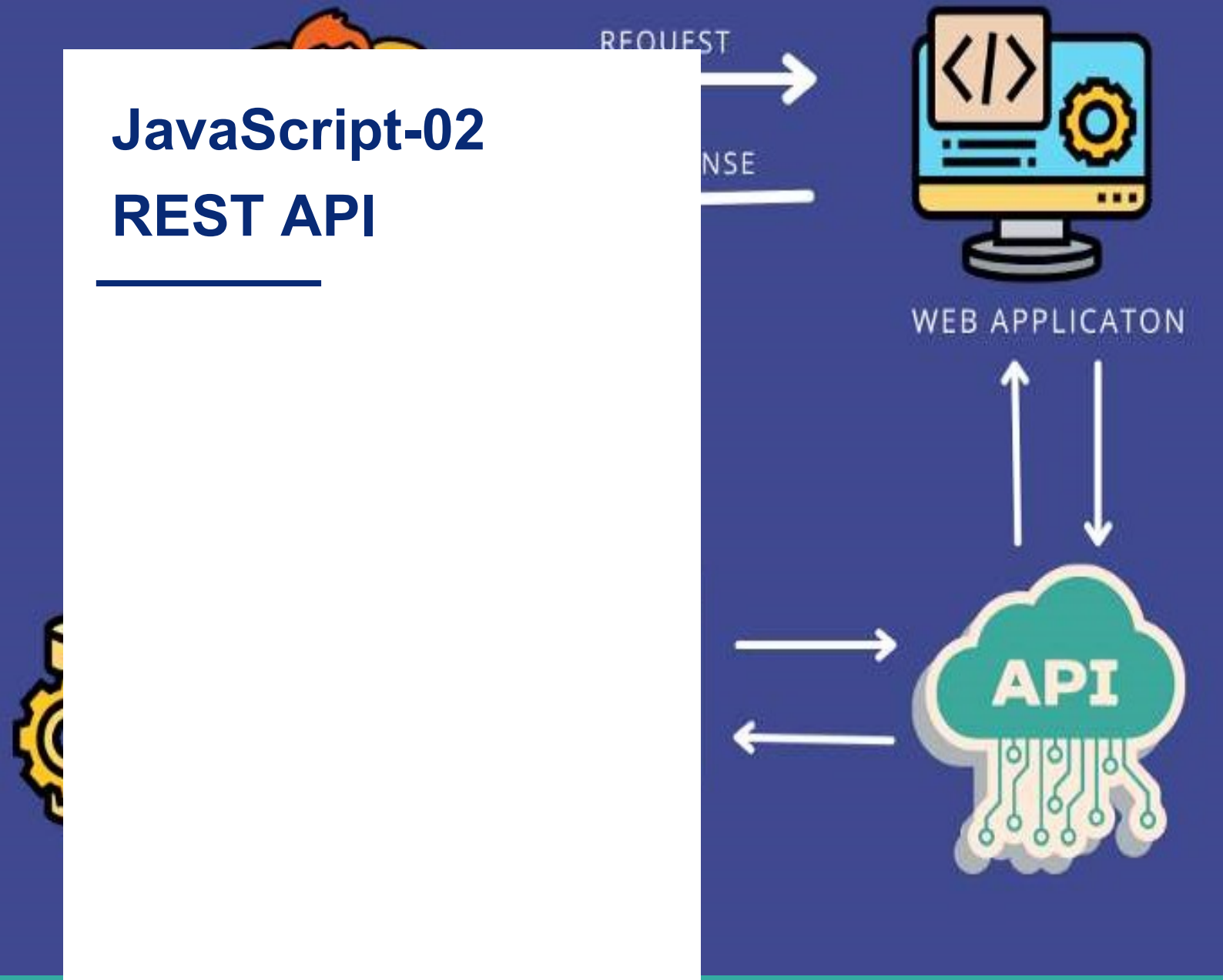
JavaScript-02
REST API



19 OKTOBER 2024

JavaScript – utan tjafs

Författare: Acke Strömberg

För en kanin

REST API

Client - server

När man skriver in adressen till en webbsida i en webbläsare och anropar en Internet-adress, så kallas den sida som gör anropet för ”client” och den andra sidan för ”server”. Det är alltså servern som har alla filer som behövs för att visa webbsidan och skickar iväg dessa när någon client skriver in adressen. Man kan tänka sig att det ser ut ungefär som på bild 1. Man ser att det är flera clients som anropar en webbserver.

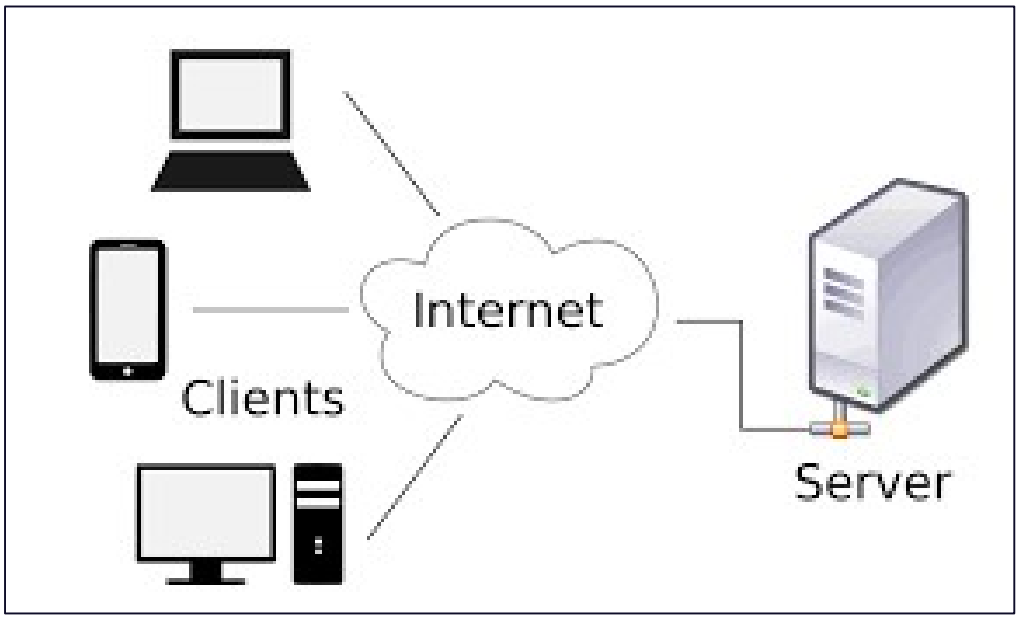


Bild 1 Flera clients anropar en webbserver.

Säg att man anropar en sida med varor. Om man söker på verktyg så skickar webbservern tillbaka resultat med verktyg i sökningen. Problemet för webbservern är att den inte vet vad nästa client kommer att vilja ha sökresultat för. Det kanske är virke. Därför måste servern i sin tur vända sig till någon som har koll på det. Här kommer ett REST API in i bilden. Man kan säga att servern tillhandahåller ett sätt att kommunicera för den applikationen som körs. Bild 2 visar en principskiss för client som kommunicerar med ett REST API mot en server.

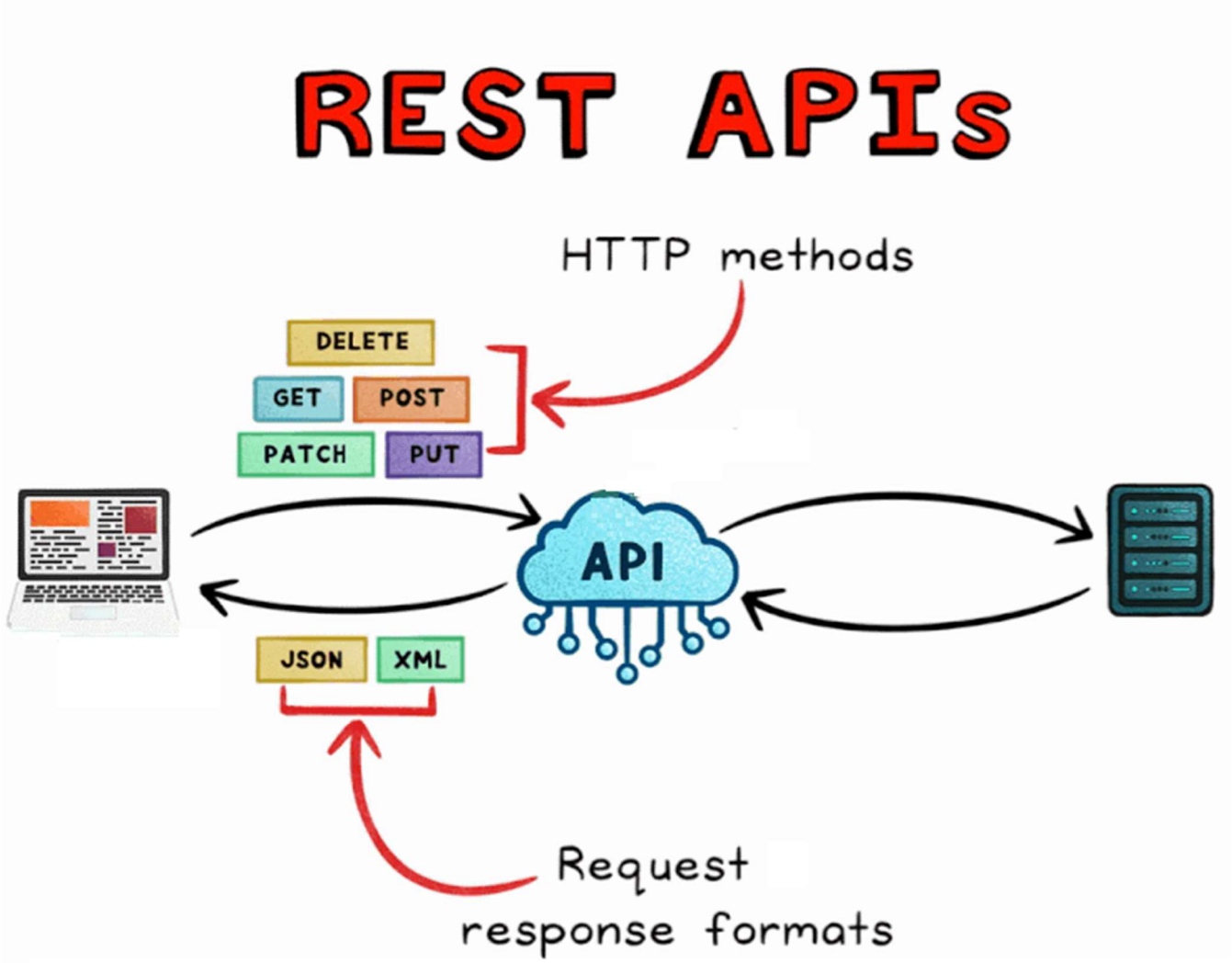


Bild 2 En principskiss för kommunikationen mellan en client, server genom ett REST API.

Om man vill se alla verktyg som finns så kan det på den här webbsidan finnas en URL (= webbadress) som applikationen (= client) kan använda för att via REST API:et få servern att skicka tillbaka ett svar med alla verktyg. Se bild 3 för exempel på en URL som en webbapplikation använder sig av.

```
https://jsonplaceholder.typicode.com/todos
```

Bild 3 En URL för en applikation som vill hämta alla todos via ett REST API.

Webbapplikationen gör ett anrop till REST API:et som har grundadressen https://jsonplaceholder.typicode.com . Efter den kommer ett snedstreck och den resurs man vill hämta. Enligt bild 3 var det fråga om alla todos som servern har just nu.

Vad kan man hämta och skicka?

Det är de personer som har skrivit ihop REST API:et som bestämmer vad som går att hämta och hur man kan kommunicera det. Det finns alltså uppsatta regler som applikationen behöver följa för att ha möjlighet att hämta och skicka information via REST API:et. Om man följer länken https://jsonplaceholder.typicode.com så finns det först och främst ett stycke som heter ”Resources”. Man kan se att det finns sex olika resurser att hämta och skicka till servern enligt bild 4.

/posts	**100 posts**
/comments	**500 comments**
/albums	**100 albums**
/photos	**5000 photos**
/todos	**200 todos**
/users	**10 users**

Bild 4 De sex olika resurser som man kan hämta och skicka till https://jsonplaceholder.typicode.com.

Synkront och asynkront

En server som har ett REST-API har mycket att göra och det är inte alltid den är beredd på att ta emot ett anrop och skicka tillbaka ett svar. Den kan vara upptagen med ett annat anrop från en annan klient när det kommer in ett nytt anrop. Därför är alltid svaret man får från ett REST API asynkront, det vill säga det är inte synkroniserat. Asynkront innebär alltså att instruktionen där man vill hämta något i koden inte utförs på direkten, utan det är olika stor väntetid varje gång koden körs. Se bild 5 för att se ett stycke JavaScript-kod med både synkron och asynkron kod.

```
// Synkron kod
const knapp = document.querySelector(”button”)
knapp.addEventlistener(”click”, function() {
  console.log(“klickat på knappen”)
})

// Asynkron kod
fetch('https://jsonplaceholder.typicode.com/posts')
  .then((response) => response.json())
  .then((json) => console.log(json));

// Synkron kod
console.log(“Nu är allt kört”)
```

Bild 5 Exempel på ett GET-anrop för att hämta alla poster som finns på servern.

En JavaScript-fil körs uppifrån och ner, rad för rad. Den stannar inte upp, utan varje rad och kommando exekveras (= utförs) direkt. Om man vill vänta på ett kommando behöver man säga till programmet att ”Hej! Nu vill jag att du tar det lugnt här ett tag.”. Enligt bild 5 kan man göra det genom att ”koppla” till en .then()-sats efter den första funktionen fetch(). Då stannar JavaScript-programmet upp och väntar in det som händer i .then()-satsen tills det är färdigt. Därefter kan programmet köra vidare med den sista synkrona raden i programmet.

HTTP-metoder

När man gör ett anrop till ett REST-API behöver man berätta för servern vilken typ av anrop man vill gör. Om vi kör vidare på vårt publika exempel med jsonplaceholder ovan, så tillåter det API:et GET, POST, PUT, PATCH och DELETE. Alla REST-API:er tillåter inte alla typer av anrop. Det beror helt enkelt på hur det är programmerat. I styckena nedan förklaras hur de olika typerna av HTTP-metoder fungerar och vad de kan göra.

GET

Syftet med GET är helt enkelt att bara hämta data från servern. Det kan vara en enskild webbsida, en bild, ett JSON-dokument, en CSS-file eller annan typ av fil. Det är en så kallad säker operation som inte ändrar status på den typen av resurser som servern har. Se bild 6 för ett exempel på ett GET-anrop.

```
fetch('https://jsonplaceholder.typicode.com/posts')
  .then((response) => response.json())
  .then((json) => console.log(json));
```

Bild 6 Exempel på ett GET-anrop för att hämta alla poster som finns på servern.

POST

Ett POST-anrop används för att skicka data till REST-API:et. Det kan vara data som hämtas från ett webbformulär, från ett JSON-dokument eller från parametrar från en URL. Se bild 7 för ett exempel på ett POST-anrop.

```
fetch('https://jsonplaceholder.typicode.com/posts', {
  method: 'POST',
  body: JSON.stringify({
    title: 'foo',
    body: 'bar',
    userId: 1,
  }),
  headers: {
```

```
    'Content-type': 'application/json; charset=UTF-8',
  },
})
  .then((response) => response.json())
  .then((json) => console.log(json));
```

Bild 7 Exempel på ett POST-anrop för att skicka data till.

Enligt bild 7 så använder man fortfarande funktionen fetch(). I exemplet med ett GET-anrop där man endast använde ett argument, en sträng med den URL som man vill hämta ifrån. Men här med POST-anropet är det två argument. Det första är fortfarande en sträng med den URL man vill skicka till, https://jsonplaceholder.typicode.com . Efter URL:en som är en sträng, kommer ett komma och ett argument till. Det består av ett stort objekt, med flera attribut, se bild 8 för objektet.

```
{
  method: 'POST',
  body: JSON.stringify({
    title: 'foo',
    body: 'bar',
    userId: 1,
  }),
  headers: {
    'Content-type': 'application/json; charset=UTF-8',
  },
}
```

Bild 8 Det stora objektet som är argument nummer två i fetch()-funktionen.

Det är attributet ”method” som man inte behöver ange om man bara kör ett vanligt GET-anrop. Sen finns attributet ”body” med som behöver vara en sträng. Slutligen finns och attributet ”headers” med som anger vilken typ av data man skickar till servern. Det som finns med under bild 8 är inget som man måste kunna utantill, utan det finns oftast med som dokumentation från de som har gjort REST API:et från första början. Finns det inte med där av någon anledning så kan man söka efter JavaScript fetch på Internet så bör man hitta den informationen som behövs.

Övriga HTTP-metoder

Totalt finns det nio HTTP-metoder, men det är tillräckligt att känna till de fem vanligaste. Det är GET och POST som har nämnts ovanför. Sen finns det dessutom PUT och PATCH för att uppdatera information om en specifik post i REST API:et. Sen är metoden DELETE också vanligt förekommande. Den används för att bort data från ett REST API.