Varför behövs andra sätt att skicka data till servern?

...vi har ju redan sökparametrar i GET-request

Säkerhet: Sökparametrar är synliga i URL:en, vilket kan vara en säkerhetsrisk om känslig information skickas. Request body är inte synlig i URL:en.

Begränsningar i URL-längd: URL:er har en maximal längdbegränsning (vanligtvis runt 2000 tecken).

Struktur och komplexitet: Request body kan innehålla mer komplex data, såsom JSON-strängar som representerar objekt och arrayer.

HTTP-metoder: GET-förfrågningar är avsedda för att hämta data och bör inte användas för att göra ändringar på servern (så som lägga till eller ändra data i databasen).

Request body

Request-body anges som ett attribut av options när man skapar en ny förfrågan:

```
const request = new Request("http://localhost:8000/", {
  method: "POST",
  body: "body test",
  headers: {
    "Content-Type": "text/plain",
  }
});
```

Beroende på vilken typ av data som servern accepterar så skickas body som "text/plain" eller "application/json" eller annat.

Hantering av body på servern

Tidigare i kursen har vi sett hur servern tar emot data som skickats via request-body:n:

```
async function handler(request) {
    if (request.method == "POST") {
        const movie = await request.json();
        console.log(movie.title, movie.year);
    }
    return new Response("Hello!");
}

Deno.serve(handler);
```

Request-bodyn tas alltså emot asynkront med hjälp av request-objektets metoder .json(), .text(), etc.

Precis på samma sätt som Response-objektet hade metoder .json(), .text(), etc. för att ta emot responsens "body" (alltså resursen som servern skickar).

Du känner igen async-await syntaxen. Det går också att använda .then() på servern, men det är onödigt komplicerat så vi tar inte upp det i DU3.

Request body

Metoderna POST, PUT, PATCH och DELETE använder request-body

Förfrågningar med metoderna ovan kodar body:n på exakt samma sätt, som ett attribut av options.

Body:n av alla dessa förfrågningar tas emot av servern på exakt samma sätt.