Engineer

Progressive enhancement

at bygge sider, der virker i ren HTML, og så lægge mere på

Hvad det er

Progressive enhancement er en måde at bygge websider på i lag, hvor man begynder med det lag, der altid virker. Bunden er HTML, som rummer hele indholdet og klarer hver opgave på egen hånd: links fører et sted hen, formularer sendes til en server, og siden kan læses og bruges, uden at noget andet er indlæst. Stylesheets lægges ovenpå for at sætte siden op og gøre den rar at se på. Scripts kommer sidst og gør den hurtigere eller mere smidig at bruge. Hvert lag forbedrer det, der ligger under, og siden er ikke afhængig af noget lag over HTML'en for at gøre sit arbejde. GOV.UK's Service Manual koger idéen ned til at få siden til at virke med HTML først. På dansk kaldes det også progressiv forbedring.

Navnet stammer fra et foredrag af Steven Champeon og Nick Finck, *Inclusive Web Design For the Future with Progressive Enhancement*, som ifølge foredragets skriftlige udgave blev holdt den 11. marts 2003 på SXSW i Austin i Texas. Teksten skriver Progressive Enhancement med store forbogstaver og forkorter det til PE. Den stiller idéen op over for graceful degradation, den ældre vane med at bygge til den mest formående browser og håbe, at de andre stadig kunne nå indholdet. PE begynder ifølge teksten med et dokument, som de mindst formående enheder kan bruge, og lægger derefter præsentationen på som separat logik, så de enkle enheder ikke tynges af det, de moderne browsere får.

Forskellen mellem de to ligger i arbejdets retning. Graceful degradation tager udgangspunkt i hele oplevelsen og spørger, hvad der er tilbage, når dele af den mangler. Progressive enhancement tager udgangspunkt i det, der altid er der, og spørger, hvad der kan lægges til. I en nyere browser på en god forbindelse kan begge ende med den samme side. Forskellen viser sig, når noget fejler: en side, der er bygget nedefra, har stadig sin bund, mens en side, der er bygget oppefra, kun har det, forfatterne huskede at sørge for.

Det er en arbejdsmåde, ikke et bibliotek eller en funktion. Intet i HTML, CSS eller JavaScript hedder progressive enhancement, men standarderne giver plads til det. En browser springer en CSS-erklæring over, som den ikke forstår, og ignorerer et script, hvis type den ikke kender, og reglen `@supports` lader et stylesheet spørge, om en funktion findes, før det regner med den.

Brug begrebet i en beslutning

Brug »Hvad er progressive enhancement?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Brugerflader, der indlæses hurtigt, virker på en telefon og opfører sig ens i alle de browsere, jeres kunder nu bruger.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har styret full-stack-udviklingen af GIS-kort — full stack-kortløsning, fra database til browser

Har styret full-stack-udviklingen af GIS-kort Kontakt os →

Hvorfor det betyder noget

Lagene fejler på forskellige måder, og GOV.UK's Service Manual gør rede for hvordan. CSS-laget er nogenlunde tilgivende: en browser springer en erklæring over, som den ikke genkender, og fortsætter, men en selektor, den ikke kender, får den til at droppe hele reglen. JavaScript-laget tilgiver slet ikke. Når et script bruger syntaks eller et interface, som browseren mangler, kører resten af scriptet ikke. En side, der har brug for sit script til sin hovedopgave, er derfor kun så pålidelig som hvert eneste led mellem serveren og læseren.

Og der er mange led. Samme side opregner grunde til, at CSS og JavaScript kan lade være med at blive indlæst eller køre: en midlertidig netværksfejl, en adblocker, et CDN, der er nede, en DNS-fejl, en fejl i en browseropdatering, en firewall i firmaet eller derhjemme. Ingen af dem har noget med en gammel browser at gøre, og læseren kan ikke se, hvilken af dem det var. Det eneste, læseren ser, er en knap, der ikke gør noget. Manualen giver ingen tal for, hvor tit hver af dem sker; dens argument hviler på listen, ikke på en procentsats.

Nogle læsere slår også ting fra med vilje. Manualen skriver, at nogle brugere bevidst slår funktioner fra i deres browser, og at deres valg skal respekteres. HTML-standarden skriver, at mange brugere slår scripting fra, og af den grund, og fordi deklarative mekanismer ofte er lettere at vedligeholde, opfordrer den forfattere til at bruge dem i stedet for script, hvor de kan. Dens eksempel er

elementet `details` , som viser og skjuler et afsnit af en side uden en eneste linje JavaScript. Standarden beder også forfattere om at sørge for, at deres applikationer stadig fungerer på et enklere niveau, når der ikke er scripting.

For britiske offentlige tjenester gør manualen det til en regel: de bør kunne bruges med HTML alene, og JavaScript bør, hvor det bruges, kun forbedre HTML og CSS, så folk stadig kan bruge tjenesten, når scriptet fejler. Reglen er skrevet til én regerings tjenester, ikke til hele nettet, men begrundelsen holder for ethvert site, hvis læsere forfatterne ikke selv kan vælge.

Holdet bag siden vinder også noget. Når en forbedring går i stykker, mister læseren forbedringen, ikke opgaven, så en fejl i et script bliver en irritation og ikke et nedbrud. En ny funktion kan sendes ud til de browsere, der understøtter den, uden en plan for dem, der ikke gør, fordi de beholder den side, de allerede havde.

Sådan ser det ud i praksis

HTML-standardens eget eksempel på noget, der virker uden script, er en tabel, hvis kolonneoverskrifter sorterer den, og det er en god prøve på metoden. Bunden er en tabel med almindelige links i overskrifterne. Et klik beder serveren om den samme tabel sorteret efter den kolonne, og serveren sender den tilbage i den rækkefølge. Det virker i enhver browser, med eller uden script.

```
<table id="orders">
  <thead>
    <tr>
      <th><a href="?sort=date">Dato</a></th>
      <th><a href="?sort=total">Beløb</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>
```

Scriptet er forbedringen. Det finder de samme links, annullerer navigationen og sorterer rækkerne om på selve siden, hvilket sparer en tur frem og tilbage til serveren.

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

Opstår der en fejl, før en lytter er tilføjet, virker det pågældende link stadig. Hvis `sort.mjs` aldrig når frem, gør linkene det samme som før. Hvis scriptet kaster en fejl halvvejs, bruger de overskrifter, som det endnu ikke har tilføjet lyttere til, stadig serveren. En browser, der ikke understøtter modulscripts, opfatter `type="module"` som en scripttype, den ikke kender, og ignorerer den, som HTML-standarden beskriver. Et modulscript uden `async` køres, når siden er færdig med at blive parset, så tabellen findes, når scriptet leder efter den. Dette eksempel håndterer ikke, at en tilføjet klikhandler kaster en fejl, efter at `preventDefault()` har annulleret navigationen.

Prisen betales på serveren: bunden kræver et endpoint, der kan sortere, og de to måder at sortere på skal give samme resultat. Det er metodens sædvanlige omkostning, betalt én gang. Den tilsigtede gevinst er en side, hvis hovedopgave stadig kan løses via serveren som reserve; en tilføjet handler skal bevare den mulighed, hvis dens eget arbejde fejler.

Dette site gør sit udgangspunkt tydeligt. Tilgængelighedserklæringen beskriver komplette sider i HTML, CSS og SVG uden JavaScript til afvikling i browseren. Søgekoden er bevaret, men slået fra, så siden ikke afhænger af et filter. Erklæringen nævner også en tilbageværende grænse: de små etiketter, der vises

ved hover eller fokus på et ikon, kan ikke fjernes med en tast. Det ville kræve browserkode, som sitet ikke leverer. Derfor opfylder det kun delvist 1.4.13 Content on Hover or Focus.

Hvor det går galt

En `noscript` -blok træder i stedet for en bund. Elementet `noscript` viser kun sit indhold, når scripting er slået fra, og HTML-standarden siger det ligeud: »The noscript element is a blunt instrument.« Scripting kan være slået til, og sidens script kan alligevel fejle, og så får læseren hverken den scriptede side eller reserveløsningen. Begynd med en side, der virker uden scriptet, og lad scriptet ændre den, sådan som standardens eget eksempel gør, når dets script tilføjer et levende resultat til en formular og fjerner formularens send-knap.

Én funktion uden understøttelse stopper hele scriptet. Fordi JavaScript-laget ikke tåler fejl, kan et enkelt stykke nyere syntaks i et bundle trække alle forbedringer i samme bundle med sig i faldet, også dem, der ellers ville have virket. Hold bunden uafhængig af scriptet, så en fejl kun koster forbedringen. Indlæs moderne kode med `type="module"` , som ældre browsere springer over, og hvis ældre browsere overhovedet skal have en forbedret udgave, så giv dem et separat klassisk bundle med `nomodule` , som browsere, der understøtter moduler, ikke kører.

En ny selektor river gode regler med sig. En browser, der møder en selektor, den ikke kender, dropper hele reglen, erklæringer og det hele. Står en nyere pseudoklasse i en liste sammen med selektorer, som alle browsere forstår, kan den derfor fjerne styles fra browsere, der ellers ville have brugt dem. Giv en selektor, som nogle browsere måske ikke understøtter, sin egen regel.

Reservestyles siver ind i det forbedrede layout. Bredder, der er skrevet til browsere uden et nyere layoutsystem, kan kæmpe imod det layoutsystem i de browsere, der har det. CSS Conditional Rules Level 3 viser løsningen med et flex-layout og en blok med `@supports not (display: flex)` til alternativet og bemærker, at breddeerklæringerne kan skade flex-layoutet. Derfor hører de kun hjemme i stylene til browsere uden det.

Billeder vælges af et script. HTML-standarden påpeger, at hvis en side ikke angiver noget billede i sin markup og overlader valget til et script, hentes der slet intet billede til læsere med scripting slået fra, og hvis et script udskifter et

billede, som markuppen allerede har angivet, kan browseren starte to downloads. Angiv kandidaterne i HTML med `srcset` og `sizes` eller med `picture`, og lad browseren vælge.

Adfærd uden noget nedenunder. Champeon og Fincks liste over strategier beder om en reserve til al adfærd: et rigtigt `href` bag hver klikhandler og et tjek på serveren bag hvert tjek i browseren. En kontrol, der kun findes som klikhandler, eller en formular, der kun valideres i browseren, har ingen bund overhovedet. Brug rigtige links og rigtige formularindsendelser først, valider altid på serveren, og betragt validering i browseren som en bekvemmelighed, der er lagt ovenpå.

Man tror, det handler om gamle browsere. Et team, der tror, at bunden er til gamle browsere, dropper den, så snart de browsere er væk. GOV.UK's liste over årsager handler om netværk, blokeringer, firewalls og fejlbehæftede opdateringer, og dem møder nutidens browsere også. Hold bunden ved lige som en vej, der bliver brugt og testet: indlæs af og til siden med scripts blokeret, og gennemfør dens hovedopgave fra start til slut.

Læs mere

- Using progressive enhancement — vejledningen i GOV.UK's Service Manual, senest opdateret den 27. september 2024: de tre lag, hvordan hvert af dem fejler, 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 at stole på det.
- CSS Conditional Rules Module Level 3 — W3C's Candidate Recommendation Draft fra den 15. august 2024, som definerer `@supports` med `not`, `and` og `or`.
- HTML Standard: Images — `srcset`, `sizes` og `picture`, og hvorfor det svigter læsere med scripting slået fra, når et script vælger billedet.

Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er progressive enhancement?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/progressive-enhancement/