2. Протокол HTTP. Принципи функціонування

HTTP (HyperText TransfeProtocol, Протокол передачі гіпертексту). Це список правил, за якими комп'ютери обмінюються даними в інтернеті. HTTP вміє передавати всі можливі формати файлів – наприклад, відео, аудіо, текст. Але при цьому складається лише з тексту.

Наприклад, якщо вписати у адресному рядку браузера `www.victoria.lviv.ua`, він складає запит і надсилає його до серверу, щоб отримати HTML-сторінку сайту. Після оброблення запиту сервер надсилає відповідь у вигляді потрібної сторінки.

Протокол HTTP використовують з 1992 року. Він простий, але досить функціональний. HTTP знаходиться на самій вершині моделі OSI (на прикладному рівні), де програми обмінюються даними. Працює HTTP за допомогою протоколів TCP/IP та використовує їх, щоб передавати дані.

Окрім HTTP в Інтернеті працює ще протокол HTTPS (HyperText Transfer Protocol Secure, Захищений протокол передачі гіпертексту). Він потрібний для безпечної передачі даних по Мережі. Все відбувається за тими ж принципами, що й у HTTP, але перед надсиланням дані додатково шифруються, а потім розшифровуються на сервері.

Наприклад, HTTPS використовують під час введення даних банківської картки чи паролів на сайтах — та й загалом більшість сучасних веб-додатків використовують саме його.

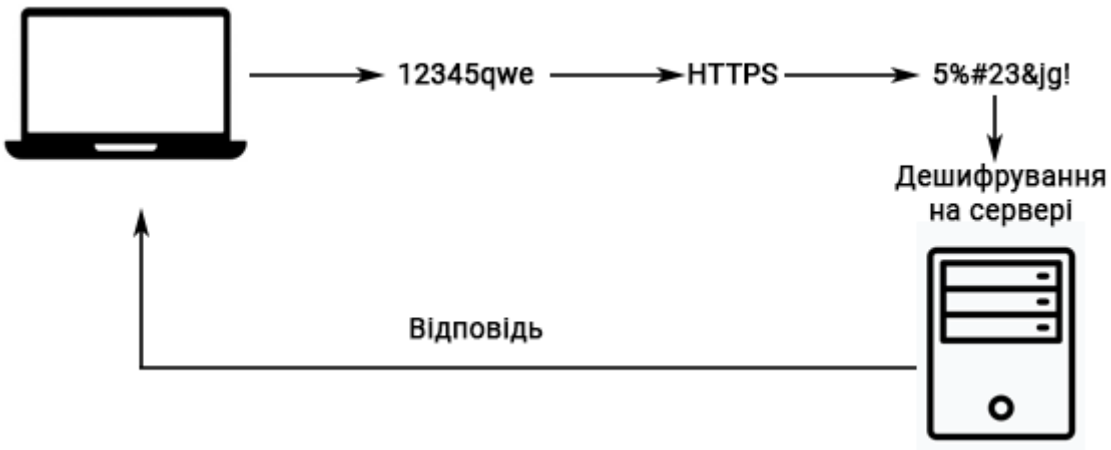

Рис.1. Шифрування та дешифрування даних

Склад протоколу HTTP

HTTP складається з правил, що описують взаємодію двох елементів: клієнта та сервера. Клієнт надсилає запити та чекає на дані від сервера. А сервер чекає, поки прийде черговий запит, обробляє його і повертає відповідь до клієнта.

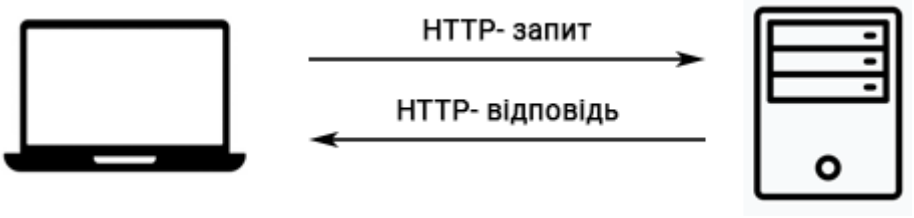

Рис.2. Взаємодія клієнта і сервера

Зазвичай, цей зв'язок між клієнтом та сервером має посередників у вигляді проксі-серверів. Вони потрібні для різних операцій – наприклад, для безпеки та конфіденційності, кешування чи розподілу навантаження на сервери.

Тому, типова процедура надсилання HTTP-запиту від клієнта виглядає так:

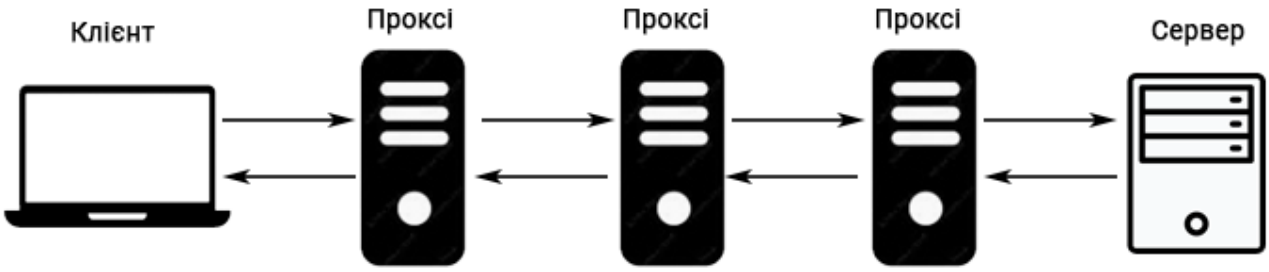

Рис.3. Зв'язок між клієнтом та сервером через проксі-сервери

- **Клієнтом** може бути будь-який пристрій, через який користувач запитує дані. Зазвичай, для веб-додатків в ролі клієнта виступає веб-браузер. Головна особливість клієнта – він завжди ініціює запит.
- **Сервер** — це пристрій, який опрацьовує запити клієнта. Він може складатися як з одного комп'ютера, так і кластера комп'ютерів. Також, на одній фізичній машині може бути кілька віртуальних серверів.
- **Проксі-сервер**и – це другорядні сервери, які розміщуються між клієнтом та головним сервером. Вони обробляють HTTP запити, а також відповіді на них. Найчастіше проксі-сервери використовують для кешування та стиснення даних, обходу обмежень та анонімних запитів. Зазвичай, між клієнтом і основним сервером знаходиться один або кілька таких проксі-серверів.

Функціонування HTTP-протоколу

Весь процес передачі HTTP-запиту можна розділити на п'ять кроків.

Крок перший – зазначення URL-адреси в адресному рядку браузера

Щоб надіслати HTTP-запит, потрібно використовувати URL-адресу (Uniform Resource Locator, Уніфікований покажчик ресурсу). Він вказує браузеру, що потрібно використовувати протокол HTTP, а потім отримати файл з цієї адреси назад. Зазвичай, URL-адреса починається з зазначення протоколу http:// або https:// (залежить від версії протоколу).

Наприклад, https://lpnu.ua - це URL-адреса головної сторінки Львівської політехніки. В URL-адресі можуть бути і піддомени - https://lpnu.ua/ikta, це головна сторінка Інституту комп'ютерних технологій, автоматики та метрології.

Крок другий – з'ясування IP-адреси сервера

Для користувачів доменна адреса – це набір зрозумілих слів: google.com, lpnu.ua, victoria.lviv.ua. Але для комп'ютера ці слова є набір незрозумілих символів.

Тому, браузер надсилає введені символи до локального DNS-серверу на сервері провайдера, де з'ясовується відповідність даної URL-адреси веб-ресурсу до IP-адреси сервера, на якому розміщено даний ресурс.

DNS розшифровується як «доменна система імен» (Domain Name System), і представляє собою ієрархічну структуру DNS-сервери різних рівнів, що містять великі таблиці з усіма зареєстрованими іменами для сайтів та IP-адреси (їх або серверів, на яких розміщено сайти).

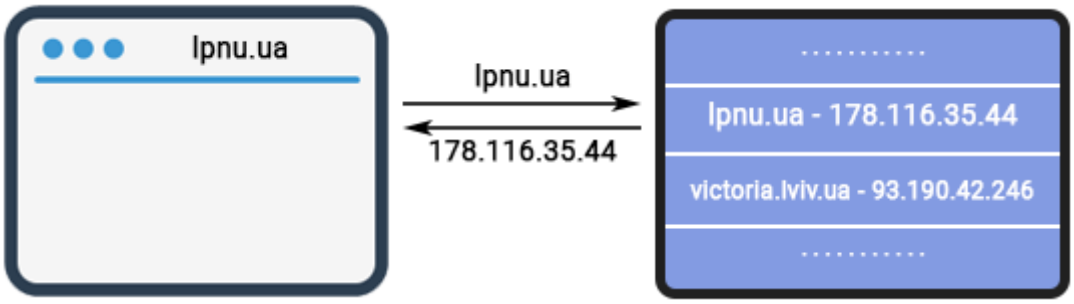

Рис. 4. Запит до DNS-серверу

Пригадати з курсу «Глобальні інформаційні мережі»

Крок третій – формування та надсилання HTTP-запиту браузером

DNS повертає до браузеру IP-адресу, з якою він вже вміє працювати. Браузер починає складати HTTP-запит із вкладеною в нього IP-адресою.

HTTP-повідомлення (HTTP-запити та HTTP-відповіді) мають однакову структуру (зміст може відрізнятися), яка представлена у вказаному на рисунку порядку.

Рис.5. Структура HTTP-повідомлення

- У стартовому рядку вказується тип запиту, адреса ресурсу та версія протоколу, код стану тощо.
- У заголовках містяться відомості про тіло повідомлення та застосовані параметри передачі.
- У тілі повідомлення передаються самі дані, відокремлені від заголовків порожнім рядком.

У повідомленні обов'язково повинні бути присутніми стартовий рядок та заголовок, тіла може не бути.

Рис.6. Типовий вигляд HTTP-запиту

На рис.6. показано чотири елементи:

- метод - "GET",
- URI - "/",
- версія HTTP - "1.1"
- адреса хосту - "lpnu.ua".

Метод

Метод - це дія, яку клієнт чекає від сервера. Щоб сервер зрозумів, чого хоче клієнт, використовуються HTTP-методи, наприклад, надіслати HTML-сторінку сайту або завантажити документ.

Протокол HTTP може застосовувати різні методи, основними з яких є:

Метод	*Характеристика*
OPTIONS	Визначає можливості сервера.
GET	Для отримання необхідних даних від сервера. В URI-адресі міститься необхідна інформація для визначення місця розташування ресурсу на сервері. Наприклад, відео з YouTube або сторінка з сайту кафедри.
HEAD	Для отримання метаданих про HTML-сторінку сайту. Це дані, які знаходяться у <head> елементі HTML-файлу. Відповідь сервера не містить тіла. Цей метод потрібний для валідації посилань.
POST	Для передачі відомостей від користувача щодо створення нового ресурсу. Запити за методом POST зазвичай містять дані для створення нового ресурсу, наприклад, повідомлення в Telegram або новий відеоролик в YouTube.
PUT	Відповідає за оновлення існуючого ресурсу. У вмісті можуть знаходитися оновлені дані для ресурсу.
PATCH	Перезаписує лише частину ресурсу.
DELETE	Для видалення даних.
TRACE	Дозволяє відстежувати інформацію, що змінюється проміжними серверами у запитах.

CONNECT	Для створення зашифрованого тунелю між клієнтом та сервером за допомогою проксі.

Найчастіше на практиці використовуються методи GET, POST, HEAD.

URI (Uniform Resource Identifier, Уніфікований ідентифікатор ресурсу)

Це повна адреса сайту в Мережі. Вона складається з двох частин: URL (Uniform Resource Locator) та URN (Uniform Resource Name).

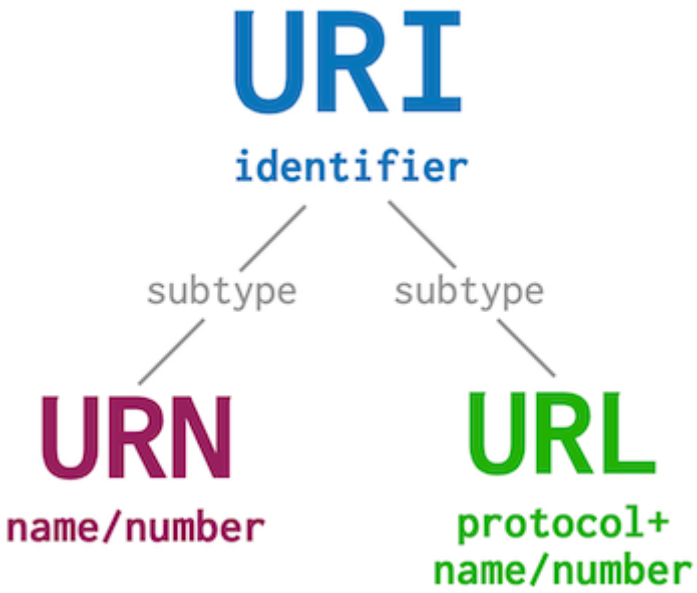

Рис.7. Склад уніфікованого ідентифікатору ресурсу

- URI – ім'я та адреса ресурсу в мережі, містить URL та URN.
  https://lpnu.ua/themes/lpnu/images/vlp-logo.png
- URL - адреса хоста в мережі, визначає місцезнаходження та спосіб звернення до нього.
  https://lpnu.ua/, https://lpnu.ua/ikta
- URN – ім'я ресурсу в мережі, визначає лише назву ресурсу, але не вказує, як до нього підключитися
  `themes/lpnu/images/vlp-logo.png`

URI і URL є однією з найбільших плутанини, і плутанина виникає тому, що обидва є ідентифікаторами ресурсів, але URL є типом URI. Усі URL-адреси є URI, але не всі URI є URL-адресами.

Справжня різниця між a URI та a URL полягає в тому, що a URI може бути просто ім'ям (`lpnu.ua`) або ім'ям у поєднанні з протоколом про те, як туди потрапити (https://), тоді як URL– це завжди комбіноване ім'я з протоколом (https://lpnu.ua) .

Версія HTTP

Версія HTTP вказує, яку версію HTTP браузер використовує під час надсилання запиту. Якщо її не вказувати, за замовченням використовується версія 1.1. Вона потрібна, щоб сервер повернув відповідь HTTP з тією ж версією протоколу HTTP і не створив помилок з читанням у клієнта.

Адреса хоста

Адреса хоста потрібна, щоб вказати, з якого сайту клієнт намагається отримати дані. Адресу вказують у вигляді домену, але вона відразу змінюється на IP-адресу перед надсиланням запиту за допомогою DNS.

Крок четвертий – формування та надсилання HTTP-відповіді сервером

Після отримання та обробки HTTP-запиту сервер створює відповідь та надсилає її назворот до клієнту. У ньому містяться додаткова інформація (метадані) і дані, що запитуються.

Проста HTTP-відповідь виглядає так:

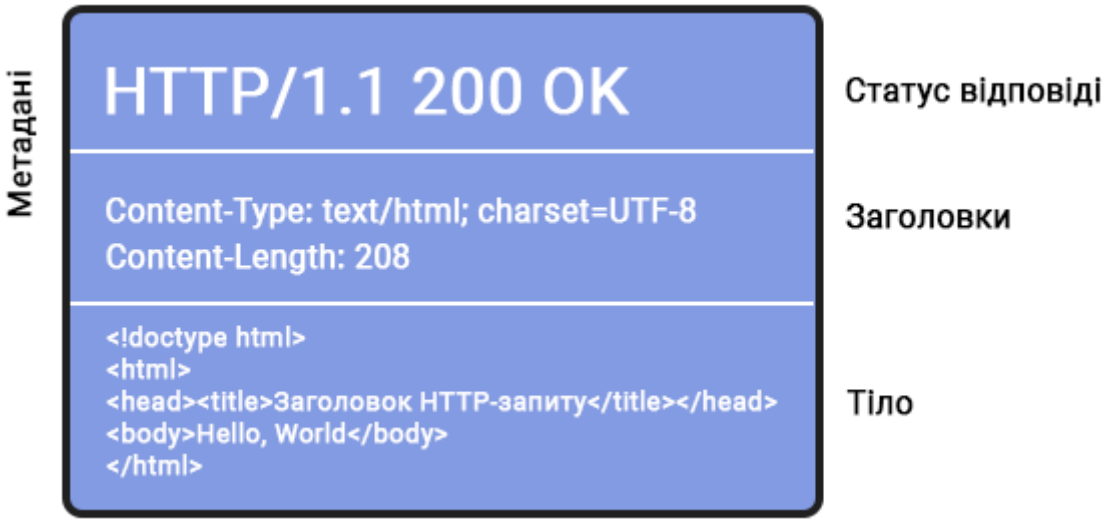

Рис.8. Типова структура HTTP-відповіді

Тут три головні частини:

- Статус відповіді - HTTP/1.1 200 OK.
- Заголовки Content-Type і Content-Length.
- Тіло відповіді - HTML-код.

Статус відповіді

Маючи URL-адресу та методи, клієнт може ініціювати запити до сервера. У відповідь сервер надсилає відповіді з кодами стану та вмістом повідомлень. Код статусу – важливий компонент повідомлення, що показує стан пристрою або процесу.

Дивитися статуси https://umbraco.com/knowledge-base/http-status-codes/.

Статусів відповіді існує 5 класів:

1. Інформаційні (1xx).
2. Успішна передача (2xx).
3. Перескерування (3xx).
4. Помилки на стороні клієнта (4xx).
5. Помилки на стороні сервера (5xx).

Статус відповіді містить версію протоколу HTTP, який клієнт вказав у запиті HTTP. Після неї йде код статусу відповіді і пояснювальний опис статусу відповіді.

У специфікації HTTP встановлюються певні діапазони чисел для конкретних типів відповідей.

1xx: Інформація про процес передачі

Цей клас кодів в HTTP/1.1 використовується для попереднього спілкування з клієнтом і сервером. Сервер може надіслати відповідь на повідомлення клієнта із заголовком `Expect: 100-continue` (100-Продовжити), інструктуючи клієнта, продовжити надсилати останню частину запиту або ігнорувати повідомлення, якщо воно вже було надіслано.

При роботі через HTTP/1.0 повідомлення з такими кодами повинні ігноруватися (у версії 1.1 клієнт повинен бути готовий прийняти цей клас повідомлення як звичайну відповідь, але серверу не потрібно надсилати що-небудь).

2xx: Інформація про успішне прийняття та обробку запиту клієнта

Коди цього класу повідомляють клієнту, що його запит успішно виконано. Найбільш часто зустрічається код 200 OK. На запити за методом GET сервер надсилає у відповідь запитувані дані в тілі повідомлення. Нижче наведено інші коди:

- 200 OK - дані успішно отримані.
- 201 Created (Створено) означає, що запит успішний, а дані створено. Його використовують, щоб підтверджувати успіх запитів PUT або POST.
- 202 Accepted (Прийнято): запит прийнято до обробки.
- 204 No Content (Немає вмісту): тіло повідомлення не передається.
- 205 Reset Content (Скинути вміст): клієнт повинен скинути введені користувачем дані.
- 206 Partial Content (Частковий вміст): вказує, що у відповіді міститься лише частина ресурсу.

3xx: Перескерування

Цей код вказує клієнту, що необхідно виконати додаткову дію. Найбільш простий варіант – виконання запиту за іншою URL-адресою для отримання запитуваного ресурсу.

- 301 Moved Permanently (Постійно переміщено): об’єкт запиту постійно перенесено на нову URL-адресу.
- 303 See Other (Дивитися інше): запитаний об’єкт тимчасово перенесено на нову URL-адресу.
- 304 Not Modified (Не модифіковано): сервер виявив, що ресурс не було змінено, і клієнту слід використовувати копію з кешу.

4xx: Інформація про помилки зі сторони клієнта

Ці коди використовуються, коли сервер вважає, що клієнт допустив помилку: помилковий запит або запит, недоступний для ресурсу клієнта. Найбільш популярні повідомлення в даному випадку:

- 400 Bad Request (Поганий запит): у запиті виявлено помилку. Зазвичай, це трапляється у зв'язку із запитами POST та PUT, коли дані не пройшли перевірку або представлені у неправильному форматі.
- 401 Unauthorized (Неавторизований): для здійснення запиту необхідна аутентифікація.
- 403 Forbidden (Заборонено): сервер відмовив клієнту на доступ до вказаного ресурсу.
- 404 Not Found (Не знайдено): запитуваного ресурсу не існує на сервері.

- 405 Method Not Allowed (Метод недопустимий): у рядку запиту використано недопустимий метод HTTP або ж сервер не підтримує цей метод.
- 409 Conflict (Конфлікт): сервер не зміг виконати запит, оскільки клієнт спробував змінити ресурс. У більшості випадків конфліктна ситуація виникає при спільному редагуванні ресурсу за допомогою запитів за методом PUT.

5xx: Інформація про помилки зі сторони сервера

Цей тип кодів використовується для повідомлень про неуспішне виконання операції на сервері.

- 500 Internal Server Error (Внутрішня помилка сервера): сервер зіткнувся з несподіваною помилкою, яка завадила йому виконати запит.
- 501 Not Implemented (Не реалізовано): сервер на даний момент не підтримує можливості, що необхідні для обробки запиту.
- 502 Bad Gateway (Неправильний шлюз): сервер, з яким намагався з'єднатися комп'ютер користувача, отримав неправильну відповідь сервера рівнем вище. Найчастіше це відбувається через проблеми в роботі DNS, проксі чи хостингу.
- 503 Service Unavailable (Сервіс недоступний): сервер не має можливості обробляти запити з технічних причин або перевантажень. Зазвичай, сервер навіть не буде відповідати, і запит визначає ліміт часу очікування від сервера.

Заголовки Content-Type і Content-Length

Заголовки HTTP важливі для контролю, оскільки кеш та браузери обробляють контент. Заголовки допомагають браузеру розібратися з отриманими даними та подати їх у правильному вигляді. Наприклад, заголовок Content-Type повідомляє, який формат файлу прийшов і які додаткові параметри, а Content-Length — скільки місця в байтах займає цей файл.

Але багато розробників використовується їх неправильно або безглуздо, витрачаючи зайві ресурси в критичний момент завантаження сторінки, і вони можуть працювати не так, як треба.

Більшість розробників знають про важливі і необхідні HTTP-заголовки: Content-Type і Content-Length.

- Content-Type використовується для того, щоб визначити MIME тип ресурсу.
  `Content-Type: text/html; charset=utf-8`
  `Content-Type: multipart/form-data; boundary=something`
- Content-Length вказує розмір надісланого тіла об’єкта в байтах.
  `Content-Length: <довжина>`

MIME тип — код, який визначає формат файлу або тип контенту, що передається мережею Інтернет.

Тіло відповіді

Тіло відповіді містить певний файл. Наприклад, сервер може повернути код HTML-документа або надіслати JPEG-картинку.

Крок п'ятий – відображення браузером веб-сторінки

Як тільки браузер отримав відповідь із веб-сторінкою, він відображає її за допомогою внутрішнього движка. І на цьому весь процес надсилання та отримання HTTP-запитів закінчується, а клієнт отримує потрібні дані.

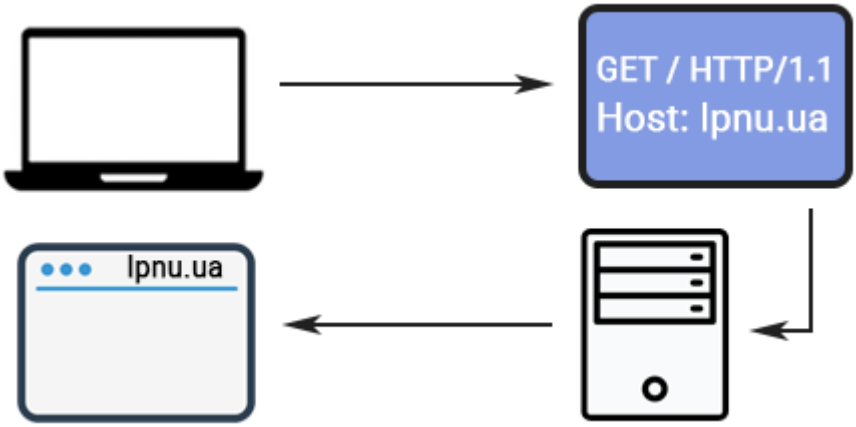

Рис.9. Надсилання HTTP-запиту — отримання HTML-сторінки

Отже:

- **HTTP-протокол** — це набір правил, якими комп'ютери обмінюються даними друг з одним. Його ініціює клієнт, а обробляє сервер та повертає назад клієнту. Між ними можуть бути проксі-сервери, які займаються додатковими завданнями - шифруванням даних, перерозподілом навантаження або кешуванням.
- **HTTP-запит** містить чотири елементи: метод, URI, версію HTTP та адресу хоста. Метод показує, яку дію необхідно вчинити. URI це шлях до конкретного файлу на сайті. Версію HTTP потрібно вказувати, щоб уникнути помилок, а адреса хоста допомагає браузеру визначити, куди надсилати HTTP-запит.
- **HTTP-відповідь** має три частини: статус відповіді, заголовки та тіло відповіді. У статусі відповіді повідомляється, чи все пройшло успішно або виникли помилки. У заголовках вказується додаткова інформація, яка допомагає браузеру коректно відобразити файл. А в тіло відповіді сервер кладе запитуваний файл.

З'єднання HTTP

Між клієнтом і сервером має бути встановлено з'єднання до того, як вони зможуть спілкуватися.

Для реалізації цього з'єднання HTTP використовує надійний протокол транспортного рівня TCP (Transmission Control Protocol), що гарантує доставку переданих пакетів даних у потрібній послідовності, але трафік при цьому може бути нерівномірним. За замовченням у мережі для даних використовується TCP-порт 80.

Поточна інформація, що передається через TCP-з'єднання, розділяється на IP-пакети (частина інформації, що передається через Інтернет; разом з даними містить адресу джерела й отримувача, а також поля, що визначають довжину дейтаграми, контрольну суму заголовка і прапорці, що вказують про фрагментації дейтаграми). Завдяки TCP-протоколу пакети завжди приймаються у правильному порядку.

Протокол TCP у свою чергу, працює на базі протоколу IP, що відповідає за передачу та маршрутизацію повідомлень між вузлами Інтернет.

Пригадати з курсу «Глобальні інформаційні мережі»

Захищений протокол HTTPS

HTTPS – це захищена версія HTTP, згідно з якою між HTTP та TCP додається додатковий шар – TLS (Transport Layer Security, Протокол захисту транспортного рівня) або SSL (Secure Sockets Layer, Протокол безпечних з'єднань). Для доступу до сторінок, захищених протоколом SSL, в URL замість звичайного префіксу http, як правило, застосовується префікс https (порт 443), що вказує на те, що буде використовуватися захищене з'єднання.

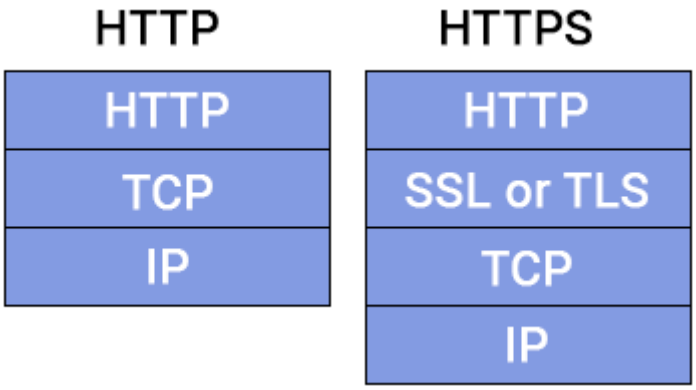

Рис.10. Рівні передачі даних через протоколи HTTP та HTTPS

Оскільки операції шифрування/дешифрування вимагають багато обчислювальних ресурсів, для зменшення навантаження на веб-сервери, використовують апаратні SSL-прискорювачі. Остання версія протоколу - SSL 3.0.

Встановлення з’єднання між двома вузлами мережі – процес, що складається з кількох етапів:

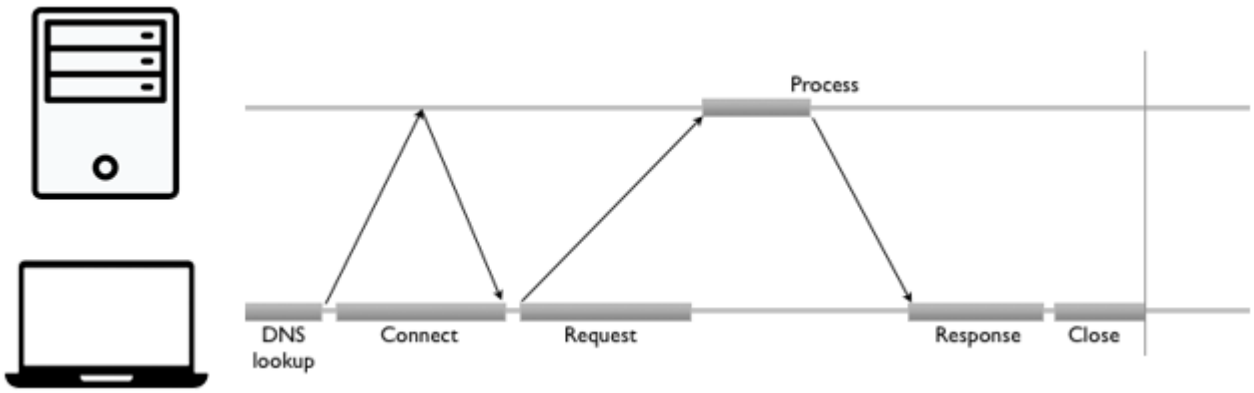

Рис.11. Процес з’єднання вузлів мережі

Встановлення з'єднання з сервером

- Надсилання запиту.
- Очікування відповіді.
- Отримання відповіді.
- Закриття з'єднання.

Сервер зобов'язаний завжди надсилати вірні заголовки та відповіді.

Відповідно до ранньої специфікації HTTP/1.0 всі з’єднання закривалися після здійснення єдиної транзакції, що складається із запиту.

Для зменшення кількості мережних запитів у HTTP/1.1 введено постійні з’єднання – довготермінові з’єднання, які залишаються відкритими, доки клієнт їх не закриє. Окрім постійних з’єднань, клієнти використовують технологію паралельних з’єднань – для зведення до мінімуму числа мережних запитів. Відповідно до цієї концепції створюється набір із кількох з’єднань.

Якщо необхідно завантажити шість ресурсів з веб-сайту, клієнт використовує шість паралельних з'єднань для їх завантаження, завдяки чому збільшується швидкість обслуговування користувачів сервером. Ця технологія є значним стрибком у порівнянні з послідовними з’єднаннями, при яких клієнт завантажує ресурс лише після отримання попереднього об’єкта.

Сьогодні за рахунок паралельних і постійних з'єднань мінімізується число мережних завантажень і підвищується якість взаємодії користувача з додатком.

Реалізація з'єднання на стороні сервера

Сервер зазвичай слухає з'єднання і приймає запити на обслуговування. Виконуються наступні дії:

- Відкривання каналу для початку прослуховування 80 порту (або будь-якого іншого).
- Отримання і розбір запитів.
- Формування відповіді.
- Додавання заголовків до відповіді.
- Надсилання відповіді до клієнта.
- Закривання з'єднань за наявності заголовка запиту `Connection: close`

Звісно, це не повний перелік дій, що виконує сервер. Більшості додатків необхідно знати, хто надсилає запит, щоб надіслати відповіді під конкретного користувача. І для цього потрібні ідентифікація та аутентифікація.

Ідентифікація

Серверу зазвичай необхідно бути в курсі, хто звертається до сервера, щоб перевірити відвідуваність сайту або моделі поведінки користувачів. Задача ідентифікації полягає в тому, щоб формувати відповідь під конкретного користувача. Зрозуміло, що для цього серверу необхідно надати інформацію про користувача.

Для її збору є кілька способів, і більшість веб-сайтів використовують їх комбінацію:

- Заголовки запиту: From, Referer, User-Agent.
- IP-адреса клієнта.
- Fat Urls (Extended Link, Розширене посилання) – інформація про поточну сесію користувача, що зберігається за рахунок змін URL-адрес. Стан сесії накопичується при кожному переході користувача за URL-адресою.
- Куки - текстовий запис розміром до 4 Кбайт з даними про користувача, що повертається веб-сервером при реєстрації користувача і зберігається на його комп’ютері. Зазвичай, від користувача потрібна попередня згода на використання куки-файлів. Самі куки

можна поділити на категорії відповідно до їх нав'язливості (ступень звернення до персональної інформації): від нульової до високої.

Найкращий спосіб ідентифікації користувача – зобов'язати його або увійти в додаток, однак для реалізації цієї можливості необхідно докласти зусиль як розробнику, так і користувачеві.

На сьогодні популярним є використання відкритого протоколу надання авторизації Open Authorization, який дозволяє надати третій стороні обмежений доступ до захищених ресурсів користувача без необхідності передавати її (третій стороні) логін і пароль.

Використання технологій, подібних до OAuth покращує реалізацію ідентифікації, однак також потрібна згода користувача.

Аутентифікація

Аутентифікація - службова функція системи контролю доступу, що забезпечує збіг введених пароля і логіна користувача з записом, який є у базі даних додатку. В системі комп'ютерної безпеки - це процес, який дозволяє встановити, що користувач або комп'ютер (сервер), намагаючись отримати інтерактивний доступ до визначеної категорії інформації, комп'ютерної системи, обчислювальної мережі або електронної пошти, дійсно той, за кого себе видає.

У самому протоколі HTTP реалізовано підтримку елементарної форми аутентифікації – Basic Authentication та більш захищеної – Digest Authentication.

При базовій аутентифікації сервер спочатку відхиляє запит клієнта і надсилає відповідь із заголовком відповіді WWW-Authenticate та кодом стану 401 Unauthorized. Після отримання такого заголовку, браузер відкриває вікно для входу на сайт, запитує пароль та ім'я користувача.

Ця інформація надсилається в закодованому вигляді за допомогою коду Base-64 в заголовку запиту Authentication. Після цього сервер перевірить правильність даних у запиті та у разі коректності даних надасть доступ. Деякі сервери також можуть надсилати заголовок Authentication-Info з додатковою інформацією про аутентифікацію.

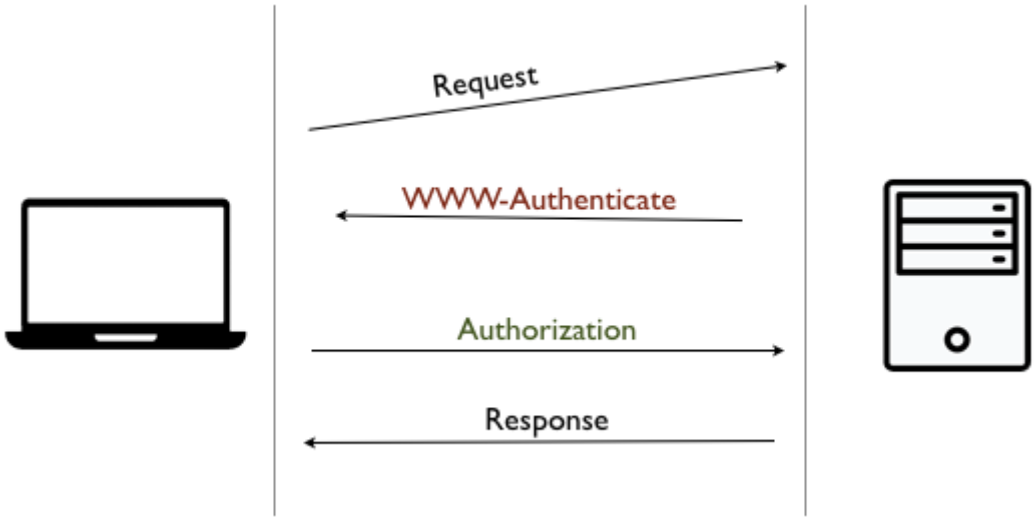

Рис. 12. Процес базової аутентифікації користувача

Для підвищення безпеки в парі з Basic Auth використовується протокол SSL.

HTTPS

За рахунок протоколу HTTPS в мережі реалізуються захищені з'єднання. Простий спосіб запевнитися у використанні HTTPS – перевірити адресний рядок браузера.

Захищена передача в HTTPS забезпечується за рахунок додавання шарів шифрування/дешифрування між HTTP і TCP. Це є протокол SSL або його вдосконалена версія – TLS.

У SSL використовується надійна форма шифрування за допомогою RSA. RSA-кодування (шифрування) – схема (алгоритм) асиметричного шифрування (Asymmetric Cipher) з відкритими ключами. Використовує пару ключів, причому кожна пара має наступні властивості: що-небудь зашифроване одним з них (ключ для шифрування називається відкритим і доступний всім) може бути розшифровано за допомогою іншого (цей ключ називається закритим, або секретним, ключем).

Існуючим клієнтам/серверам немає потреби змінювати свій спосіб обробки повідомлень, оскільки основна частина складної роботи виконується на рівні SSL. Тому, можна розробляти веб-додаток за допомогою Basic Authentication і автоматично скористатися перевагами SSL шляхом перемикання на протокол https://.

Однак для того, щоб веб-додаток працював за протоколом HTTPS, на сервері повинен бути дійсний цифровий сертифікат.

Сертифікат

Для ідентифікації веб-сервера використовується цифровий сертифікат, який унікальним чином ідентифікує сайт, показує, що можна довіряти цьому ресурсу і забезпечує безпечну конфіденційну інформацію.

Сертифікати випускаються у центрах сертифікації (Certificate Authority) (https://www.ssls.com/ https://ssl.com.ua/), в них міститься наступна інформація:

- Видавець сертифікату.
- Алгоритм, що використано для створення сертифіката.
- Ім'я суб'єкта (окремий користувач або організація), для якого створено цей сертифікат.
- Інформація про публічний ключ суб'єкта.
- Підпис центра сертифікації, виконаний за допомогою заданого алгоритму створення підпису.

При виконанні запиту клієнта, HTTPS спочатку намагається отримати сертифікат від сервера. У разі успіху клієнт перевіряє його з центрами сертифікації, що знаходяться в його списку. Якщо видавець сертифіката не відноситься до будь-якого з них, клієнт може показати користувачеві діалогове вікно з повідомленням про відсутність сертифіката веб-сайту.

Як тільки довжина сертифіката встановлена, SSL-рукостискання завершено і починається безпечна передача даних.

Типи SSL/TLS-сертифікатів та хто їх видає

Сертифікати бувають різні: групові, багатодоменні, що підтверджують організацію. Нижче наведено перелік популярних типів сертифікатів, розташувавши їх від небезпечних до безпечних.

- **Самопідписані**. Такий сертифікат можна згенерувати на власному сервері та самому підписати. Самопідписаний сертифікат забезпечує такий самий рівень безпеки в частині криптографії, але має менший рівень довіри, доки видавець не долучиться до списку довірених видавців. Іноді самопідписані сертифікати використовують для тестування процесу встановлення сертифіката.
- **Групові сертифікати.** На CloudFlare (https://www.cloudflare.com/) чи Let's Encrypt (https://letsencrypt.org/uk/) видають безкоштовні сертифікати. Але їх треба продовжувати кілька разів на рік. Такі сертифікати видаються на групу сайтів. Wildcard-сертифікати видаються на доменне ім'я рівнем вище, ніж передбачається використовувати: тобто сертифікат на lpnu.ua буде валідним для всіх доменних імен нижчого рівня (eom.lpnu.ua). Для деяких компаній такий сертифікат – можливість закрити велику кількість доменів одним сертифікатом.
- **Підписані довіреним центром сертифікації.** Такі сертифікати видають центри сертифікації або центри, що засвідчують сертифікати. Краще вибирати великі та надійні центри, до яких є довіра у браузерів. Вони підтверджують справжність ключів шифрування своїм електронним підписом.

Сертифікати за рівнем перевірки та організації в HTTPS

- **Domain Validation** – сертифікат, який підтверджує домен. Центр, що засвідчує сертифікати перевіряє право власності на домен, зазвичай через електронну пошту. Підійдуть для простих сайтів — наприклад, персональних блогів, сайтів новин та інших, на яких не збираються персональні дані та не проводяться платежі.
- **Organization Validation** – сертифікат, який підтверджує організацію. Тут Центр, що засвідчує сертифікати перевіряє, чи існує організація юридично: ідентифікує власника домену, його особу, просить надати документи. Підійдуть для магазинів та інших ресурсів, де можна проводити платежі.
- **Extended Validation** – сертифікати з розширеною перевіркою. Сайти з таким сертифікатом мають зелений замочок та видають всю інформацію про ресурс при натисканні на нього.

Кешування

Оскільки багато дій користувача здійснюється через мережу, то кеш сприяє збереженню часу, зменшенню витрат і підвищенню пропускної здатності, а також зменшенню часу реакцій на запит користувача.

Робота з кешем відбувається на декількох рівнях мережної інфраструктури: починаючи від браузера і закінчуючи початковим сервером. У залежності від того, де розміщений кеш, можна виділити його категорії:

- **Приватний**: всередині браузера. Тут кешуються паролі, імена користувачів, URL-адреси, історія відвідувань і веб-контент. Цей тип кеш-пам'яті зазвичай невеликого об'єму і призначений для одного користувача.
- **Публічні**: розміщуються на проксі-серверах для кешування між сервером і клієнтом. Об'єм пам'яті цього типу кеш-пам'яті значно більше, оскільки тут обслуговується багато користувачів. Відповідно до існуючої практики між клієнтом і початковим сервером слід розмістити багато проксі-серверів для кешування. Завдяки цьому забезпечується

швидке обслуговування контенту, до якого часто звертаються, і в той же час зберігається можливість отримати контент, який запитують значно менше.

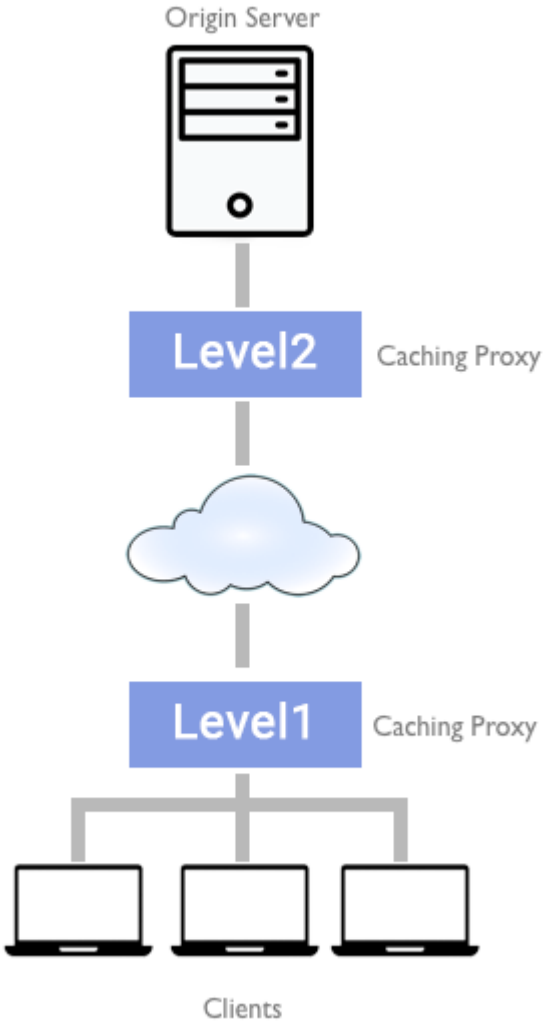

Рис.13. Розміщення кешу на кількох рівнях мережної інфраструктури

Незалежно від того, де розміщений кеш, процес підтримки кеш-пам'яті досить схожий:

- Отримання повідомлення запиту.
- Розбір URL-адрес і заголовків.
- Пошук локальної копії, у разі відсутності надсилається запит на отримання ресурсу і збереження його локально.
- Перевірка актуальності контенту в кеш-пам'яті. Запит для оновлення контенту виконується лише за необхідності.
- Формування відповіді з кеша, що складається з тіла повідомлення та оновлених заголовків.
- Надсилання відповіді до клієнта.
- Фіксація подій в журналі реєстрації подій (Необов'язково).

Сервер зобов'язаний завжди надсилати вірні заголовки та відповіді. Якщо документ не змінився, сервер повинен надіслати відповідь із кодом стану 304 Not Modified. Якщо термін дійсності копіювання з кеша вичерпано, сервер повинен створити нову відповідь з оновленими заголовками відповіді та 200 OK. У випадку, якщо ресурс видалено, має бути надіслано відповідь з 404 Not Found. Завдяки цим відповідям передбачається налагодження кеша та гарантується, що старий контент не буде зберігатися занадто довго.

Підтримка контенту в актуальному стані – це одна з основних задач кеша. Для підтримки копії ресурсу в кеш-пам’яті відповідно до імені на сервері, HTTP надає деякі прості механізми, а саме: використання терміну дійсності документа (Термін дії документа) і повторна перевірка ресурсу на сервері (Повторна перевірка сервера).

Використання терміну дійсності документа

HTTP надає можливість первинному серверу додати дату завершення терміну до кожного документа за допомогою заголовків відповіді Cache-Control та Expires. Завдяки цьому забезпечується контроль клієнта та інших серверів для кешування актуальності контенту. Копія з кеша за запитами буде надсилатися доки не завершився термін дійсності документу. Якщо ця умова не виконується, кеш повинен звірятися з сервером на наявність оновленої версії документа та оновлювати відповідно свою локальну копію.

- Expires – це заголовок відповіді HTTP/1.0, в якому задається абсолютне значення дати. Буде корисним в тому випадку, якщо час на сервері відповідає часу клієнта, що дуже малоймовірно.
- Cache-Control: max-age=<s>, це заголовок відповіді HTTP/1.1. Тут max-age задається відносний термін, вказаний у секундах, з моменту створення відповіді. Таким чином, якщо термін документа повинен вичерпатися по завершенні одного дня з моменту створення, необхідно використовувати наступний запис: Cache-Control: max-age=86400

Перевірка ресурсу на сервері

Як тільки термін дійсності документа з кеша завершується, він повинен бути перевірений із сервером для з’ясування, чи був змінений документ. Цей процес є способом перевірки актуальності документа. Він необхідний, оскільки вичерпання терміну збереження в кеш-пам'яті ще не означає, що на сервері з'явилася нова версія ресурсу. Така перевірка гарантує актуальність контенту в кеш-пам'яті. Якщо задано час завершення терміну (у попередній відповіді сервера), кеш не потрібно звіряти з сервером при кожному запиті, завдяки чому підвищується пропускна здатність, зменшується час і мережний трафік.

Комбінація перевірки ресурсу на сервері та використання терміну дійсності документа – ефективний механізм підвищення якості використання мережі. Завдяки ньому стає можливим підтримання в розподілених системах актуальності копій ресурсів на основі дати завершення терміну.

Якщо відомо, що контент часто змінюється, то значення дати терміну завершення можна зменшити, надавати можливість системам повторно синхронізуватися частіше.

Етап перевірки може бути виконаний за допомогою двох видів заголовків запиту, де використовуються значення дати або ETag, що отримані в попередній відповіді сервера.

- If-Modified-Since використовується для здійснення перевірок на підставі дати. З ним використовується заголовок відповіді Last-Modified.
- If-None-Match. Разом з ним використовується заголовок Entity-Tag (ETag) – хэш-значення вмісту.

Контроль можливості кешування

Термін дійсності документа повинен визначатися сервером, що надсилає документ. У разі веб-сайту з новинами термін головної сторінки повинен закінчуватися по завершенні одного дня (або іноді навіть кожної години). HTTP надає заголовки відповіді Cache-Control та Expires для встановлення дати закінчення терміну документів.

Заголовок Expires не є надійним рішенням для контролю кеша. Заголовок Cache-Control значно корисніше, і в ньому можуть бути використано кілька різних значень для визначення способу кешування відповіді клієнта:

- Cache-Control: no-cache: клієнту дозволено зберегти документ; однак він повинен виконувати перевірку актуальності контенту з сервером при кожному запиті.
- Cache-Control: no-store: директива до клієнта взагалі не зберігати документ.
- Cache-Control: must-revalidate: клієнт повинен пропустити визначення актуальності контенту та завжди виконувати перевірку ресурсу на сервері. Не дозволяється використовувати відповідь із кеша у випадку, якщо сервер недоступний.
- Cache-Control: max-age : тут задається відносний вік (у секундах) з моменту створення відповіді.

У випадку, якщо сервер не надсилає заголовок Cache-Control з будь-яким значенням, клієнт використовує власний евристичний алгоритм для визначення актуальності контенту.

Обмеження актуальності ресурсу зі сторони клієнта

Можливість кешування контенту визначається не лише сервером. Клієнт також може в цьому брати участь і накласти обмеження на відповідь. Ця можливість реалізується за допомогою заголовку Cache-Control, хоча і з деякими іншими значеннями:

- Cache-Control: min-fresh=<s>: документ має бути дійсним як мінімум <s> секунд.
- Cache-Control: max-stale або Cache-Control: max-stale=<s>: документ не може бути надіслано з кеша, якщо його буде видалено більше, ніж на <s> секунді.
- Cache-Control: max-age=<s>: документ не може бути надісланий з кеша, якщо він знаходився там більше <s> секунди.
- Cache-Control: no-cache: клієнт не буде приймати ресурс із кеш-пам’яті до цього порту, доки не буде виконана його перевірка на сервері.

Кешування в HTTP – це дуже цікава тема, що містить складні алгоритми для управління контентом в кеш.

Практична перевірка HTTP заголовків

Розглянемо деякі способи, як переглянути HTTP-заголовки сторінки або окремого файлу.

Перегляд HTTP-заголовків у браузері Google Chrome

HTTP Headers у Chrome можна знайти в інструментах розробника. Для цього необхідно натиснути або Ctrl+Shift+I (багато хто використовує просто F12), або правою кнопкою миші і вибрати пункт «Подивитися код», або у верхньому меню браузера вибрати «Інші інструменти» → «Інструменти розробника».

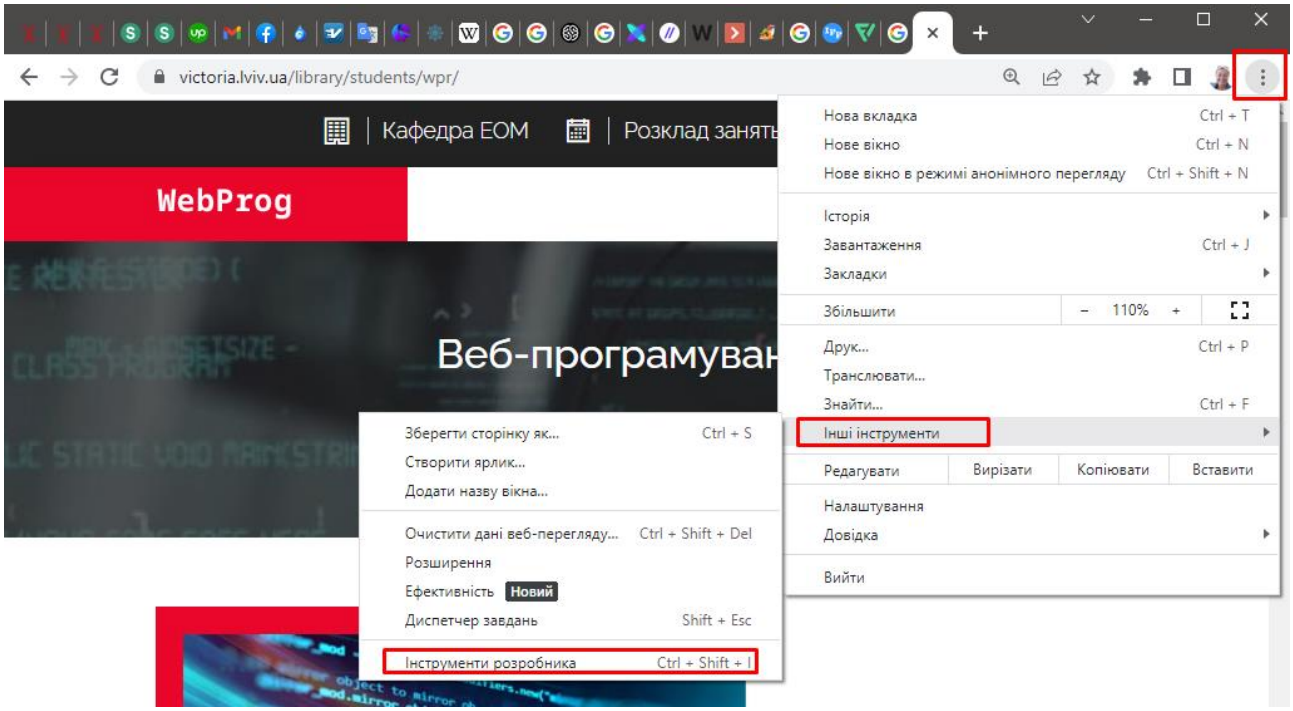

Рис.14. Вибір меню «Інструменти розробника»

Після цього вибрати вкладку «Network» та оновити сторінку (F5).

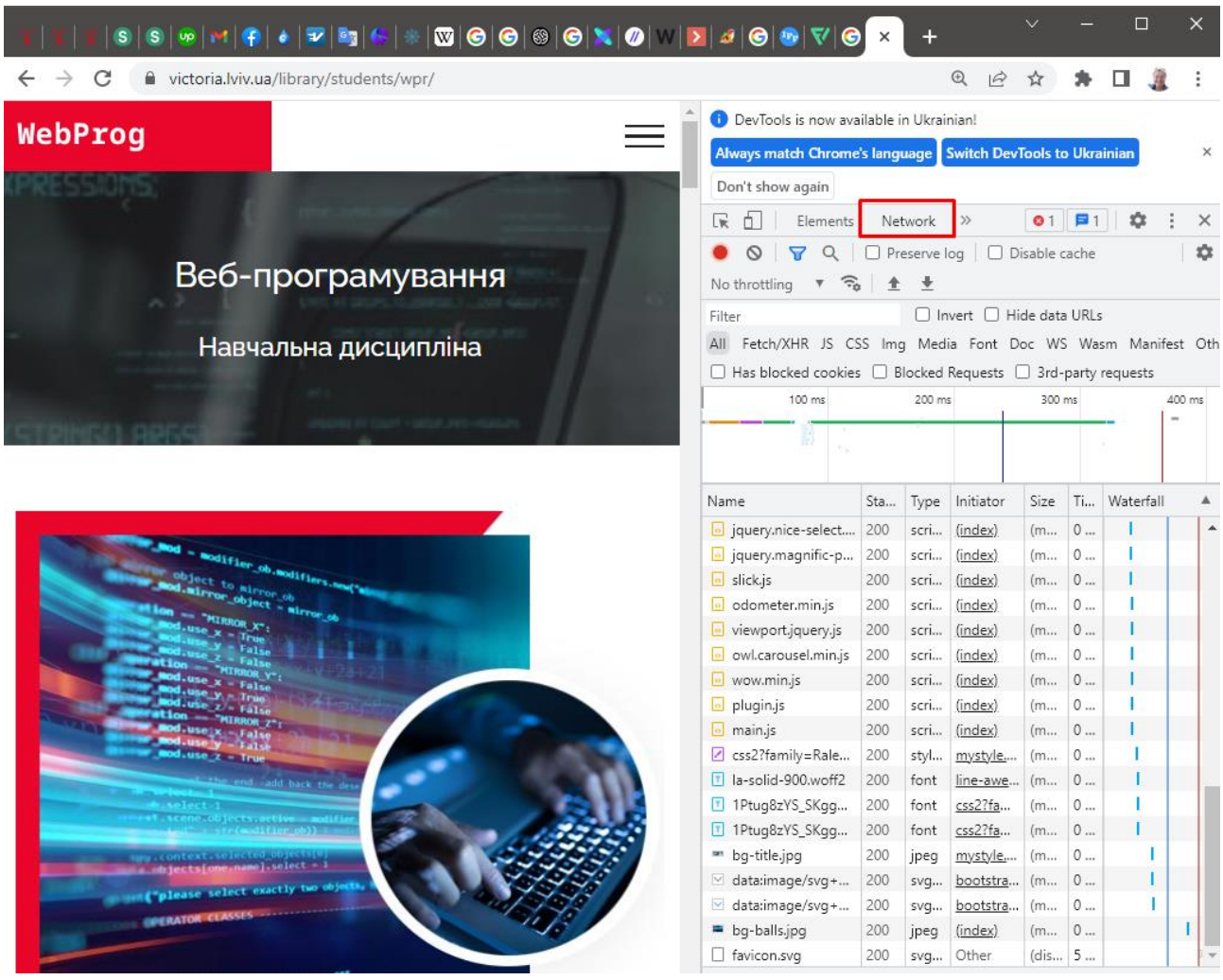

Рис.15. Панель властивостей веб-ресурсів у браузері Chrome

У графі "Name" необхідно вибрати тип файлу, для якого ви хочете перевірити заголовки, і праворуч у вкладці Headers будуть вказані всі заголовки поточного файлу.

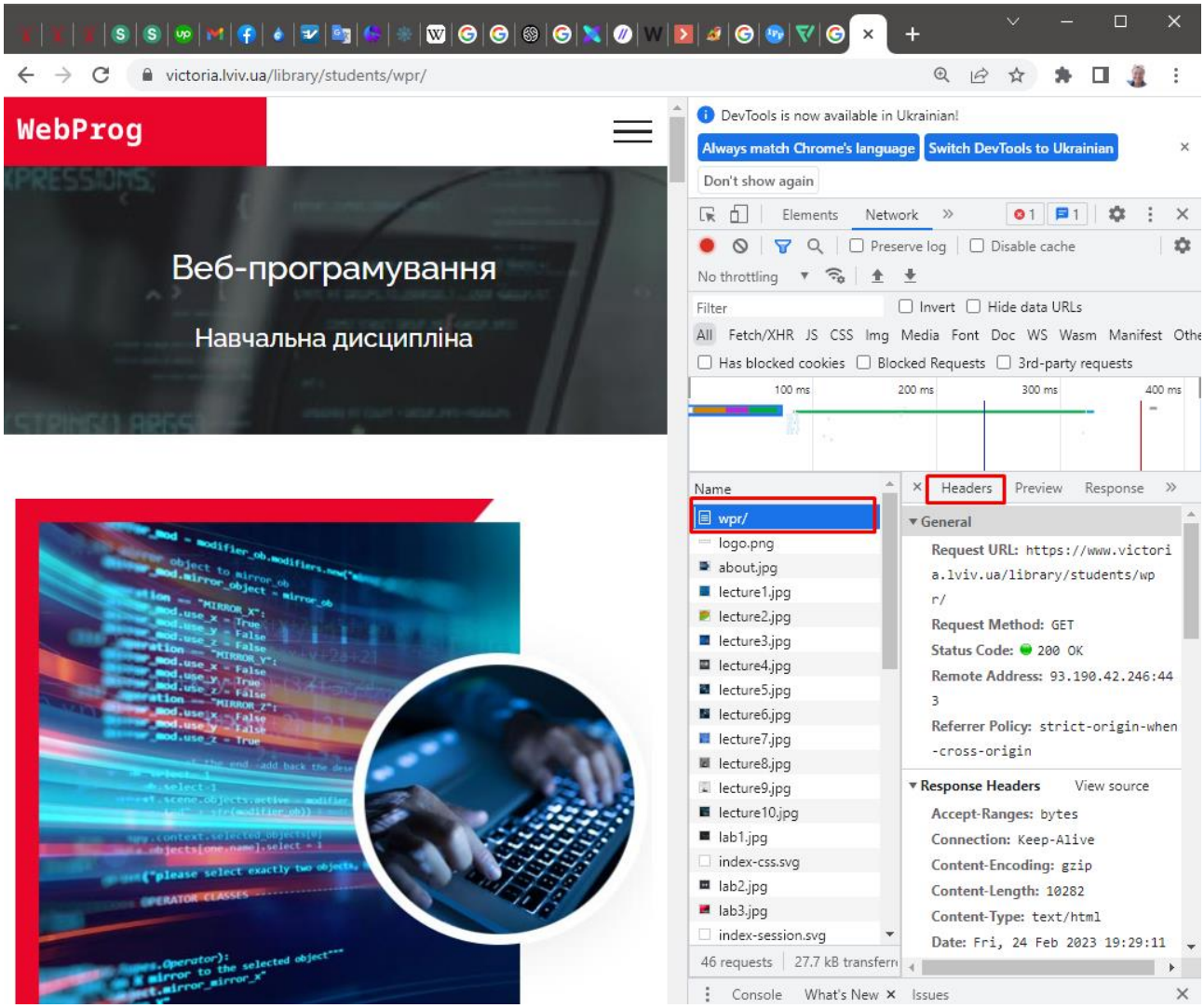

Рис.16. Перегляд заголовків відповідей сервера у браузері Chrome

Перегляд HTTP-заголовків у браузері Firefox

Аналогічним способом можна перевірити заголовки і в Firefox: за допомогою Ctrl+Shift+C або у верхньому випадному меню вибрати «Веб-розробка» → «Інструменти розробника». Далі вибрати вкладку «Мережа» та оновити сторінку (F5). Після цього вибрати тип документа для перевірки та у правій частині екрана вибрати вкладку «Заголовки». Перед вами з'являться заголовки поточної сторінки.

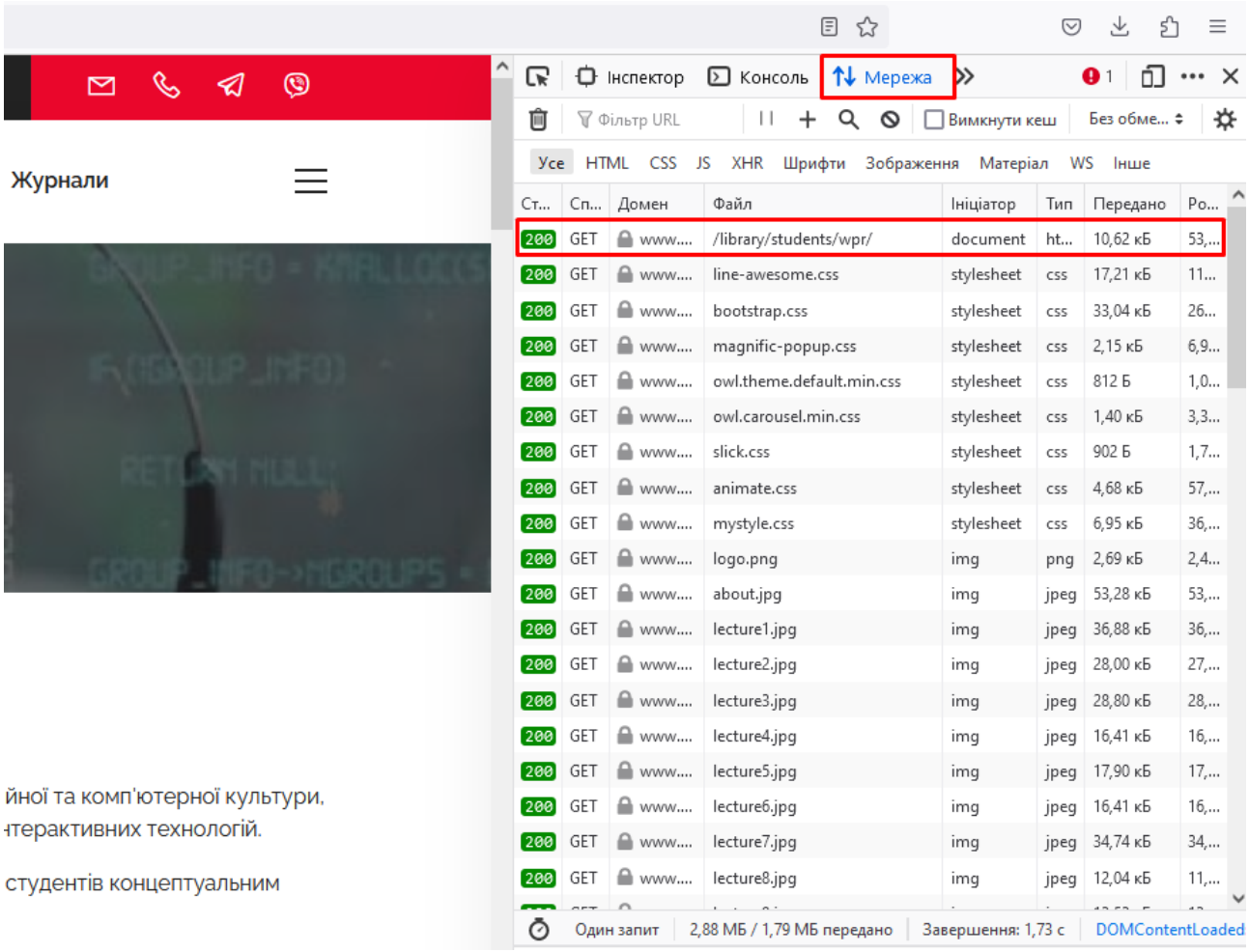

Рис.17. Перегляд заголовків відповідей сервера у браузері Mozilla

Інші способи перевірки HTTP-заголовків

Для того щоб подивитися HTTP-заголовки є багато розширень для будь-якого браузера Google Chrome, Mozilla Firefox або Internet Explorer.

Джерела інформації

1. HTTP-запити: структура, методи, рядок статусу и коди стану https://selectel.ru/blog/http-request/
2. В чому різниця протоколів HTTP и HTTPS https://selectel.ru/blog/http-https/
3. Протокол HTTP: принцип дії https://selectel.ru/blog/http-https/