Engineer

HTTP

⇄ запит і відповідь, на яких побудовано веб

Що це таке

HTTP, або Hypertext Transfer Protocol, — це набір правил, за якими браузер, застосунок чи скрипт просить у сервера щось і отримує відповідь; коли ця розмова йде всередині зашифрованого з'єднання TLS, той самий протокол називають HTTPS. Кожен обмін складається з двох половин. Клієнт надсилає запит: метод на кшталт `GET` чи `POST`, ціль, яку хоче отримати, набір полів заголовка і часом тіло. Сервер відповідає тризначним кодом стану, власними полями заголовка й, зазвичай, тілом. Завантаження сторінки, отримання зображення, виклик однієї програми з іншої — це один і той самий обмін, повторений багато разів.

```
GET /pricing/ HTTP/1.1
Host: example.com
Accept: text/html

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Cache-Control: max-age=600
```

Специфікація називає HTTP протоколом без стану. Кожен запит мусить нести все, що потрібно серверу, щоб його зрозуміти, а сам протокол нічого не пам'ятає між обмінами. Вхід до облікового запису, кошик і сесії будують поверх нього, здебільшого на cookie: це поле заголовка, яке сервер встановлює один раз, а клієнт повертає з кожним наступним запитом.

Словник у HTTP невеликий, зате точний. `GET` і `HEAD` — безпечні методи: вони просять представлення ресурсу й нічого не мають змінювати. `PUT` та `DELETE` ідемпотентні, тобто той самий запит, надісланий двічі, лишає сервер у тому ж стані, що й надісланий один раз. `POST` не обіцяє ні того, ні іншого. Коди стану згруповано за першою цифрою: 2xx означає успіх, 3xx —

перенаправлення, 4xx — проблему із запитом, 5xx — проблему на сервері. Тож клієнт, який бачить незнайомий код, усе одно приблизно розуміє, що сталося.

Значення всіх цих слів лишається сталим, хоча формат передавання вже двічі замінили. HTTP/1.1 пише повідомлення текстовими рядками поверх з'єднання TCP, і кожне з'єднання повертає відповіді по одній, у порядку запитів. HTTP/2 зберігає ті самі методи й поля, але пакує їх у двійкові кадри та переплітає багато запитів в одному з'єднанні. HTTP/3 передає ту саму семантику через QUIC, транспорт на основі UDP, який усуває властиве TCP транспортне блокування початку черги між незалежними потоками. Від 2022 року IETF публікує семантику в одному документі, а формат кожної версії — в окремому, тож запитання «що означає 404» і «як розміщені байти» тепер мають окремі відповіді.

Отже, HTTP чи HTTPS — це вибір не між двома протоколами. HTTPS — це HTTP, під яким лежить TLS: він шифрує обмін і підтверджує ім'я сервера. WebSocket починається як звичайний запит HTTP, а потім перемикає з'єднання на канал, де писати можуть обидві сторони. gRPC — це домовленість про віддалені виклики, що працює поверх HTTP/2.

Застосуйте поняття до рішення

Застосуйте «Що таке HTTP?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Wi-Fi, фаєрволи і віддалений доступ налаштовані так, щоб працювати можна було звідусіль, а заходити — лише своїм.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Керувала 30 інстансами Ubuntu Linux VPS — 30 серверів Linux під управлінням, із планом відновлення

Керувала 30 інстансами Ubuntu Linux VPS Зв'язатися з нами →

Чому це важливо

Майже кожна система, яка спілкується з іншими через інтернет, уже говорить HTTP. Тому справжнє питання інше: користуватися тим, що протокол визначає, чи сприймати його як трубу й будувати решту самотужки.

На користь першого варіанта є вагомий аргумент: довгий ланцюжок програм уже розуміє цей словник. Клієнтська бібліотека може повторити `GET`, на який не дочекалася відповіді, але `POST` мусить лишити як є, і так само має вчинити будь-який проксі посередині. Кеш може зберігати відповідь із

позначкою `max-age=600` десять хвилин і віддавати її наступному читачеві, не питаючи сервер. Моніторинг зараховує відповіді 5xx до помилок без додаткових підказок. API, що надсилає кожен виклик як `POST` на одну адресу й повідомляє про збій усередині відповіді 200, усього цього позбавляється, і кожен клієнт мусить вивчати його приватні правила.

У моделі є межі. Запит і відповідь добре пасують клієнтові, який знає, чого хоче, і значно гірше — серверу, якому є що повідомити першим. Через це живі оновлення зазвичай ідуть через WebSocket, server-sent events або регулярні повторні запити.

З відкритим HTTP у публічному інтернеті відповідь коротка. Браузери позначають його як небезпечний, будь-хто на шляху може прочитати й змінити вміст, а сертифікати безкоштовні, тож публічний трафік має йти через HTTPS. Вибір між HTTP/1.1, HTTP/2 і HTTP/3 — здебільшого налаштування сервера або периферії перед ним. Код застосунку лишається тим самим, бо кожен запит означає те саме в усіх трьох версіях.

Як це виглядає на практиці

Найвиразніший приклад на цьому сайті — API, яке перенесли на фреймворк, побудований довкола самої форми HTTP. HTTP API переїхало з Fiber на Huma v2, що працює поверх Fiber. Кожен ендпоінт оголошено через структури входу й виходу, і валідація випливає з типів, а не з написаних уручну перевірок. З тих самих оголошень виходить опис OpenAPI, а перевірка в CI порівнює маршрути, які реєструє сервер, з тими, що їх декларує опис. На 649 шляхах із 760 операціями паритет становить 100%, і він тримається, бо маршрут без опису завалює білд. Рівно два ендпоінти лишилися на чистому Fiber — ті, що працюють через WebSocket, де модель запиту й відповіді Huma не підходить.

Сам сайт використовує узгодження вмісту. Кожну сторінку опубліковано і як HTML, і як Markdown, і один зі способів отримати Markdown — це надіслати `Accept: text/markdown` і дати серверу вибрати. Та сама сторінка пояснює, чому адреса onion навмисно має звичайний `http://` : адреса кодує відкритий ключ сервісу, тож Tor автентифікує сервіс і шифрує канал ще до першого запиту HTTP.

HTTP трапляється і як інструмент в інфраструктурному коді. Коли розгортали другий сервер, налаштування брандмауерів обох провайдерів написали безпосередньо для їхніх REST API через узагальнене HTTP-завдання, а не через колекцію постачальника. Так зникла залежність разом з її версійним дрейфом, а платою стало ручне написання форми кожного запиту. Сторінка називає й іншу ціну: узагальнене HTTP-завдання не оголошує підтримки режиму перевірки, тож пробний прогін сценарію Ansible, який створює машину, пропускає в ньому кожне завдання. Тому перевірку проводять на сценарії налаштування брандмауера.

Де виникають проблеми

Більшість помилок із HTTP з'являється тоді, коли слово з його словника вживають в іншому значенні, ніж те, яке чує решта ланцюжка.

Помилка, що виглядає як успіх

API відповідає кодом 200 і кладе в тіло `"error": "card declined"`. Кеш може таку відповідь зберегти, дашборди зараховують її як успіх, а логіка повторних спроб у клієнті так і не спрацьовує. Надсилайте код, який відповідає ситуації, 4xx або 5xx, а суть проблеми описуйте в тілі. RFC 9457 визначає для цього формат JSON із типом `application/problem+json`.

Посилання, яке видаляє

Запит `GET` змінює стан, наприклад посилання для відписки чи видалення в листі. Пошукові роботи, попередній перегляд посилань у месенджерах, сканери безпеки й попереднє завантаження в браузері переходять за посиланнями самі, і кожен із них може запустити дію. Усе, що змінює дані, має стояти за `POST`, `PUT` або `DELETE`, а якщо посилання таки потрібне, — за сторінкою підтвердження.

Платіж, списаний двічі

Клієнт надсилає `POST`, з'єднання обривається до того, як прийшла відповідь, і клієнт пробує ще раз. Сервер тим часом уже обробив перший запит. HTTP не дає клієнтові способу розрізнити ці два випадки, тож це має робити API: приймати ключ ідемпотентності в полі заголовка й повертати збережений результат, коли ключ повторюється, або дозволити клієнтові самому вибрати ідентифікатор нового ресурсу і створити його через `PUT`.

Форма, що перетворюється на GET

Сервер відповідає на `POST` кодом 301 або 302, щоб відправити його деінде. З історичних причин специфікація дозволяє агентові користувача змінити метод на `GET` саме для цих двох кодів, і багато хто так робить, тож тіло губиться дорогою. Якщо метод має пережити перенаправлення, потрібні коди 307 чи 308.

Чужа версія з кешу

Одна URL-адреса віддає HTML або Markdown, українською чи англійською, залежно від полів заголовка запиту. Кеш, якому про це не сказали, зберігає ту версію, яку побачив першою, і віддає її наступному читачеві. Відповідь мусить назвати поля, від яких залежить, наприклад `Vary: Accept`, щоб кеш тримав окрему копію для кожного варіанта, а не одну на всю адресу.

Що почитати далі

- RFC 9110: HTTP Semantics — методи, коди стану й поля заголовка, визначені один раз для всіх версій.
- RFC 9111: HTTP Caching — коли збережену відповідь можна використати повторно і що кешу каже `Cache-Control`.
- RFC 9112: HTTP/1.1 — текстовий формат передавання, з’єднання і межі повідомлень.
- RFC 9113: HTTP/2 — двійкові кадри й багато потоків в одному з’єднанні.
- RFC 9114: HTTP/3 — та сама семантика поверх потоків QUIC.
- RFC 9457: Problem Details for HTTP APIs — стандартне тіло JSON для опису того, що пішло не так.

Словник

Обирайте за фактичними умовами

Опишіть шлях розгортання, експлуатаційні ризики та чергову команду. Ми можемо визначити зміну з перевірками, шляхом відновлення та чіткою відповідальністю.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке HTTP?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/http/