DOI: 10.59647/978-617-520-936-3/1

Використання Fetch API для мережевих запитів

Максим Жураковський, студент[1] (ORCID: 0009-0004-5024-8685)**. Сергій Позднякович,** студент[1] (ORCID: 0009-0006-7765-8339)**. Тетяна Гончаренко,** д.т.н., доцент, завідувач кафедри ІТ [1] (ORCID: 0000-0003-2577-6916)

[1] *Київський національний університет будівництва і архітектури, 03037, м. Київ, проспект Повітряних Сил, 31, Україна*

АНОТАЦІЯ

Зроблено введення до поняття API. Представлено результат дослідження можливостей використання Fetch API для мережевих запитів та означені місця його використання при розробці різноманітних додатків. Розписана робота об'єктів promises, request та responce. Перераховані переваги та недоліки інтерфейсу та показано, що він є сучасним і конкурентоспроміжним.

Ключові слова: API, Fetch, мережа, браузер, запит, Інтернет, jQuery, Ajax.

1. ВСТУП

При створені сучасних сайтів розробник не може обійтися без використання сторонніх ресурсів, будь то графічний інтерфейс, зв'язок із соціальними мережами, дані про користувачів або оновлення серверів. Для їх під'єднання використовуються так звані API, які полегшують створення складної функціональності.

Fetch відноситься до приймаючого дані від серверів типу інтерфейсів, на рівні із XMLHttpRequest (XHR) для Ajax, та добре підтримується сучасними браузерами, що робить його актуальним, корисним та зручним інструментом розробки.

2. МЕТА РОБОТИ

Метою даної роботи є дослідження можливостей обробки мережевих запитів за допомогою Fetch API. Для досягнення мети доцільно здійснити опис переваг та недоліків цього інтерфейсу в порівнянні з іншими існуючими інформаційними технологіями та продемонструвати на прикладах можливості його практичного використання.

3. ДОСЛІДЖЕННЯ

3.1. API

Інтерфейси прикладного програмування (Application Programming Interfaces) – готові конструкції мови програмування, які задають правила для зв'язку різних програмних систем. API забезпечують більш легкий доступ для їх використання, залишаючи складний код поза межею навантажень розробника. [1]

В якості аналогії можна привести процес заправки автомобіля – замість того, щоб переносити та заливати бензин, використовуючи відра, власнику машини достатньо вставити шланг у відповідний отвір.

3.2. Fetch

Fetch API – інтерфейс JavaScript для вилучення ресурсів у мережі. Способи виконання HTTP запитів за його допомогою є більш гнучкими та ефективними, ніж старий XMLHttpRequest.

Для обробки відповідей використовуються promises – об'єкти, які надають результати асинхронних операцій (запитів до серверу). Основний потік роботи на сайті при цьому не блокується та не оновлюється.

Promise може перебувати в одному з трьох станів: Pending (очікування), коли операція ще не завершена; Fulfilled (виконано), коли операція завершена успішно, а результат – відповідь від серверу – повернено; Rejected (відхилено), коли сталася помилка, наприклад, запит не вдалося передати.

```
let promise = fetch(url, [options])
```

Рисунок 1. Базовий синтаксис promise

У синтаксисі promise (рис. 1) url – це посилання для запитів, а options – додаткові параметри (методи, заголовки тощо). Без використання налаштувань об'єкт буде виконувати функцію завантаження змісту зазначеного у першому параметрі сайту.

Fetch використовує синтаксис об'єктів Request та Response, що дозволяє користуватись ними іншими сервісами та API, які займаються обробкою та зміною запитів і відповідей. Також їх можна застосовувати для генерації власних програмних відповідей.

Найчастіше новий об'єкт Request створюється за допомогою service.worker FetchEvent.request, але можна використовувати й конструктор Request() (рис. 2).

```
const request = new Request("https://www.mozilla.org/favicon.ico");

const URL = request.url;
const method = request.method;
const credentials = request.credentials;
```

Рисунок 2. Приклад створення Request [2]

Новий об'єкт Response створюється за допомогою або Response.Response(), або service worker Fetchevent.respondWith (рис. 3).

```
const response = new Response();
```

Рисунок 3. Приклад створення Response [3]

Запит відсилається браузером та повертає promise для того, щоб зовнішній код міг використати його для отримання результату.

Процес отримання відповіді проходить у два етами:

1. Як тільки сервер присилає заголовки відповіді, promise виконується з об'єктом вбудованого класу Response у якості результату.
   Вже можна переглянути статус запиту та з'ясувати, виконався він успішно чи отримано помилку. Помилкою promise завершується, якщо мережа була нестабільною та виникли перебої, або обраного сайту не існує зовсім. При цьому, при отримані навіть таких помилок HTTP, як 404 та 500, promise не буде відкинуто, замість цього він повернеться із статусом "ok", чиє логічне значення дорівнюватиме "false".
   Також можна переглянути заголовки відповіді, але без тіла.
2. Для отримання тіла відповіді треба скористатися додатковими методами. Усі вони надають різні формати для доступу:
   - text() –прочитавши відповідь, повертає звичайний текст;
   - json() – відповідь декодується у форматі JSON;
   - formData() – повертає об'єкт типу FormData;
   - blob() – повертає тип Blob, що є типованими бінарними даними;
   - arrayBuffer() – повертає тип ArrayBuffer, що є низькорівневим представленням бінарних даних.

   Також існує body – об'єкт ReadableStream, що надає можливість розбивати тіло запита на частини та зчитувати їх незалежно один від одного.

Fetch може працювати з заголовками credentials, що полегшує роботу з сесіями та автентифікацією. Він не буде відправляти cookie-файли, якщо вручну не змінити цей параметр при налаштуванні API.

3.3. Переваги та недоліки

Переваги:

Надає сучасний та зручний інтерфейс для роботи з HTTP-запитами, заснований на використанні promises. Це покращує читаність і спрощує підтримку коду, оскільки promises дозволяють позбутися складних коллбеків, роблячи логіку обробки запитів лінійною та зрозумілою.

Автоматична робота із форматами даних, такими як JSON. Розробникам не потрібно вручну обробляти відповідь сервера – достатньо викликати відповідний метод, і дані будуть автоматично перетворені на об'єкт, що значно спрощує процес взаємодії з API.

Синтаксис Fetch API набагато лаконічніший і простий у порівнянні зі застарілим XMLHttpRequest. Це скорочує кількість шаблонного та повторюваного коду, роблячи розробку швидше та ефективніше.

Надійні механізми обробки помилок. У Fetch вбудована підтримка вилову мережевих збоїв та некоректних відповідей сервера. Це дозволяє легко відстежувати помилки та реагувати на них, забезпечуючи стабільнішу роботу додатків.

Недоліки:

Fetch API не підтримується в старих браузерах, таких як Internet Explorer версій нижче 11, що може стати проблемою розробки додатків, орієнтованих на подібні системи. Щоб уникнути цього обмеження, доводиться використовувати поліфіли або сторонні бібліотеки, які додають підтримку Fetch у застарілі браузери.

Відсутність вбудованої обробки тайм-аутів. Fetch не завершує запит автоматично, якщо він займає занадто багато часу, що потребує ручної реалізації даної функції.

Для деяких специфічних завдань, таких як відстеження прогресу завантаження або роботи з типами даних, відмінними від JSON (наприклад, бінарні дані або великі файли), може знадобитися більше зусиль та додаткових рішень. Це накладає певні обмеження на його використання у складніших сценаріях, особливо, коли потрібен точний контроль над процесом завантаження або відправлення даних.

Перехід з XHR на Fetch API може викликати складнощі у тих розробників, які не знайомі з концепціями promises та асинхронного програмування. Вивчення цих нових для них підходів потребуватиме часу та адаптації. [4]

3.4. Можливості застосування

Інтерфейс, який ми розглядаємо, може бути застосований у багатьох сферах програмування: веб-розробка (обробка запитів та даних з серверів), мобільні та крос-платформні додатки, односторінкові сайти (лендінги) та інтерактивні веб-додатки.

4. ВИСНОВКИ

1) Розглянуто загальний синтаксис та методи для роботи з Fetch API.
2) Описані можливості роботи Fetch із різними файловими та мережевими поняттями.
3) Перераховані переваги та недоліки API, яке досліджується, як самостійно, так і в порівнянні із більш застарілою технологією.
4) Описані можливі сфери застосування інтерфейсу.

Список літератури

[1] Матеріал з вільного сайту «MDN Web Docs» [Електронний ресурс] Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction

[2] Матеріал з вільного сайту «MDN Web Docs» [Електронний ресурс] Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Request

[3] Матеріал з вільного сайту «MDN Web Docs» [Електронний ресурс] Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Response

[4] Матеріал з вільного сайту «Geeksforgeeks» [Електронний ресурс] Режим доступу: https://www.geeksforgeeks.org/difference-between-ajax-and-fetch-api/?_x_tr_sl=en&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_pto=rq