Resurse suplimentare
pe care recomand să le parcurgi

Cum funcționează comunicarea client-server

```
[Client] ↔ [Internet] ↔ [Server]
  ↓                        ↓
Browser                Web Server
Mobile App             API Server
Desktop App            Database Server
```

Fluxul de comunicare:

1. **Client-ul inițiază cererea** - Utilizatorul face o acțiune (click, form submit, etc.)
2. **Cererea este trimisă** - Browser-ul trimite o cerere HTTP către server
3. **Server-ul procesează** - Server-ul primește cererea și o procesează
4. **Server-ul răspunde** - Server-ul trimite înapoi datele solicitate
5. **Client-ul afișează** - Browser-ul primește răspunsul și actualizează interfața

Ce este protocolul HTTP?

HTTP (HyperText Transfer Protocol) este protocolul de comunicare standard folosit pentru transferul de date pe web. Este un protocol de tip request-response care definește cum client-ul și server-ul comunică.

Caracteristicile HTTP:

- **Stateless** - Fiecare cerere este independentă
- **Text-based** - Mesajele sunt în format text lizibil
- **Request-Response** - Client-ul trimite cereri, server-ul răspunde
- **Port standard** - HTTP folosește portul 80, HTTPS portul 443

Structura unei cereri HTTP

O cerere HTTP conține următoarele componente:

```
GET /api/users HTTP/1.1
Host: example.com
Accept: application/json
Authorization: Bearer token123
User-Agent: Mozilla/5.0

{optional body data}
```

Componentele cererii:

1. **Metoda HTTP** - GET, POST, PUT, DELETE, etc.
2. **URL-ul** - Calea către resursa solicitată
3. **Versiunea HTTP** - De obicei HTTP/1.1 sau HTTP/2
4. **Headers** - Metadate despre cerere
5. **Body** - Date opționale (pentru POST, PUT)

Metodele HTTP principale

GET - Obținerea datelor

```
// Exemplu de cerere GET
fetch('<https://api.example.com/users>')
  .then((response) => response.json())
  .then((users) => console.log(users))
```

Caracteristici:

- Folosită pentru a obține date
- Nu modifică starea server-ului
- Parametrii sunt în URL (query string)
- Poate fi cache-ată de browser

POST - Crearea de resurse noi

```
// Exemplu de cerere POST
fetch('<https://api.example.com/users>', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    name: 'John Doe',
    email: 'john@example.com',
  }),
})
```

Caracteristici:

- Folosită pentru a crea resurse noi
- Conține date în body
- Nu este idempotentă (apeluri multiple = efecte multiple)

PUT - Actualizarea completă

```
// Exemplu de cerere PUT
fetch('<https://api.example.com/users/123>', {
  method: 'PUT',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    id: 123,
    name: 'John Smith',
    email: 'johnsmith@example.com',
  }),
})
```

Caracteristici:

- Înlocuiește complet o resursă existentă
- Idempotentă (același rezultat la apeluri multiple)
- Conține datele complete ale resursei

PATCH - Actualizarea parțială

```
// Exemplu de cerere PATCH
fetch('<https://api.example.com/users/123>', {
  method: 'PATCH',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    name: 'John Smith', // Doar câmpurile care se schimbă
  }),
})
```

DELETE - Ștergerea resurselor

```
// Exemplu de cerere DELETE
fetch('<https://api.example.com/users/123>', {
  method: 'DELETE',
})
```

Headers HTTP importante

Request Headers (trimise de client):

```
{
  'Accept': 'application/json',           // Tipul de conținut dorit
  'Content-Type': 'application/json',     // Tipul conținutului trimis
  'Authorization': 'Bearer token123',     // Autentificare
  'User-Agent': 'Mozilla/5.0...',        // Informații despre client
  'Accept-Language': 'ro,en;q=0.9',      // Limbile preferate
  'Cache-Control': 'no-cache'            // Instrucțiuni de cache
}
```

Response Headers (trimise de server):

```
{
  'Content-Type': 'application/json',     // Tipul conținutului răspuns
  'Content-Length': '1234',               // Dimensiunea răspunsului
  'Cache-Control': 'max-age=3600',        // Instrucțiuni de cache
  'Access-Control-Allow-Origin': '*',     // Politici CORS
  'Set-Cookie': 'sessionid=abc123',       // Setarea de cookie-uri
  'Location': '/new-resource'             // Pentru redirect-uri
}
```

Codurile de status HTTP

Codurile de status indică rezultatul unei cereri HTTP:

1xx - Informații

- **100 Continue** - Server-ul așteaptă restul cererii

2xx - Succes

- **200 OK** - Cererea a fost procesată cu succes
- **201 Created** - O nouă resursă a fost creată
- **204 No Content** - Succes, dar fără conținut de returnat

3xx - Redirect

- **301 Moved Permanently** - Resursa s-a mutat permanent
- **302 Found** - Redirect temporar
- **304 Not Modified** - Conținutul nu s-a schimbat (cache valid)

4xx - Erori client

- **400 Bad Request** - Cererea este malformată
- **401 Unauthorized** - Autentificare necesară
- **403 Forbidden** - Acces interzis
- **404 Not Found** - Resursa nu există
- **429 Too Many Requests** - Prea multe cereri

5xx - Erori server

- **500 Internal Server Error** - Eroare generală a server-ului
- **502 Bad Gateway** - Problemă cu proxy/gateway
- **503 Service Unavailable** - Server-ul este temporar indisponibil

Gestionarea erorilor HTTP

```
async function handleHttpResponse(response) {
  if (response.ok) {
    // Status 200-299
    return await response.json()
  }

  // Gestionarea diferitelor tipuri de erori
  switch (response.status) {
    case 400:
      throw new Error('Cererea este invalidă')
    case 401:
      throw new Error('Nu ești autentificat')
    case 403:
      throw new Error('Nu ai permisiuni')
    case 404:
      throw new Error('Resursa nu a fost găsită')
    case 429:
      throw new Error('Prea multe cereri, încearcă mai târziu')
    case 500:
      throw new Error('Eroare de server')
    default:
      throw new Error(`Eroare HTTP: ${response.status}`)
  }
}

// Utilizare
try {
  const response = await fetch('/api/users')
  const data = await handleHttpResponse(response)
  console.log(data)
} catch (error) {
  console.error('Eroare:', error.message)
}
```

Ce sunt API-urile REST?

REST (Representational State Transfer) este un stil arhitectural pentru designul API-urilor web care folosește HTTP într-un mod standardizat.

Principiile REST:

1. **Stateless** - Fiecare cerere conține toate informațiile necesare
2. **Resource-based** - Totul este o resursă identificată prin URL
3. **HTTP methods** - Folosește metodele HTTP pentru operații CRUD
4. **Representation** - Resursele pot avea multiple reprezentări (JSON, XML)

Exemplu de API REST:

```
GET    /api/users          - Obține toți utilizatorii
GET    /api/users/123      - Obține utilizatorul cu ID 123
POST   /api/users          - Creează un utilizator nou
PUT    /api/users/123      - Actualizează utilizatorul 123
DELETE /api/users/123      - Șterge utilizatorul 123
```

JSON - Formatul de date standard

JSON (JavaScript Object Notation) este formatul predominant pentru schimbul de date în aplicațiile web.

Structura JSON:

```json
{
  "id": 1,
  "name": "John Doe",
  "email": "john@example.com",
  "active": true,
  "roles": ["user", "admin"],
  "profile": {
    "age": 30,
    "city": "București"
  },
  "lastLogin": null
}
```

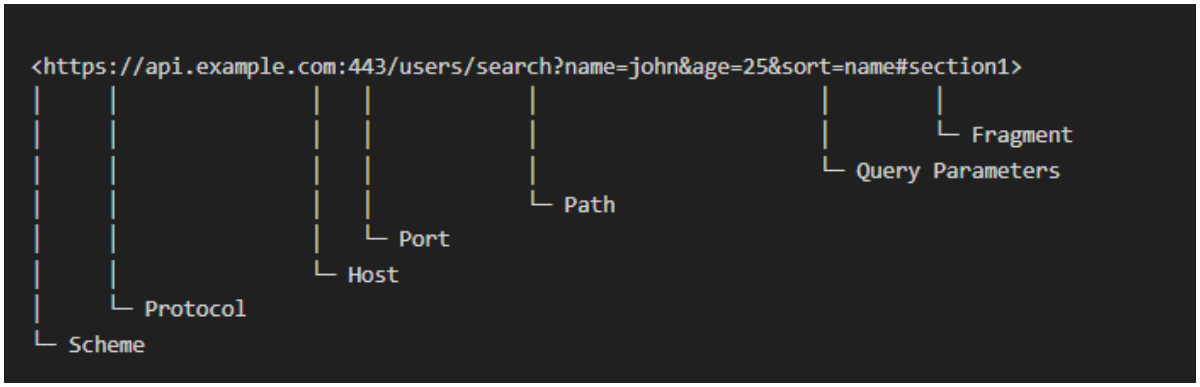
Lucrul cu JSON în JavaScript:

```
// Convertirea din JavaScript object în JSON string
const user = { name: 'John', age: 30 }
const jsonString = JSON.stringify(user)
console.log(jsonString) // '{"name":"John","age":30}'

// Convertirea din JSON string în JavaScript object
const userData = '{"name":"John","age":30}'
const userObject = JSON.parse(userData)
console.log(userObject.name) // 'John'
```

Query Parameters și URL-uri

Structura unui URL complet:



Query Parameters în JavaScript:

```
// Construirea URL-urilor cu parametri
const baseUrl = '<https://api.example.com/users>'
const params = new URLSearchParams({
  page: 1,
  limit: 10,
  search: 'john doe',
  status: 'active',
})

const fullUrl = `${baseUrl}?${params.toString()}`
// Rezultat: <https://api.example.com/users?page=1&limit=10&search=john+doe&status=active
>

// Extragerea parametrilor din URL
const url = new URL('<https://api.example.com/users?page=2&limit=5>')
console.log(url.searchParams.get('page')) // '2'
console.log(url.searchParams.get('limit')) // '5'
```

CORS - Cross-Origin Resource Sharing

CORS este o politică de securitate care controlează cum paginile web pot accesa resurse de pe alte domenii.

Probleme CORS comune:

```
// Această cerere va fi blocată de CORS dacă server-ul nu permite
fetch('<https://api.altdomeniu.com/data>')
  .then((response) => response.json())
  .catch((error) => {
    console.error('CORS Error:', error)
    // Error: Access to fetch at '<https://api.altdomeniu.com/data>'
    // from origin '<https://meudomeniu.com>' has been blocked by CORS policy
  })
```

Rezolvarea problemelor CORS:

Server-ul setează header-e CORS:

```
Access-Control-Allow-Origin: <https://meudomeniu.com>
Access-Control-Allow-Methods: GET, POST, PUT, DELETE
Access-Control-Allow-Headers: Content-Type, Authorization
```

Folosirea unui proxy pentru dezvoltare:

```
// În loc de cererea directă, folosește un proxy local
fetch('/api/proxy/altdomeniu/data')
```

Cereri cu mode CORS:

```
fetch('<https://api.altdomeniu.com/data>', {
  mode: 'cors', // 'cors', 'no-cors', 'same-origin'
  credentials: 'include', // pentru trimiterea cookie-urilor
})
```