# 03-02 fetch API

Immagine che contiene testo, schermata, diagramma, design

Descrizione generata automaticamenteObiettivo: caricare dati da un server in maniera asincrona attraverso richieste http.

Applicazione react che viene visualizzata ed eseguita all’interno di un browser con cui un utente può interagire:

* Fare chiamate http per poter interagire con un server che usa un browser sqlite
  + Connettere i due componenti (applicazione react-server api)
    - Recuperare dati da server api
    - Visualizzare su app react, modificare su app react
    - Inviare modifiche al server
* Fetch → chiamate http
* useEffect → gestire side effect in componenti react

Scambiare dati in maniera asincrona tra browser-server:

* api **fetch**
  + metodo http che ogni browser conosce e riesce ad utilizzare in javascript
  + parametri:
    - url risorsa a cui fare richiesta http
    - parametri per richiesta http
    - di default la richiesta è una GET ma è possibile gestire POST, PUT,…
  + ritorna una Promise
    - response → permette di accedere ai dettagli della rispsota http e dell’eventuale contenuto del suo body
    - rifiutata solo in caso di problemi di rete
      * *in caso di errori (404, 502, ..) non viene rifiutata ma torna con response, dobbiamo verificare noi*

*Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente  
  
  
dentro un async  
  
(altrimenti non   
  
potrei usare await)*

#### **Proprietà di response:**

* Response.**ok** (boolean): *HTTP successful (code 200-299)*
* Response.**status**, Response.statusText → stato intero della risposta http
* Response.**headers**: collection di eventuali HTTP headers of the response
* Response.**url**: final URL (potentially after HTTP redirects)
  + Nella maggior parte dei casi è la stessa su cui facciamo la domanda
* Response.**body**: a readable stream of the body content
  + Stream di informazioni → va convertito al corpo vero e proprio
    - .json()

#### Accedere agli header della risposta:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

### Gestione errori

La promise è rifiutata solo per errori di rete → errori relativi alla connessione-

negli altri casi, ritorna sempre una promise risolta con successo, per gestire gli errori del contenuto http, bisogna controllare:

* response.**ok** (true/false):
  + True → ok
  + False → controllare status per comprendere cosa e successo e gestirlo al meglio
* **Content type** da header
  + Se mi aspetto un json, verifico che content type sia json

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

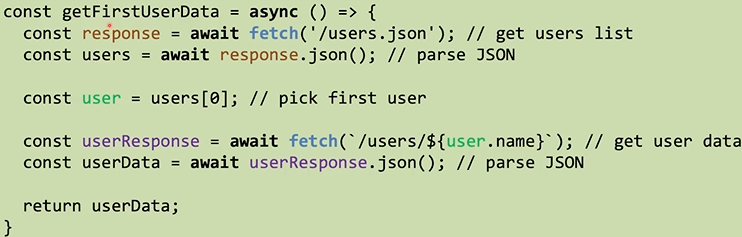
**La fetch ha delle opzioni:**

* Primo parametro → url
* Oggetto opzionale come secondo parametro che permette di definire eventuali proprietà
  + **Method**
    - Metodo http per indicare di non fare get ma PUT/POST/DELETE
  + **Headers**
    - Nel caso in cui serva passare queste informazioni nella richiesta http
  + **Body**
    - Nel caso in cui la nostra richiesta http abbia un corpo da passare
  + **Mode**
    - Modo in cui la richiesta viene effettuata
      * Cors, no-cors, same-origin
  + **Credentials**
    - Per mandare cookie con la richiesta
  + **Signal**
    - Immagine che contiene testo, schermata, Carattere

      Descrizione generata automaticamenteTerminare in un momento specifico la richiesta di catch

**Fetch torna una promise** contenente un oggetto, se vogliamo accedere al corpo di questa risposta:

* response.**text**(): as plain text (string)
* response.**json**(): as a JS object, by parsing the body as JSON
* response.**formData**(): as a FormData object
* response.**blob**(): as Blob (binary data with type)
* response.**arrayBuffer**(): as ArrayBuffer (low-level representation of binary data)
* → questi metodi **ritornano una promise**, quindi bisogna aspettare usanso
  + Await-async
  + Then catch
* →questi metodi possono conservare il body solo una volta
  + *O faccio .text o .json, non posso fare uno successivamente all’altro*

Nel caso in cui ci sia bisogno di effettuare **fetch sequenziali**:

Nel caso in cui servisse effettuare **fetch in parallelo**, possiamo utilizzare promise.all e aspettare che tutte vengano soddisfatte e processare le informazioni in base all’ordine in cui l’applicazione ne ha bisogno.

