Engineer

Progressive enhancement

å bygge sider som virker i ren HTML, og så legge på mer

Hva det er

Progressive enhancement er en måte å bygge nettsider på i lag, der man begynner med det laget som alltid virker. Bunnen er HTML som inneholder alt innholdet og klarer hver oppgave på egen hånd: lenker fører et sted, skjemaer sendes til en server, og siden kan leses og brukes uten at noe annet er lastet inn. Stilark legges oppå for å sette opp siden og gjøre den behagelig å se på. Skript kommer sist og gjør den raskere eller smidigere å bruke. Hvert lag forbedrer det som ligger under, og siden er ikke avhengig av noe lag over HTML-en for å gjøre jobben sin. GOV.UKs Service Manual koker idéen ned til å få siden til å virke med HTML først. På norsk kalles det også progressiv forbedring.

Navnet kommer fra et foredrag av Steven Champeon og Nick Finck, *Inclusive Web Design For the Future with Progressive Enhancement*, som ifølge den skriftlige versjonen av foredraget ble holdt 11. mars 2003 på SXSW i Austin i Texas. Teksten skriver Progressive Enhancement med store forbokstaver og forkorter det til PE. Den stiller idéen opp mot graceful degradation, den eldre vanen med å bygge for den mest kapable nettleseren og håpe at de andre fortsatt kunne nå innholdet. PE begynner ifølge teksten med et dokument som de minst kapable enhetene kan bruke, og legger deretter presentasjonen på som separat logikk, slik at de enkle enhetene ikke tynges av det de moderne nettleserne får.

Forskjellen mellom de to ligger i retningen på arbeidet. Graceful degradation tar utgangspunkt i hele opplevelsen og spør hva som er igjen når deler av den mangler. Progressive enhancement tar utgangspunkt i det som alltid er der, og spør hva som kan legges til. I en nyere nettleser på en god forbindelse kan begge ende med den samme siden. Forskjellen viser seg når noe svikter: en side som er bygd nedenfra, har fortsatt bunnen sin, mens en side som er bygd ovenfra, bare har det forfatterne husket å sørge for.

Det er en arbeidsmåte, ikke et bibliotek eller en funksjon. Ingenting i HTML, CSS eller JavaScript heter progressive enhancement, men standardene gir rom for det. En nettleser hopper over en CSS-deklarasjon den ikke forstår, og ignorerer et skript med en type den ikke kjenner, og regelen `@supports` lar et stilark spørre om en funksjon finnes før det regner med den.

Bruk begrepet i en beslutning

Bruk «Hva er progressive enhancement?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Grensesnitt som laster raskt, fungerer på mobilen og oppfører seg likt i alle nettlesere kundene dine måtte bruke.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Styrte fullstack-utviklingen av GIS-kart — fullstack-kartutvikling, fra database til nettleser

Styrte fullstack-utviklingen av GIS-kart Ta kontakt →

Hvorfor det er viktig

Lagene svikter på ulike måter, og GOV.UKs Service Manual gjør rede for hvordan. CSS-laget er rimelig tilgivende: en nettleser hopper over en deklarasjon den ikke kjenner igjen, og fortsetter, men en selektor den ikke kjenner, får den til å forkaste hele regelen. JavaScript-laget tilgir ikke i det hele tatt. Når et skript bruker syntaks eller et grensesnitt som nettleseren mangler, kjører ikke resten av skriptet. En side som trenger skriptet sitt til hovedoppgaven, er derfor bare så pålitelig som hvert eneste ledd mellom serveren og leseren.

Og det er mange ledd. Den samme siden lister opp grunner til at CSS og JavaScript kan la være å bli lastet inn eller kjørt: en midlertidig nettverksfeil, en annonseblokkerer, et CDN som er nede, en DNS-feil, en feil i en nettleseroppdatering, en brannmur på jobben eller hjemme. Ingen av dem har noe med en gammel nettleser å gjøre, og leseren kan ikke se hvilken av dem det var. Det eneste leseren ser, er en knapp som ikke gjør noe. Manualen oppgir ingen tall for hvor ofte hver av dem skjer; argumentet hviler på listen, ikke på en prosentsats.

Noen lesere slår også av ting med vilje. Manualen skriver at noen brukere bevisst slår av funksjoner i nettleseren sin, og at valget deres skal respekteres. HTML-standarden skriver at mange brukere slår av skripting, og av den grunn, og fordi deklarative mekanismer ofte er lettere å vedlikeholde, oppfordrer den forfattere til å bruke dem i stedet for skript der de kan. Eksempelet dens er elementet

`details`, som viser og skjuler en del av en side uten en eneste linje JavaScript. Standarden ber også forfattere sørge for at applikasjonene deres fortsatt fungerer på et enklere nivå når det ikke er skripting.

For britiske offentlige tjenester gjør manualen det til en regel: de bør kunne brukes med HTML alene, og JavaScript bør, der det brukes, bare forbedre HTML og CSS, slik at folk fortsatt kan bruke tjenesten når skriptet svikter. Regelen er skrevet for én regjerings tjenester, ikke for hele nettet, men begrunnelsen holder for ethvert nettsted der forfatterne ikke selv kan velge leserne sine.

Teamet bak siden vinner også noe. Når en forbedring går i stykker, mister leseren forbedringen, ikke oppgaven, så en feil i et skript blir en irritasjon og ikke et driftsavbrudd. En ny funksjon kan sendes ut til nettleserne som støtter den, uten en plan for dem som ikke gjør det, fordi de beholder siden de allerede hadde.

Slik ser det ut i praksis

HTML-standardens eget eksempel på noe som virker uten skript, er en tabell der kolonneoverskriftene sorterer den, og det er en god prøve på metoden. Bunnen er en tabell med vanlige lenker i overskriftene. Et klikk ber serveren om den samme tabellen sortert etter den kolonnen, og serveren sender den tilbake i den rekkefølgen. Det virker i alle nettlesere, med eller uten skript.

```
<table id="orders">
  <thead>
    <tr>
      <th><a href="?sort=date">Dato</a></th>
      <th><a href="?sort=total">Beløp</a></th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>2026-10-20</td>
      <td>120</td>
    </tr>
    <tr>
      <td>2026-10-24</td>
      <td>45</td>
    </tr>
  </tbody>
</table>
<script type="module" src="sort.mjs"></script>
```

Skriptet er forbedringen. Det finner de samme lenkene, avbryter navigasjonen og sorterer radene på selve siden, noe som sparer en tur fram og tilbake til serveren.

```
const table = document.querySelector("#orders");
const body = table.tBodies[0];
const collator = new Intl.Collator("nb", { numeric: true });

function sortBy(column) {
  const text = (row) => row.cells[column].textContent;
  const rows = [...body.rows];
  rows.sort((a, b) => collator.compare(text(a), text(b)));
  body.append(...rows);
}

for (const link of table.tHead.querySelectorAll("a")) {
  link.addEventListener("click", (event) => {
    event.preventDefault();
    sortBy(link.closest("th").cellIndex);
  });
}
```

Oppstår det en feil før en lytter er lagt til, virker den aktuelle lenken fortsatt. Hvis `sort.mjs` aldri kommer fram, gjør lenkene det samme som før. Hvis skriptet kaster en feil halvveis, bruker overskriftene som skriptet ikke rakk å legge til lyttere for, fortsatt serveren. En nettleser som ikke støtter modulskript, oppfatter `type="module"` som en skripttype den ikke kjenner, og ignorerer det, slik HTML-standarden beskriver. Et modulskript uten `async` kjøres når siden er ferdig parset, så tabellen finnes når skriptet leter etter den. Dette eksemplet håndterer ikke at en tilknyttet klikkbehandler kaster en feil etter at `preventDefault()` har avbrutt navigasjonen.

Prisen betales på serveren: bunnen krever et endepunkt som kan sortere, og de to måtene å sortere på må gi samme resultat. Det er metodens vanlige kostnad, betalt én gang. Den tiltenkte gevinsten er en side der hovedoppgaven fortsatt kan utføres med serveren som reserve; en tilknyttet hendelsesbehandler må bevare den muligheten når dens eget arbeid svikter.

Dette nettstedet gjør grunnlaget tydelig. Tilgjengelighetserklæringen beskriver komplette sider i HTML, CSS og SVG uten kjørbar JavaScript. Søkekoden er bevart, men slått av, så sidene ikke avhenger av et filter. Erklæringen nevner også en grense som står igjen: de små etikettene som vises når pekeren holdes over et

ikon eller ikonet får fokus, kan ikke lukkes med en tast. Det ville kreve nettleserkode nettstedet ikke leverer. Derfor oppfylles 1.4.13 Content on Hover or Focus bare delvis.

Hvor det går galt

En `noscript` -blokk brukes i stedet for en bunn. Elementet `noscript` viser bare innholdet sitt når skripting er slått av, og HTML-standarden sier det rett ut: «The noscript element is a blunt instrument.» Skripting kan være slått på og sidens skript likevel svikte, og da får leseren verken den skriptede siden eller reserveløsningen. Begynn med en side som virker uten skriptet, og la skriptet endre den, slik standardens eget eksempel gjør når skriptet legger et levende resultat til et skjema og fjerner skjemaets sendeknapp.

Én funksjon uten støtte stopper hele skriptet. Fordi JavaScript-laget ikke tåler feil, kan en enkelt bit nyere syntaks i en skriptpakke trekke alle forbedringene i den samme pakken med seg i fallet, også dem som ellers ville ha virket. Hold bunnen uavhengig av skriptet, så en feil bare koster forbedringen. Last inn moderne kode med `type="module"` , som eldre nettlesere hopper over, og hvis eldre nettlesere i det hele tatt skal ha en forbedret versjon, gi dem en separat klassisk pakke merket med `nomodule` , som nettlesere som støtter moduler, ikke kjører.

En ny selektor river gode regler med seg. En nettleser som møter en selektor den ikke kjenner, forkaster hele regelen, deklarasjoner og alt. Står en nyere pseudoklasse i en liste sammen med selektorer som alle nettlesere forstår, kan den derfor fjerne stiler fra nettlesere som ellers ville ha brukt dem. Gi en selektor som noen nettlesere kanskje ikke støtter, sin egen regel.

Reservestiler lekker inn i den forbedrede layouten. Bredder som er skrevet for nettlesere uten et nyere layoutsystem, kan kjempe mot det layoutsystemet i nettleserne som har det. CSS Conditional Rules Level 3 viser løsningen med en flex-layout og en blokk med `@supports not (display: flex)` for alternativet og bemerker at breddedeklarasjonene kan skade flex-layouten. Derfor hører de bare hjemme i stilene for nettlesere uten den.

Bilder velges av et skript. HTML-standarden påpeker at hvis en side ikke angir noe bilde i HTML-koden sin og overlater valget til et skript, lastes det ikke ned noe bilde i det hele tatt for lesere med skripting slått av, og hvis et skript bytter

ut et bilde som HTML-koden allerede har angitt, kan nettleseren starte to nedlastinger. Angi kandidatene i HTML med `srcset` og `sizes` eller med `picture`, og la nettleseren velge.

Atferd uten noe under. Champeon og Fincks liste over strategier ber om en reserve for all atferd: en ekte `href` bak hver klikkhåndterer og en sjekk på serveren bak hver sjekk i nettleseren. En kontroll som bare finnes som klikkhåndterer, eller et skjema som bare valideres i nettleseren, har ingen bunn i det hele tatt. Bruk ekte lenker og ekte skjemainnsendinger først, valider alltid på serveren, og se på validering i nettleseren som en bekvemmelighet som er lagt oppå.

Man tror det handler om gamle nettlesere. Et team som tror at bunnen er for gamle nettlesere, dropper den så snart de nettleserne er borte. GOV.UKs liste over årsaker handler om nettverk, blokkere, brannmurer og feilaktige oppdateringer, og dem møter dagens nettlesere også. Hold bunnen ved like som en vei som blir brukt og testet: last av og til inn siden med skript blokkert, og fullfør hovedoppgaven fra start til slutt.

Les mer

- Using progressive enhancement — veiledningen i GOV.UKs Service Manual, sist oppdatert 27. september 2024: de tre lagene, hvordan hvert av dem svikter, og hvorfor offentlige tjenester bør virke med HTML alene.
- HTML Standard: Scripting — elementet `script` med `type="module"` og `nomodule`, og elementet `noscript` med rådet om ikke å stole på det.
- CSS Conditional Rules Module Level 3 — W3Cs Candidate Recommendation Draft fra 15. august 2024, som definerer `@supports` med `not`, `and` og `or`.
- HTML Standard: Images — `srcset`, `sizes` og `picture`, og hvorfor det svikter lesere med skripting slått av når et skript velger bildet.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er progressive enhancement?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/progressive-enhancement/