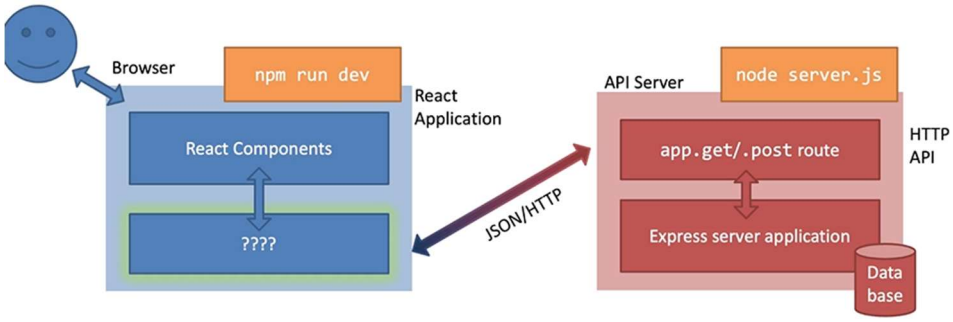
03-02 fetch API

Obiettivo: 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*

```
fetch('http://example.com/exams.json')
  .then((response) => {
    return response.json();
  })
  .then((data) => {
    console.log(data);
  });
```

```
const response = await
     fetch('http://example.com/exams.json');

const data = await response.json();

console.log(data);
```

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:

```
fetch('http://localhost/data.json')
  .then(response => {
    console.log(response.headers.get('Content-Type'));
    console.log(response.headers.get('Date'));

    console.log(response.status);
    console.log(response.statusText);
    console.log(response.type);
    console.log(response.url);
  }
```

```
application/html; charset=utf-8
Sat, 11 Apr 2020 13:41:04 GMT

404
Not Found
undefined
http://localhost/data.json
```

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

```
fetch(url)
  .then(response => {
    if (!response.ok) { throw Error(response.statusText) }
    let type = response.headers.get('Content-Type');
    if (type !== 'application/json') {
        //then() returns a rejected promise if something is thrown
        throw new TypeError(`Expected JSON, got ${type}`)
    }
    return response;
  })
  .then(response => {
  //...
  })
  .catch(err => console.log(err))  // either the throw value or other errors
```

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**
    - Terminare in un momento specifico la richiesta di catch

```
let objectToSend = {'title': 'Do homework' , 'urgent': true, 'private': false,
'sharedWithIds': [3, 24, 58] };

fetch(url, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(objectToSend),  // Conversion in JSON format
  })
  .catch(function (error) {
    console.log('Failed to store data on server: ', error);
  });
```

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**:

```
const getFirstUserData = async () => {
  const response = await fetch('/users.json'); // get users list
  const users = await response.json(); // parse JSON

  const user = users[0]; // pick first user

  const userResponse = await fetch(`/users/${user.name}`); // get user data
  const userData = await userResponse.json(); // parse JSON

  return userData;
}
```

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.

```
// array of URLs
const urls = [url1, url2];

// Convert to an array of Promises
const promises = urls.map(url => fetch(url) );
// Wait only for the fetch Promise

// Run all promises in parallel, wait for all
Promise.all(promises)
  .then(results => { // process according to the order needed by the app
       for (const res of results) res.text().then( t => console.log(t) );
  })
  .catch(e => console.error(e))
```