hillel IT school

# Lesson 24

# План заняття

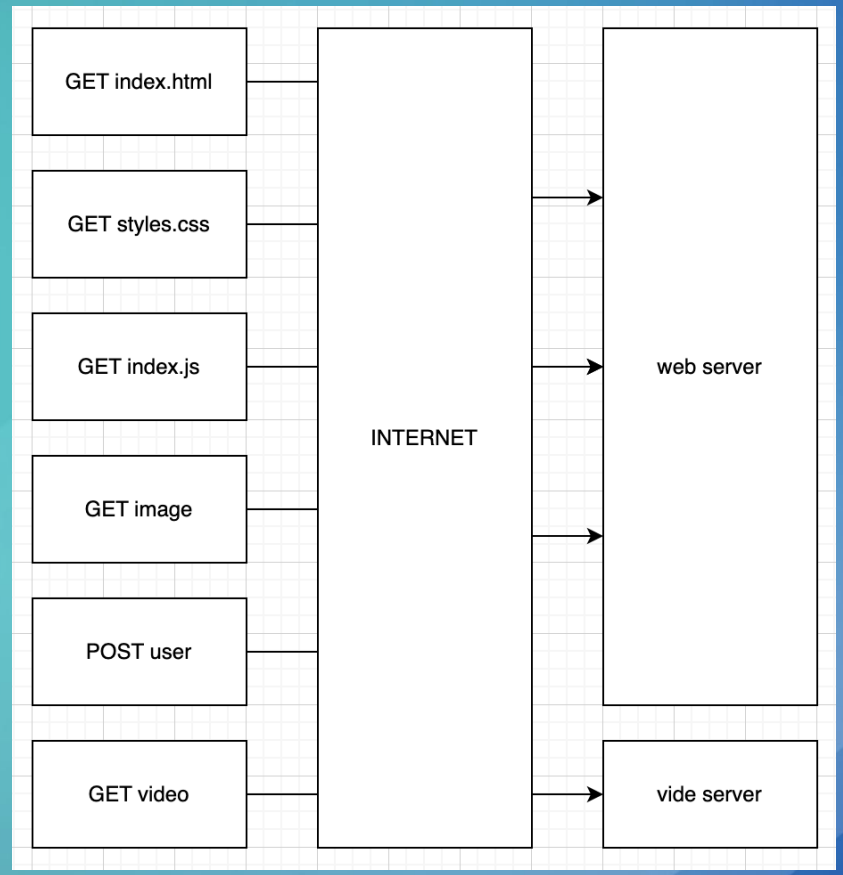
- HTTP;
- Приклади відкритих APIs;
- знайомство з Ajax;
- Fetch;
- CORS;

# Http

HTTP – це протокол, який дозволяє отримувати різні ресурси, наприклад HTML-документи. Протокол HTTP є основою обміну даними в Інтернеті. HTTP є протоколом клієнт-серверної взаємодії, що означає ініціювання запитів до сервера самим одержувачем, як правило, веб-браузером (web-browser). Отриманий підсумковий документ (може) складатися з різних піддокументів, що є частиною підсумкового документа: наприклад, з окремо отриманого тексту, описи структури документа, зображень, відео-файлів, скриптів та багато іншого.

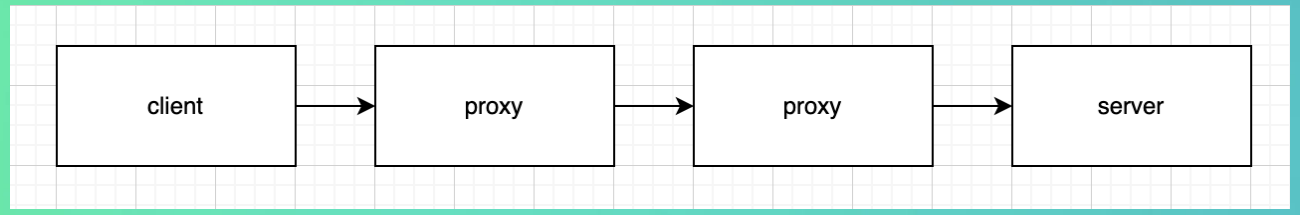
Повідомлення, надіслані клієнтом, зазвичай веб-браузером, називаються запитами, а повідомлення, надіслані сервером, називаються відповідями.

HTTP є протоколом прикладного рівня, який найчастіше використовує можливості іншого протоколу – TCP (або TLS – захищений TCP) – для пересилання своїх повідомлень, проте будь-який інший надійний транспортний протокол теоретично може бути використаний для доставки таких повідомлень.



# Складові системи, засновані на HTTP

Кожен запит (англ. request) відправляється серверу, який обробляє його та повертає відповідь (англ. response). Між цими запитами та відповідями зазвичай існують численні посередники, звані проксі, які виконують різні операції та працюють як шлюзи або кеш, наприклад.




У цій системі рівнів HTTP займає найвищий рівень, який називається "прикладним" (або "рівнем додатків").

# User agent

Учасник обміну (user agent) – це будь-який інструмент або пристрій, що діють від імені користувача. Це завдання переважно виконує веб-браузер. Браузер завжди є тією сутністю, що створює запит. Браузер завжди є тією сутністю, яка створює запит.

Щоб відобразити веб-сторінку, браузер надсилає початковий запит для отримання HTML-документа цієї сторінки. Після цього браузер вивчає цей документ і запитує додаткові файли, необхідні для відображення змісту веб-сторінки (скрипти, інформацію про макет сторінки - CSS таблиці стилів, додаткові ресурси у вигляді зображень і відео-файлів), які безпосередньо є частиною вихідного документа, але розташовані в інших місцях мережі. Далі браузер з'єднує всі ці ресурси для відображення їх користувачу як єдиного документа — веб-сторінки. Скрипти, що виконуються самим браузером, можуть отримувати додаткові ресурси по мережі на наступних етапах обробки веб-сторінки, і браузер відповідним чином оновлює відображення цієї сторінки для користувача.

# Web server

З точки зору кінцевого користувача, сервер завжди є якоюсь однією віртуальною машиною, що повністю або частково генерує документ, хоча фактично він може бути групою серверів, між якими балансується навантаження, тобто перерозподіляються запити різних користувачів, або складним програмним забезпеченням, що опитує інші комп'ютери (такі як кешують сервери, сервери баз даних, сервери додатків електронної комерції та інші).

# Proxy

Між веб-браузером та сервером знаходяться велика кількість мережних вузлів, що надсилають повідомлення HTTP. Через шарувату структуру більшість з них оперують також на транспортному мережевому або фізичному рівнях, стаючи прозорим на шарі HTTP і потенційно знижуючи продуктивність. Ці операції лише на рівні додатків називаються проксі. Вони можуть бути прозорими чи ні, (зміни запитів не пройдуть через них), і здатні виконувати безліч функцій:

- caching (кеш може бути публічним або приватним, як кеш браузера)
- фільтрація (як сканування антивірусу, батьківський контроль, …)
- вирівнювання навантаження (дозволити кільком серверам обслуговувати різні запити)
- автентифікація (контролювати доступом до різних ресурсів)
- протоколювання (дозвіл на зберігання історії операцій)

# HTTP flow

Коли клієнт хоче взаємодіяти з сервером, що є кінцевим сервером або проміжним проксі, він виконує такі кроки:

- Відкриття TCP-з'єднання: TCP-з'єднання буде використовуватися для надсилання запиту (або запитів) та отримання відповіді. Клієнт може відкрити нове з'єднання, перевикористовувати існуюче або відкрити кілька з'єднань TCP до сервера.
- Надсилання HTTP-повідомлення: HTTP-повідомлення (до HTTP/2) є людиночитаними. Починаючи з HTTP/2, прості повідомлення інкапсулюються у кадри, унеможливлюючи їх читання безпосередньо, але принципово залишаються такими ж. Наприклад:

```
GET / HTTP/1.1
Host: example.org
Accept-Language: en
```

# HTTP flow

- Читає відповідь від сервера:

HTTP/1.1 200 OK
Date: Sat, 09 Oct 2024 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
<!DOCTYPE html... (here comes the 29769 bytes of the requested web page)

- Закриває або перевикористовує з'єднання для подальших запитів.

# Request

Запити містять такі елементи:

- HTTP-метод, зазвичай дієслово подібно до GET, POST або іменник, як OPTIONS або HEAD, що визначає операцію, яку клієнт хоче виконати. Зазвичай клієнт хоче отримати ресурс (використовуючи GET) або передати значення HTML-форми (використовуючи POST), хоча інші операції можуть бути необхідні в інших випадках.
- Шлях до ресурсу: URL ресурси позбавлені елементів, які є очевидними з контексту, наприклад без протоколу (http://), домену (тут developer.mozilla.org), або TCP порту (тут 80).
- Версію HTTP-протоколу.
- Заголовки (опціонально), які надають додаткову інформацію для сервера.
- Або тіло для деяких методів, таких як POST, що містить відправлений ресурс.

# Response

Відповіді містять такі елементи:

- Версію HTTP-протоколу.
- HTTP код стану, що повідомляє про успішність запиту або причину невдачі.
- Повідомлення стану – короткий опис коду стану.
- HTTP заголовки, подібно до заголовків у запитах.
- Опціонально: тіло, що містить ресурс, що пересилається.

Більш детально про HTTP можна прочитати тут.

# Houston i have a problem!

Проблема з моделлю request => response полягає в тому, що кожен раз, коли ви хочете оновити будь-яку частину сторінки, наприклад, щоб відобразити новий набір продуктів або завантажити нову сторінку, вам потрібно знову завантажити всю сторінку.

# Ajax

Ajax - технологія, що дозволяє веб-сторінкам вимагати невеликі фрагменти даних (наприклад, HTML, XML, JSON або звичайний текст) і відображати їх лише за необхідності. Є загальний термін “AJAX” (абревіатура від Asynchronous JavaScript And XML) для мережевих запитів від JavaScript коду. Але формат XML використовувати не обов’язково: цей термін застарілий, тому це слово (XML) тут.

Модель Ajax передбачає використання веб-API як проксі для більш розумного запиту даних, а не просто для того, щоб браузер перезавантажував всю сторінку.

# fetch

Синтаксис:

const promise = fetch(url, [options])

1. url – URL для відправлення запиту.
2. options – додаткові параметри: метод, заголовки і т.д.

Без options, це просто GET запит, який завантажує зміст за адресою url.

Процес отримання запиту зазвичай відбувається у два етапи.

1. Promise завершиться із об'єктом вбудованого класу Response у якості результату, одразу коли сервер надішле заголовки відповіді. Проміс закінчується помилкою, якщо fetch не зміг виконати HTTP-запит, наприклад, через помилку мережі або, якщо такого сайту не існує. 404 та 500, не викликатимуть помилку.
2. Для отримання тіла запиту, потрібно використовувати додатковий виклик методу.

# fetch

Response надає декілька методів, які повертають проміс, для доступу до тіла запиту в різних форматах:

- response.text() – читає відповід та повертає, як звичайний текст;
- response.json() – декодує відповідь у форматі JSON;
- response.formData() – повертає відповідь, як об’єкт FormData;
- response.blob() – повертає відповідь, як Blob (бінарні дані з типом);
- response.arrayBuffer() – повертає відповідь, як [ArrayBuffer](інформація: буфер масиву – бінарний масиви) (низькорівневе представлення двійкових даних);
- крім того, response.body це об’єкт ReadableStream, за допомогою якого можна отримувати (зчитувати) тіло відповіді частинами;

# headers

Заголовки відповіді зберігаються у схожому на Map об’єкті response.headers.

Для встановлення заголовка запиту в fetch, можна використати властивість headers в об’єкті options.

Список заборонених заголовків можна знайти тут. Ці заголовки забезпечують достовірність HTTP, через це вони контролюються і встановлюються лише браузером.

# POST

Для відправлення POST запиту або запиту з іншим методом, треба використати fetch параметри:

- method – HTTP-метод, наприклад POST,
- body – тіло запиту, щось одне із списку:

рядок (наприклад, у форматі JSON),

об’єкт FormData, для відправки даних як multipart/form-data,

Blob/BufferSource для відправлення бінарних даних,

URLSearchParams, для відправлення даних у кодуванні x-www-form-urlencoded, використовуєся рідко.

Частіше використовуєся JSON формат.

Можна відправити бінарні дані за допомогою fetch, використовуючи об’єкт Blob або BufferSource.

# Методи об’єкта FormData

Можемо змінювати поля в об’єкті FormData за допомогою:

- formData.append(name, value) – додає до об’єкта поле з іменем name і значенням value,
- formData.append(name, blob, fileName) – додає поле так, ніби це <input type="file">, третій аргумент fileName встановлює ім’я файлу (не ім’я поля форми), ніби це ім’я з файлової системи користувача,
- formData.delete(name) – видаляє поле по заданому name,
- formData.get(name) – дістає значення поля по заданому name,
- formData.has(name) – перевіряє чи існує поле по заданому name, повертає true, інакше false

Ще існує метод set, його синтаксис такий самий, як у append. Різниця в тому, що .set видаляє всі наявні поля з ім’ям name і тільки потім додає нове. Тобто цей метод гарантує, що існуватиме лише одне поле з ім’ям name, у всьому іншому він аналогічний .append:

- formData.set(name, value),
- formData.set(name, blob, fileName).

# Надсилання форми з файлом

Об’єкти FormData завжди посилаються із заголовком Content-Type: multipart/form-data, цей спосіб кодування дозволяє надсилати файли. Таким чином, поля <input type="file"> теж відправляються, як і при використанні разі звичайної форми.

# Хід завантаження

Метод fetch дозволяє відстежувати хід завантаження, але fetch не може відстежувати хід вивантаження.

Щоб відстежувати хід завантаження, ми можемо використовувати властивість response.body. Це ReadableStream – спеціальний об’єкт, який надає тіло відповіді фрагментами, в міру надходження.

Результатом виклику await reader.read() є об’єкт з двома властивостями:

done – true, коли зчитування завершено, інакше – false.

value – типізований масив байтів: Uint8Array.

# Переривання fetch

Для таких цілей є спеціальний вбудований об’єкт: AbortController. Його можна використовувати для переривання не тільки fetch, але й інших асинхронних завдань.

Створімо контролер:

```
let controller = new AbortController();
```

# Переривання fetch

Контролер – це об’єкт.

- Він має єдиний метод abort(),
- І єдину властивість signal, що дозволяє встановлювати на ньому обробники подій.

Коли викликається abort():

- controller.signal генерує подію "abort".
- Властивість controller.signal.aborted стає true.

Як правило, у нас є дві сторони в процесі:

- Та, що виконує операцію, яку можна скасувати, встановлює прослуховувач на controller.signal.
- Та, що скасовує: вона викликає controller.abort(), коли потрібно.

# Дякую за увагу