader: false, // Игнорировать Cache-Control сервера
  },
})

console.log(data) // Данные из кеша или сервера

```

Настройка кастомного TTL

```
1 import { setupCache, buildStorage } from 'axios-cache-interceptor';
2
3 const api = setupCache(axios, {
4   ttl: 1000 * 60 * 5, // 5 минут
5 });
```

Можем интегрировать популярные хранилища данных

```
1 import { setupCache, buildStorage } from 'axios-cache-interceptor';
2
3 const api = setupCache(axios, {
4   storage: buildStorage({
5     // Можно использовать localStorage для сохранения кэша между сессиями
6     set: (key, value) => localStorage.setItem(key, JSON.stringify(value)),
7     get: (key) => JSON.parse(localStorage.getItem(key)),
8     remove: (key) => localStorage.removeItem(key)
9   })
10 });
```

Можем использовать Etag и Last-Modified

```
const api = setupCache(axios, {
  etag: true, // Включаем поддержку ETag
  modifiedSince: true // Включаем поддержку Last-Modified
});
```

Кастомизация поведения при ошибках

```
1 const api = setupCache(axios, {
2   // Возвращаем закэшированные данные при ошибке сети
3   staleIfError: (error) ⇒ error.isAxiosError && !error.response
4 });
```

Лёгкость отладки

```
const api = setupCache(axios, { debug: console.log });
await api.get('https://api.example.com/data'); // Первый запрос
await api.get('https://api.example.com/data'); // Повторный запрос


[Axios Cache] Checking cache for: https://api.example.com/data
[Axios Cache] No cache found for: https://api.example.com/data
[Axios Cache] Storing new response in cache for: https://api.example.com/data
[Axios Cache] Stored successfully. TTL: 300000ms

[Axios Cache] Checking cache for: https://api.example.com/data
[Axios Cache] Found valid cache for: https://api.example.com/data
[Axios Cache] Serving response from cache for: https://api.example.com/data
```

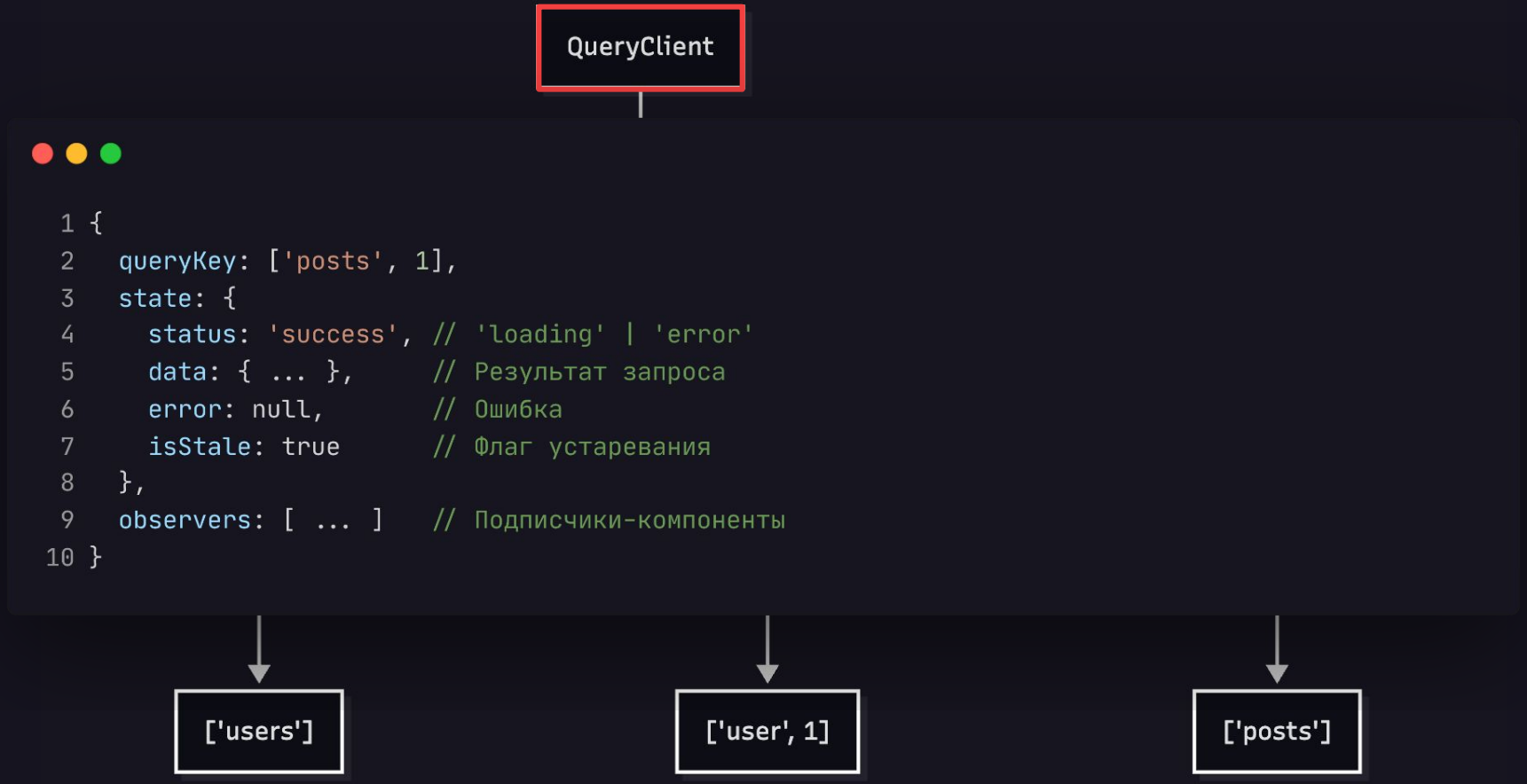
TanStack Query: кеширование как состояние приложения

01

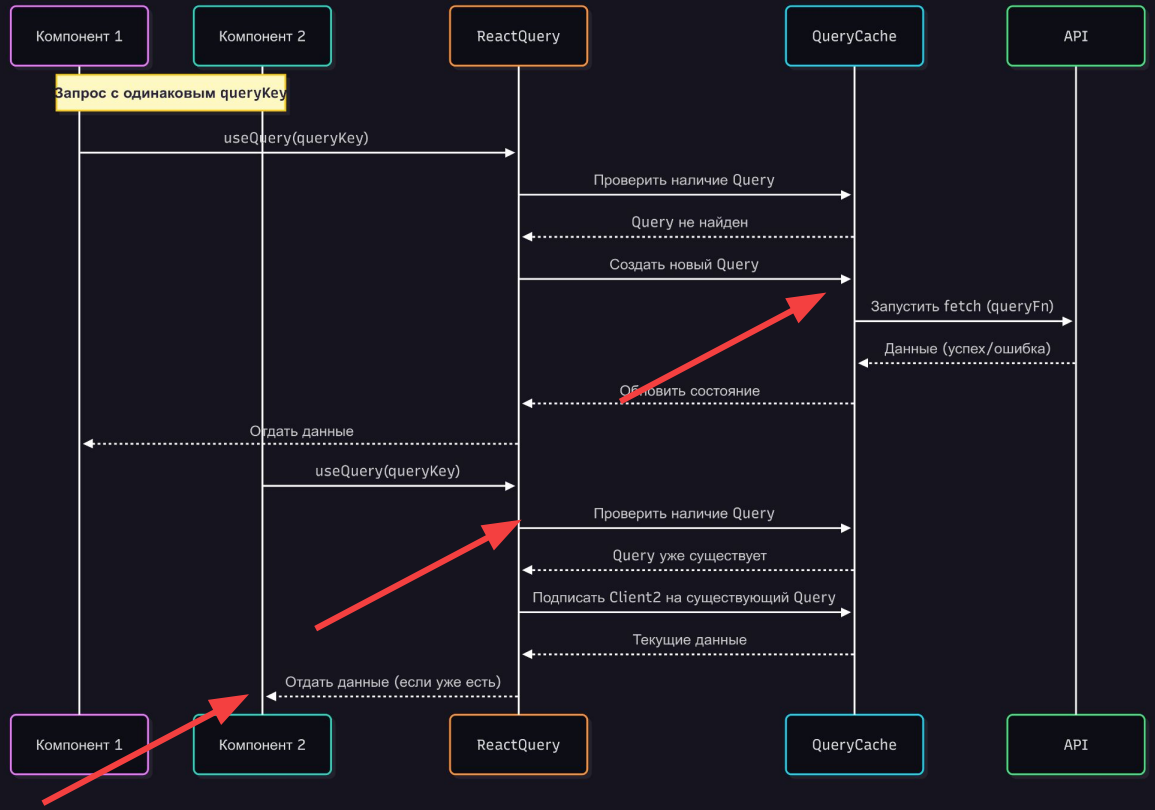
Данные API — полноценная часть состояния

02

Библиотека сама управляет их актуальностью, обновлением и синхронизацией

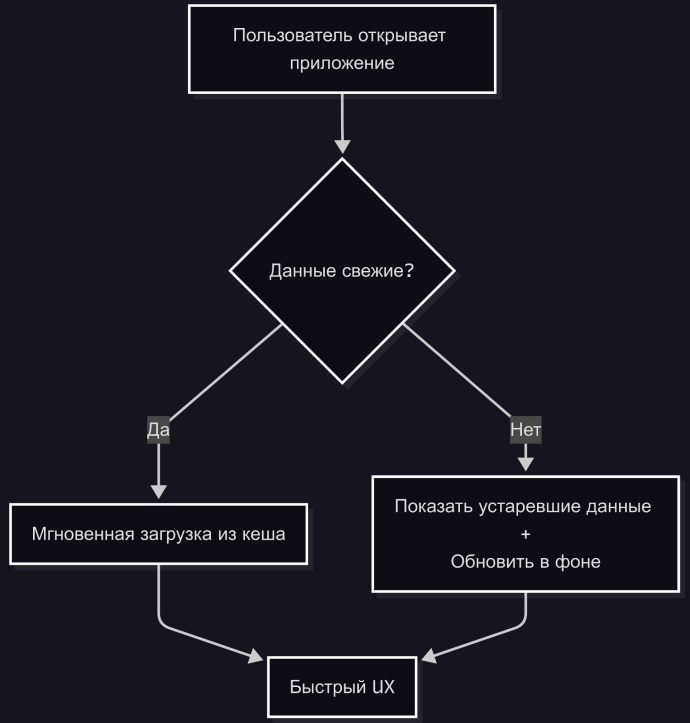

QueryClient
{
queryKey: ['posts', 1],
state: {
status: 'success', // 'loading' | 'error'
data: { ... }, // Результат запроса
error: null, // Ошибка
isStale: true // Флаг устаревания
},
observers: [...] // Подписчики-компоненты
}
['users']
['user', 1]
['posts']

Дедупликация запросов (Query Deduplication)



Stale-While-Revalidate (Фоновое обновление)

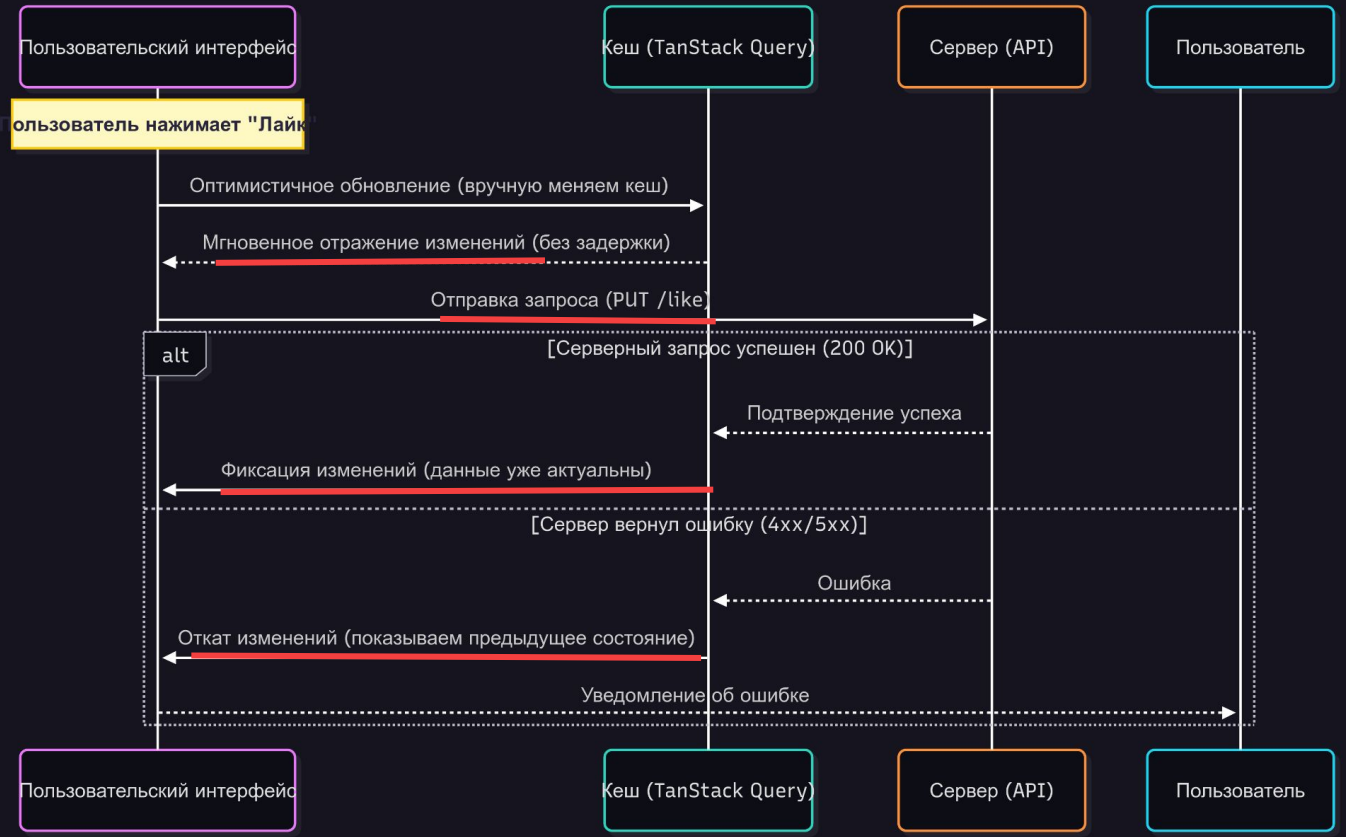
```
staleTime: 5 * 60 * 1000 // 5 минут
```



Stale-While-Revalidate

```tsx
import { useQuery } from '@tanstack/react-query'

function TodoList() {
  const { data } = useQuery({
    queryKey: ['todos'],
    queryFn: () => fetch('/api/todos').then((res) => res.json()),
  })

  // 1. Сначала покажет кэш (если он есть), затем обновит в фоне
  // 2. При смене фокуса на вкладке — автоматически запросит свежие данные
  return (
    <div>
      {data?.map((todo) => (
        <div key={todo.id}>{todo.title}</div>
      ))}
    </div>
  )
}

```

Оптимистические обновления



Оптимистические обновления

```
useMutation({
  mutationFn: updateTodo,
  onMutate: async (newTodo) => {
    // Отменяем текущие запросы, чтобы они не перезаписали оптимистичное
  обновление
    await queryClient.cancelQueries({ queryKey: ['todos'] })

    // Сохраняем предыдущее состояние для отката
    const previousTodos = queryClient.getQueryData(['todos'])

    // Оптимистично обновляем кэш
    queryClient.setQueryData(['todos'], (old) => [...old, newTodo])

    // Возвращаем контекст с предыдущими данными для отката
    return { previousTodos }
  },
  onError: (err, newTodo, context) => {
    // В случае ошибки откатываем изменения
    queryClient.setQueryData(['todos'], context.previousTodos)
  },
  onSettled: () => {
    // В любом случае инвалидируем кэш для синхронизации с сервером
    queryClient.invalidateQueries({ queryKey: ['todos'] })
  },
})

```

Префетчинг данных (Prefetching)

```tsx
import { useQueryClient } from '@tanstack/react-query'

function ArticleLink({ id }) {
  const queryClient = useQueryClient()

  const prefetchArticle = async () => {
    // Предзагружаем данные
    await queryClient.prefetchQuery({
      queryKey: ['article', id],
      queryFn: () => fetchArticle(id),
      staleTime: 60 * 1000, // 1 минута
    })
  }

  return (
    <Link
      to={`/articles/${id}`}
      onMouseEnter={prefetchArticle} // Префетчим при наведении
    >
      Читать статью
    </Link>
  )
}

```

Пагинация и бесконечные запросы

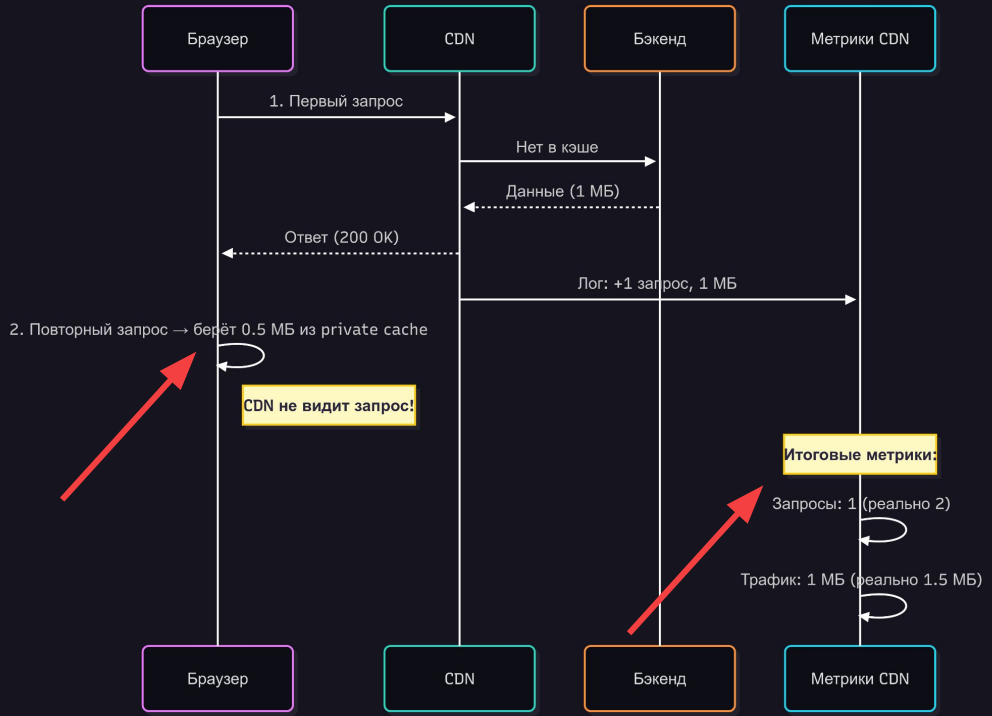
```
1 const { data, isFetching } = useQuery({
2     queryKey: ['articles', page],
3     queryFn: () ⇒ fetchArticles(page),
4     keepPreviousData: true, // Сохраняем предыдущие данные
5   });
```

А нужны ли вам Service Workers ?

Кеширование API запросов ≠ SW

Метрики кеширования на клиенте

Почему нельзя полагаться только на серверные метрики кеширования ?



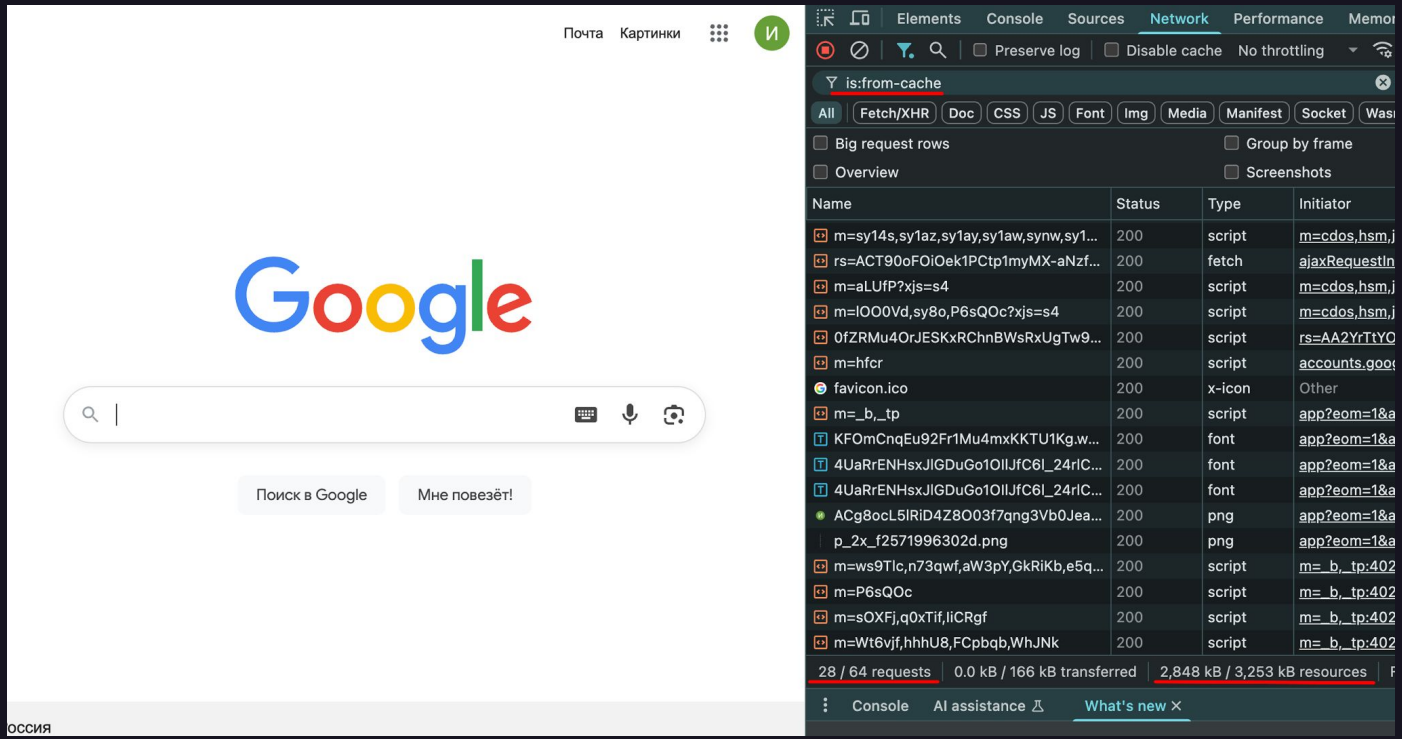
Cache Hit Rate (Число попаданий в кеш)

Cache Hit Rate (%) = (Кол-во запросов из кеша / Общее кол-во запросов) × 100%

Bandwidth Savings (Экономия трафика)

Bandwidth Savings (%) = (Размер данных из кеша / Общий запрошенный размер данных) × 100%

Анализ метрик через devtools



На примере google.com

CHR = 28 / 64 = **44%**

Bandwidth Savings - 2848 / 3253 = **88%**

Автоматический анализ через Performance API

```
1 performance.getEntriesByType('resource')
```

PerformanceObserver - API для отслеживания метрик производительности динамически

```
this.observer = new PerformanceObserver(list => {
  list.getEntries().forEach(entry => this.classifyAndPush(entry));
});
this.observer.observe({ type: 'resource' });
```

Каждый ресурс

```
{
  name: "https://example.com/style.css", // URL ресурса
  entryType: "resource",
  startTime: 150.5, // Время начала загрузки (мс относительно navigationStart)
  duration: 25.3,   // Общее время загрузки (мс)
  initiatorType: "link", // Тип инициатора (script, img, css, xmlhttprequest и др.)
  nextHopProtocol: "h2", // Протокол (http/1.1, h2, h3 и др.)
  workerStart: 0,
  redirectStart: 0,
  redirectEnd: 0,
  fetchStart: 150.5,
  domainLookupStart: 150.5,
  domainLookupEnd: 150.5,
  connectStart: 150.5,
  connectEnd: 150.5,
  secureConnectionStart: 150.5,
  requestStart: 150.5,
  responseStart: 175.5,
  responseEnd: 175.8, // Время завершения загрузки
  transferSize: 1024, // Общий размер переданных данных (включая заголовки)
  encodedBodySize: 900, // Размер тела ресурса после сжатия
  decodedBodySize: 1500, // Размер после распаковки
  serverTiming: [], // Серверные метрики (если есть)
}

```

Ключевые свойства

```
{
  duration: 25.3,   // Общее время загрузки (мс)
  transferSize: 1024, // Общий размер переданных данных (включая заголовки)
  encodedBodySize: 900, // Размер тела ресурса после сжатия
  decodedBodySize: 1500, // Размер после распаковки
}
```

Создадим класс для отслеживания и отправки метрик

```
1 const tracker = new CachePerformanceTracker();
2 tracker.start();
```

Метод start

```
start() {
  if (!window.performance?.getEntriesByType) return;

  // Собираем уже загруженные ресурсы
  performance.getEntriesByType('resource').forEach(entry => {
    this.classifyAndPush(entry);
  });

  // Настраиваем наблюдение за новыми ресурсами
  this.observer = new PerformanceObserver(list => {
    list.getEntries().forEach(entry => this.classifyAndPush(entry));
  });
  this.observer.observe({ type: 'resource' });

  // Отправляем метрики при закрытии страницы
  window.addEventListener('beforeunload', this.sendToSentry);
}
```

Вспомогательные методы

```
isStaticResource(entry) {
  const staticPatterns = [
    /\.(js|css|png|jpe?g|gif|svg|woff2?|ttf|ico)(\?.*)?$/i, // Расширения файлов
    /\/static\//,  // Пути
    /\/assets\//,
    /\/images\//,
    /\/fonts\//
  ];
  return staticPatterns.some(pattern => pattern.test(entry.name));
}
```

Метод для расчета метрик

```
calculate() {
  if (!resources.length) return null;

  let cachedCount = 0;
  let actualBandwidth = 0;
  let potentialBandwidth = 0;

  resources.forEach(res => {
    if (res.transferSize === 0) cachedCount++;  // transferSize = 0 означает кэш
    actualBandwidth += res.transferSize;
    potentialBandwidth += res.decodedSize;
  });

  return {
    cacheHitRate: parseFloat((cachedCount / resources.length).toFixed(4)),
    bandwidthSavings: parseFloat((1 - actualBandwidth / potentialBandwidth).toFixed(4)),
    totalResources: resources.length,
    cachedResources: cachedCount
  };
}
```

Метод для отправки в sentry

```
sendToSentry = () => {
  if (typeof Sentry === 'undefined') return;

  const metrics = this.calculateMetrics();

  Sentry.captureMessage('Cache Performance Metrics', {
    level: 'info',
    contexts: {
      cache: metrics,
      environment: {
        connection: navigator.connection?.effectiveType || 'unknown',
        memory: performance.memory?.usedJSHeapSize ?
          Math.round(performance.memory.usedJSHeapSize / 1024 / 1024) + 'MB' : 'N/A'
      }
    }
  });
```

Цель: решить проблему максимально простым способом

Заключение