Jak działa Internet – HTTP

Po lekturze dwóch poprzednich artykułów tej serii wiemy już, jak skonfigurować własną sieć domową (rozumiemy działanie routerów i switchy) oraz jak uruchomić prosty serwis internetowy stworzony przez kogoś innego (potrafimy przypisać nazwy DNS do konkretnych adresów IP). Teraz nadszedł czas, aby zgłębić „język", którym posługuje się większość serwisów – protokół HTTP. Dokładne jego zrozumienie otworzy nam drogę do samodzielnej implementacji serwisów, lepszego rozumienia komunikacji internetowej na poziomie aplikacji, a także do efektywnego wykrywania i naprawiania problemów z komunikacją.

CZYM JEST HTTP?

HTTP (ang. *HyperText Transfer Protocol*) to jeden z protokołów warstwy aplikacji, działający na siódmej warstwie modelu OSI [2]. Został stworzony z myślą o komunikacji między klientami (najczęściej przeglądarkami internetowymi) a serwerami (komputerami przechowującymi dane) [1]. Ta relacja stanowi podstawę najprostszej wersji architektury serwisów internetowych. W kolejnych rozdziałach (Proxy – mitmproxy, Cache) omówimy również bardziej zaawansowane elementy tej architektury, jak serwery proxy, a także *cache* [2].

HTTP zazwyczaj korzysta z protokołu TCP do nawiązywania połączenia oraz do zapewnienia integralności przesyłanych danych. Warto wspomnieć, że HTTP powstał już w 1991 roku [3] w wersji HTTP/0.9. Od tamtej pory doczekał się wydania wielu rozszerzeń oraz kilku nowych istotnych wersji, które wprowadzały usprawnienia w zakresie funkcjonalności (HTTP/1.0 oraz HTTP/1.1) oraz wydajności (HTTP/2.0 oraz HTTP/3).

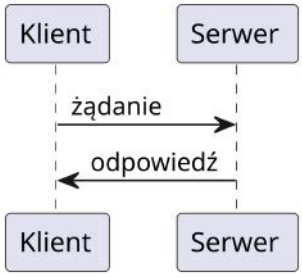
W artykule skupimy się głównie na wersji HTTP/1.1 protokołu, z odniesieniami do HTTP/2, tam gdzie będzie to potrzebne. Protokół HTTP/1.1 jest tekstowym protokołem, przez co też łatwiejszy do zrozumienia, analizy i implementacji, a mimo upływu czasu wciąż szeroko stosowany. HTTP/2, choć funkcjonalnie zbliżony do HTTP/1.1, wprowadza istotne zmiany w sposobie przesyłania danych. Mimo tych różnic podstawowe zasady działania protokołu pozostają bardzo podobne.

ARCHITEKTURA HTTP [7]

Jak już wspomnieliśmy, HTTP opiera się na architekturze klient-serwer, gdzie to klient zawsze inicjuje komunikację. Rolą serwera jest nasłuchiwanie nadchodzących żądań (ang. *requests*) i wysyłanie odpowiednich odpowiedzi (ang. *responses*). Zarówno żądanie, jak i odpowiedź są określane jako „wiadomości" (ang. *messages*).

Często możemy spotkać inną nazwę dla klienta – *user agent*. Jest to nic innego jak program, który wysyła zapytania w imieniu użytkownika. Najbardziej popularnymi „user-agentami" są oczywiście nasze przeglądarki internetowe, chociaż może to być dowolne inne oprogramowanie, które wysyła żądania i przetwarza odpowiedź. Serwery, które mają docelowe informacje o pytane zasoby, nazywamy serwerami backendowymi.

Najprostszy sposób komunikacji przedstawiono na Rysunku 1.



Rysunek 1. Najprostsza architektura komunikacji HTTP

Jak nietrudno się domyślić, tak prosta architektura często nie zawsze jest wystarczająca w rzeczywistych scenariuszach. Serwery obsługujące zapytania zazwyczaj nie działają w pojedynkę.

Przede wszystkim serwisy internetowe potrzebują odpowiedniej redundancji. Oznacza to, że żądania obsługiwane są przez wiele serwerów jednocześnie. Takie podejście zapewnia ciągłość działania w przypadku awarii jednego z serwerów, a także pozwala równomiernie rozłożyć obciążenie serwerów.

Aby obsłużyć taką architekturę, potrzebne są nowe elementy i w naszej architekturze pojawiają się tzw. bramki (ang. *gateways*) (inna popularna nazwa to reverse-proxy) i load balancery. Jeśli kojarzycie termin default gateway z poprzednich artykułów, to doprecyzujmy, że nie jest to ten sam gateway, o którym mówimy tutaj.

Zadaniem bramek jest właśnie odbieranie zapytań od klientów, ich wstępne przetwarzanie (jeśli jest potrzebne) oraz dystrybucja zapytań do jednego z serwerów.

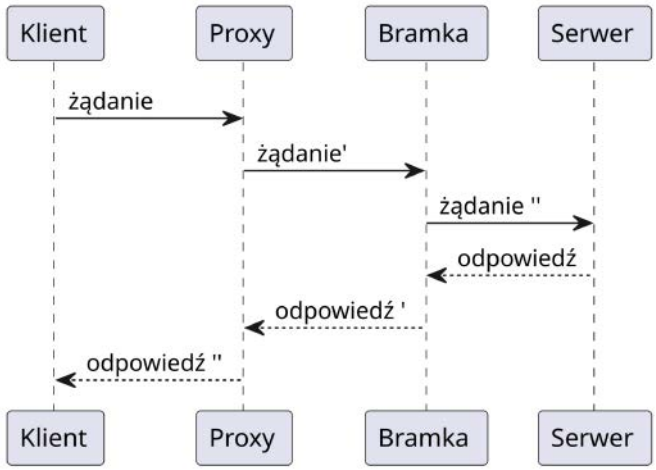
Co więcej, mogą one pełnić rolę cache'a. Jeśli klient pyta o zasób, który jest taki sam dla wszystkich klientów, nie ma potrzeby, żeby zapytanie trafiało bezpośrednio do serwera źródłowego i tym samym go obciążało. Dodatkowo skracają one zazwyczaj fizyczny dystans między klientem a odpowiadającym urządzeniem, co zmniejsza opóźnienie odpowiedzi.

Cache może znajdować się również po stronie klienta (np. w przeglądarkach internetowych). Nietrudno się jednak domyślić, że nie każdy zasób może być cache'owany. Protokół HTTP definiuje, w jaki sposób sterować zachowaniem cache'a (więcej na ten temat w sekcji Cache).

Tak samo, jak serwery mogą stać za bramkami, tak klienci mogą korzystać z tzw. proxy. Proxy może być używane z wielu powodów. Jednym z nich jest chowanie prawdziwego adresu IP (chociaż istnie-

ją lepsze narzędzia do tego celu). Innym powodem jest chęć analizy naszego ruchu i modyfikowania go w locie. Jest to szczególnie przydatne w fazie implementacji serwisu. Przykładem aplikacji proxy jest aplikacja mitmproxy, której poświęciliśmy osobną sekcję artykułu.

Podsumowując, nasza architektura po uwzględnieniu wszystkich opisanych elementów może wyglądać jak ta na Rysunku 2 (może być nawet bardziej skomplikowana, ponieważ ominęliśmy niektóre elementy architektury).



Rysunek 2. Typowa architektura komunikacji HTTP

Warto wiedzieć (czego Rysunek 2 nie oddaje), że każdy z elementów może jednocześnie obsługiwać wielu klientów. Na przykład bramka może jednocześnie obsługiwać wiele zapytań od proxy oraz od klientów, a proxy może również odbierać żądania od wielu klientów.

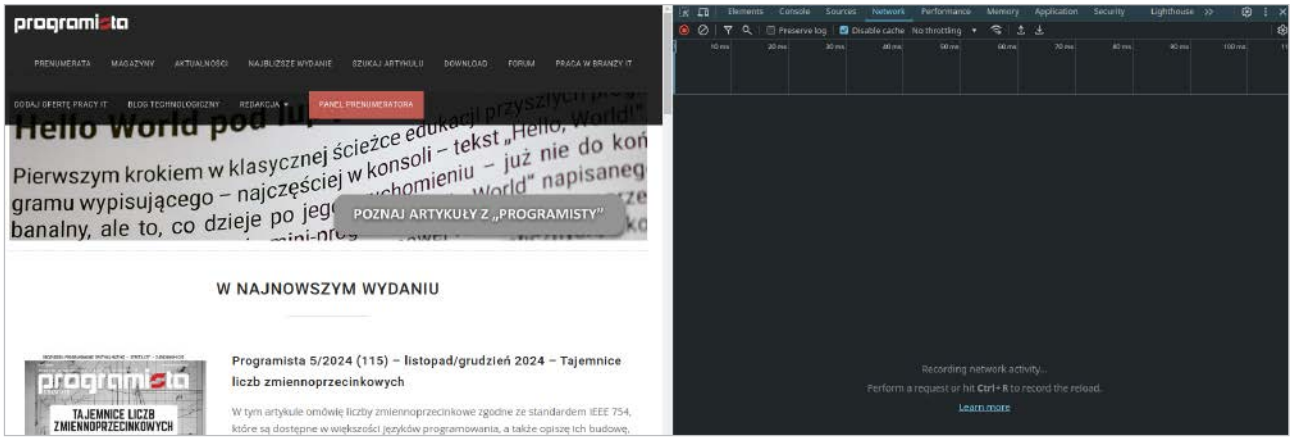
To tyle słowem wstępu. Czas zagłębić się w szczegóły zapytań HTTP.

WIADOMOŚCI

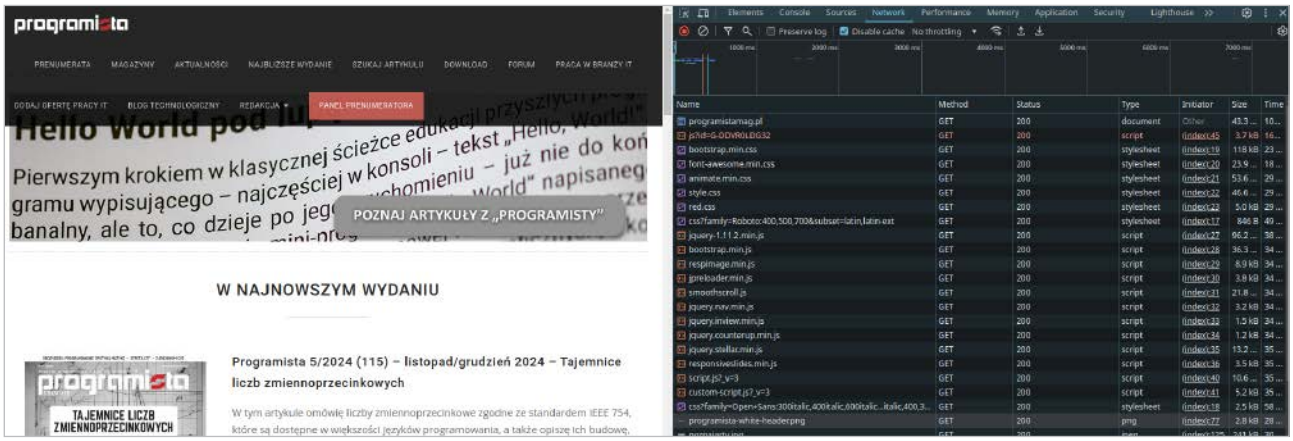
Czas włączyć naszego ulubionego user-agenta (czyli przeglądarkę) i udać się na stronę, którą wszyscy znamy: https://programistamag.pl, następnie otwórzmy narzędzia deweloperskie (developer tools). Otworzenie ich zależy od konkretnej przeglądarki. W Chrome, Brave lub Chromium możemy to zrobić skrótem klawiszowym Ctrl+Shift+I lub klikając prawym przyciskiem myszy w dowolne miejsce na stronie i wybierając opcję Inspect. Następnie przechodzimy do zakładki Network, ponieważ to ona nas interesuje.

Nasz ekran powinien wyglądać podobnie do zrzutu ekranu na Rysunku 3.

Jak widzimy, nasza zakładka Network jest pusta. Aby zobaczyć cały ruch sieciowy związany z odwiedzaną stroną, musimy ją odświeżyć. Wynik możemy podziwiać na Rysunku 4.



Rysunek 3. Narzędzia deweloperskie w przeglądarce Brave



Rysunek 4. Zakładka Network ze wszystkimi żądaniami związanymi z odwiedzeniem strony

Przyjrzyjmy się zatem temu, co widzimy. Kolumny, które nas najbardziej interesują, to nazwa, metoda i status. Po wybraniu danego żądania pojawi się nam więcej szczegółów rozbitych na sekcje: informacje ogólne, nagłówki odpowiedzi (ang. *response headers*) oraz nagłówki zapytania (ang. *request headers*).

W telegraficznym skrócie nazwa odnosi się do zasobu – może to być dokument HTML, obraz, styl CSS lub dowolny inny element. Każdy zasób ma swoją lokalizację, którą musimy znać, aby móc się do niego odwołać. Jest ona w określonym formacie, zwanym URI. Wkrótce przyjrzymy się temu formatowi nieco bliżej.

Wszystkie żądania na naszym zrzucie ekranu mają metodę GET. Metoda mówi serwisowi, co chcemy zrobić z danym zasobem – w przypadku GET chcemy po prostu pobrać zasób. Inne popularne metody to:

- PUT, POST – do aktualizacji zasobu, wysłania danych do przetworzenia na serwer,
- DELETE – prośba o usunięcie zasobu.

To serwer decyduje, które metody są dostępne. Może się również zdarzyć, że wybrane metody są dostępne tylko dla wybranych użytkowników – w przeciwnym razie każdy mógłby dokonywać modyfikacji na zasobach. Co więcej, serwery mogą implementować własne metody nieopisane w standardzie HTTP.

No i w końcu statusy odpowiedzi. Z pewnością nie raz spotkaliśmy się ze stroną internetową, wyświetlającą status 404 Not Found, a może nawet 500 Internal Server Error. Statusy te służą do informowania klienta o rezultacie przetworzenia żądania. W skrócie, statusy dzielimy na 4 kategorie:

- 200-299 – świadczą o sukcesie, z których najczęstszy to po prostu 200.
- 300-399 – zazwyczaj mówią nam o tym, że zasób, o który pytamy, istnieje, ale został przeniesiony pod nowy adres URI.
- 400-499 – kody błędu zapytania. Serwer nie mógł przetworzyć żądania ze względu na błąd w nim zawarty. Najpopularniejsze z nich to 404, który informuje nas o tym, że żądany zasób nie istnieje.
- 500-599 – najbardziej problematyczne statusy. Wskazują na błąd po stronie serwera, który uniemożliwił przetworzenie żądania. Jako klienci często nie możemy nic z tym zrobić, natomiast jako programiści musimy dany problem po prostu naprawić.

Każde zapytanie, tak jak każda odpowiedź, zawiera sekcję nagłówków. To właśnie w nich przesyłane są dodatkowe informacje o zasobie. Mogą to być instrukcje dotyczące cacheowania, informacje o typie i formacie zasobu, a także o sposobie kompresji. Jeden z nagłówków to `User-Agent`, który identyfikuje oprogramowanie użyte do wysłania żądania – zazwyczaj przeglądarkę. Odpowiedzi HTTP mogą być w ten sposób dostosowane pod konkretną przeglądarkę, co miało znaczenie zwłaszcza podczas pierwszej wojny przeglądarkowej [4, 5], kiedy to różniły się one od siebie w niekompatybilny sposób.

URI/URL

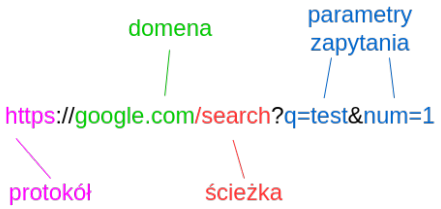
Zacznijmy od tego, że w praktyce terminy URI [9] i URL [8] są używane naprzemiennie. Nie jest to dziwne, ponieważ każdy URL jest również URI (tak jak każdy kwadrat jest prostokątem), ale nie każde URI to URL. W tym artykule również będziemy używać tych nazw naprzemiennie. URI, które nie są URL, nie są używane w komunikacji HTTP.

W skrócie, URL wskazuje lokalizację konkretnego zasobu w sieci. Przykład takiego URLa jest przedstawiony na Rysunku 5.

Rysunek 5. URL do obrazu przedstawiającego okładkę jednego z wydań magazynu „Programista”.

Jest to przykładowy zasób z domeny programistamag.pl. Taki URL w swojej podstawowej postaci składa się z protokołu HTTP (lub jego zaszyfrowanej postaci HTTPS), domeny identyfikującej host, na którym znajduje się zasób (oddzielonej od protokołu znakami `://`), oraz ścieżki do zasobu.

To jednak nie wszystko. URL może zawierać także dodatkowe elementy, takie jak parametry zapytania (ang. *query parameters*) czy fragmenty (ang. *fragments*). Przykład z parametrami zapytania możemy zobaczyć, korzystając chociażby z wyszukiwarki Google:



Rysunek 6. URL do wyników wyszukiwania tekstu „test” w przeglądarce Google

Widzimy tutaj dodatkowe parametry zasobu:

- `q` – odpowiada za tekst wyszukiwania, czyli to, co wpisujemy w pole wyszukiwarki,
- `num` – ilość wyników prezentowanych na stronie.

Warto zauważyć, że pierwszy parametr jest oddzielony od ścieżki znakiem `?`, natomiast kolejne parametry są rozdzielane znakiem `&`.

Parametry zapytania to zazwyczaj pary klucz-wartość. Nazwa parametru znajduje się po lewej stronie znaku `=`, a jego wartość po prawej. Warto jednak pamiętać, że jest to jedynie konwencja. Możliwy jest parametr składający się wyłącznie z nazwy (zwłaszcza popularny dla wartości logicznych) czy używanie znaku innego niż `=` do rozdzielania klucza od wartości.

Ostatnią nieomówioną częścią URL są tzw. fragmenty. Jak sama nazwa wskazuje – odnoszą się one jedynie do fragmentu zasobu. Żeby wskazać na część zasobu, i tak potrzebny jest cały zasób, tak więc fragmenty w przeciwieństwie do całej reszty URL nie grają roli, jeśli chodzi o komunikację klient-serwer (klienci po prostu nie wysyłają tej części URL do serwisów). Skoro fragmenty nie są wysyłane do serwera, to ich obsługa siłą rzeczy pozostaje po stronie klienta.

WORDPRESS JAKO HEADLESS CMS. INTEGRACJA Z NOWOCZESNYM FRONT-ENDEM DLA ELASTYCZNYCH APLIKACJI

INDEX: 285358

www.programistamag.pl

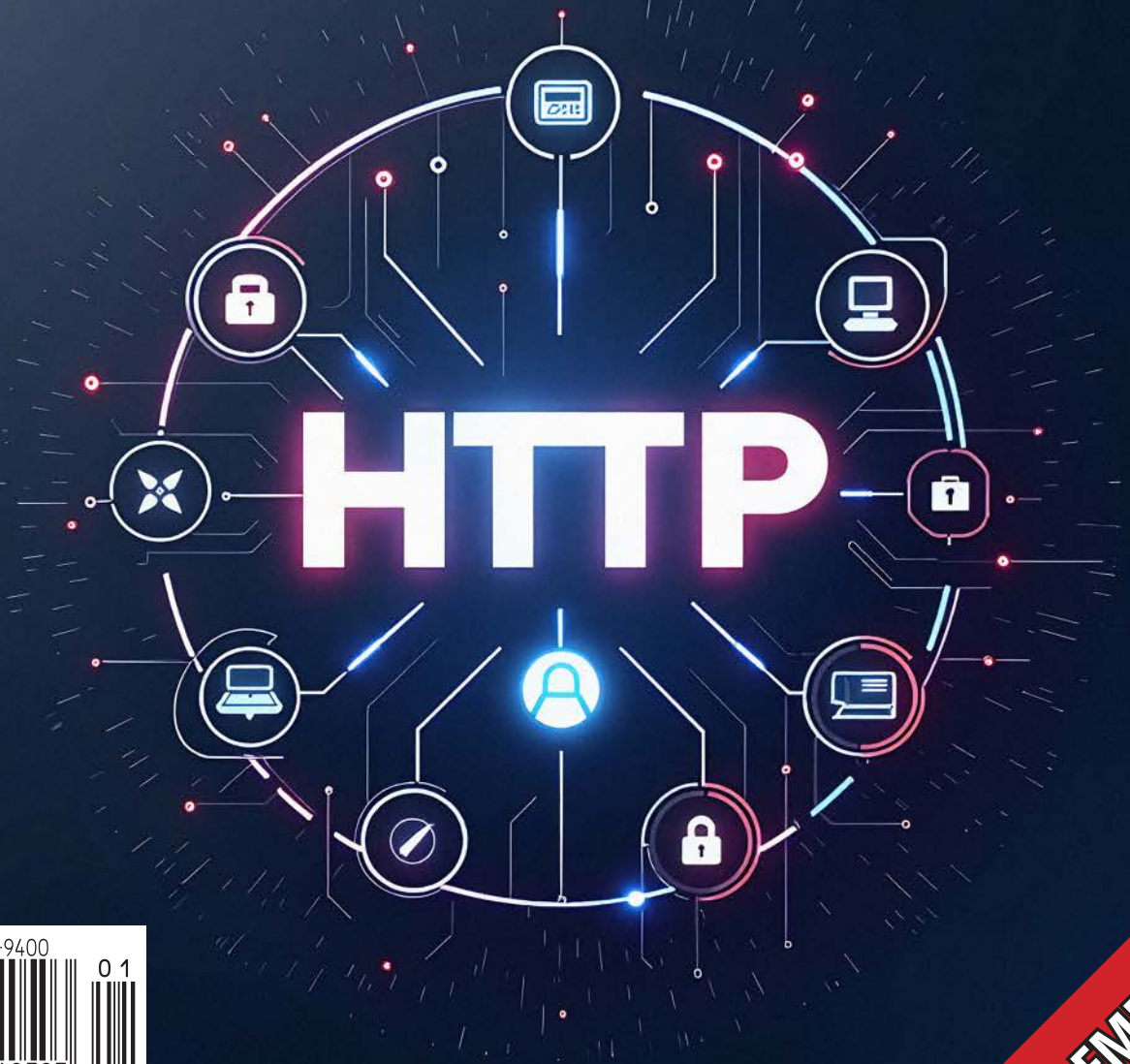
Magazyn programistów i liderów zespołów IT

programista

1/2025 (116)

Cena 32.90 zł (w tym VAT 8%)

JAK DZIAŁA INTERNET



ISSN 2084-9400

01

9 772084 940503

Współbieżność CSP w języku Go

Rekin w strumieniu... ...czyli hakujemy Stream Decka

Gwiazda m… …orutyny, czyli Sea… …aktyce

CSS… …lasses …adzie TailwindCSS

NOWY NUMER JUZ W EMPIKACH