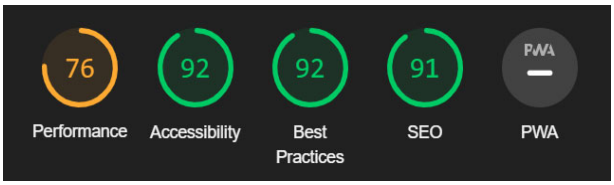
PROCESSANALYS

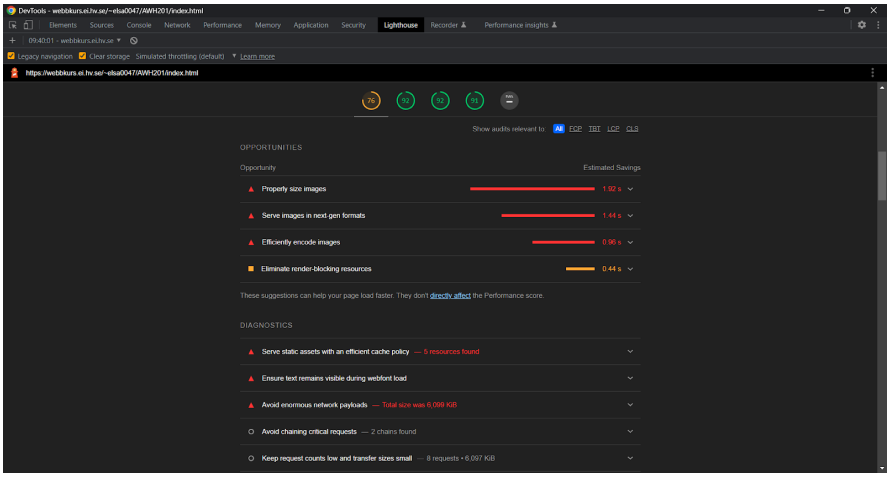
Elina Samuelsson

MODUL 1 - OPTIMERING



Hemsidan får “grönt ljus” i tre av fyra kategorier från start - enbart performance, alltså prestandan lyser gult. Samtliga kategorier kommer dock med rekommendationer på förbättringar, inte bara prestandan.

Prestanda



Majoriteten av optimeringen kring hemsidans prestanda gällde bilderna. Ingen av hemsidans bilder hade lämplig upplösning enligt Lighthouse-rapporten, som jämför den faktiska upplösningen med den upplösning den används på hemsidan (Chrome Developers, 2019a).

Google lämnar en mängd rekommendationer för att förbättra posterna gällande bilder, så som att lägga upp bilderna genom CMS, eller att använda olika plugin-tjänster. Jag har dock valt att manuellt ändra upplösning på mina bilder genom Photoshop, samt att konvertera dem från jpeg till webp eftersom man från Googles sida hävdar att ett ‘next-gen’-format erbjuder överlägsen kompression och kvalitet (Chrome Developers, 2019b).

Resultatet blir markant snabbare laddning av hemsidan, och vi skalar bort 1.9 sekunder

genom att bilderna nu är mindre och därför tar kortare tid att ladda. 1.4 + 0.9 sekunder sparas genom att bilderna nu kommer i webp-format.

Ytterligare en åtgärd jag rekommenderats är att att eliminera renderingsblockerande resurser. Här fanns enligt Lighthouse-rapporten cirka 0.4 sekunder bespara i desktopformat, och mer än så i mobilt läge.

Renderingsblockerande resurser är filer såsom script-, css- och html-filer som blockerar renderingen tills hämtningen av dem är över. Detta kan ha en direkt negativ inverkan på hemsidans Largest Contentful Paint (Monus, 2022).

För att minska påfrestningen främst på laddningen i mobilt format har jag på Monus (ibid) rekommendation delat min css i tre delar baserat på formatet den är ämnad till. I html-koden där jag länkar in min css har jag sedan använt mig av attributet media för att berätta vilka filer som är relevanta för vilken skärmupplösning.

Både min HTML- och CSS-kod är minifierad för att plocka bort alla tecken som anses onödvändiga. Koden blir mindre läsbar men filstorleken blir mindre, och tar därför mindre tid att ladda ner.

Tack vare de åtgärder som tagits besparar vi ytterligare 0.2 sekunder i desktop format, och har gått från över en sekunds möjlig besparing till 0.6 sekunder i mobilt format.

Ytterligare en åtgärder som rekommenderas av Monus (ibid) är att lägga till egna fonter lokalt snarare än att länka dem över nätet, att leta efter och ta bort oanvänd kod, och att senarelägga nedladdning av icke-kritisk CSS och Javascript.

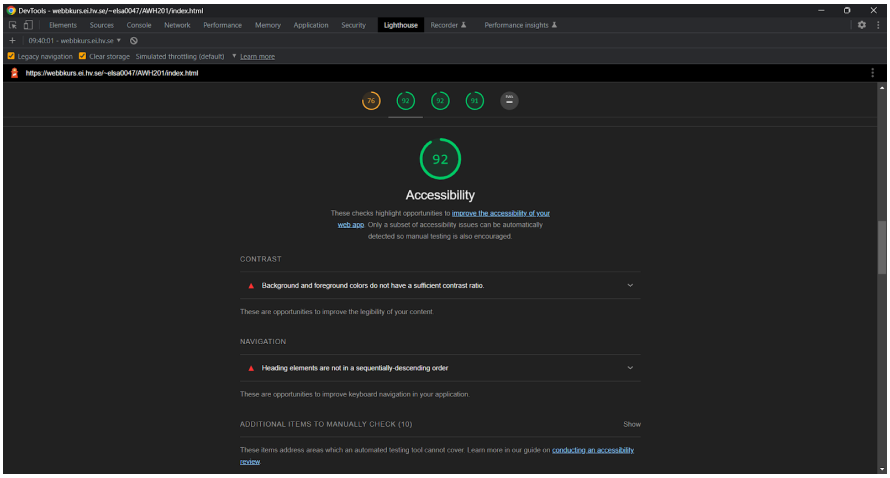
När man länkar in fonter kan man inte begränsa sig till de fontstorlekar och stilar man själv enbart använder sig av utan får ofta mer än man behöver. Om man istället förvarar fonterna lokalt har man full kontroll på all kod, och kan minifiera den vid behov. Här finner vi orsaken till att jag i mobilt format fortfarande har över en halv sekund kvar att bespara. Jag är dock redan nöjd med de besparingar jag gjort och faller väl inom godkända poäng även utan att korrigera användandet av fonten och låter därför detta vara som det är.

Att sålla bort oanvänd kod talar till stor del för sig själv. Oanvända tecken tar upp filutrymme och saktar ner nedladdningen. I hemsidans kod finns igen oanvänd CSS och jag har därför ingen tid att bespara här.

När det kommer till att senarelägga hämtning av filer hänvisas främst till CSS som används under hemsidans fold, och JavaScript som inte är kritisk för hemsidans

laddning. Här har jag redan gjort mycket genom att separera min CSS i tre filer och använda mig av media-attributet för att prioritera nedladdningarna. Man kan även separera koden i innehåll ovanför och under hemsidans fold, för att prioritera laddning av det som direkt syns när man öppnar hemsidan.

Tillgänglighet

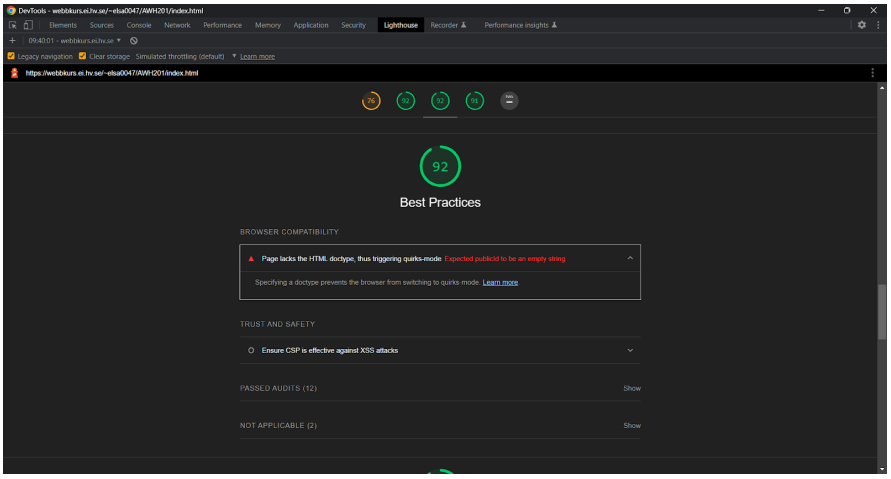


Gällande tillgängligheten framfördes två förbättringsmöjligheter; för låg kontrast mellan bakgrund och länktext, samt rubriker som inte följer logisk sekvens.

Länktexten har därför fått en mörkare färg för att skapa mer kontrast mot bakgrunden, och rubrikerna är nu taggade med en h2-tagg snarare än en h3-tagg som tidigare. Utseendet justeras nu genom ett uppdaterat stylesheet skrivet med OOCSS-metodik.

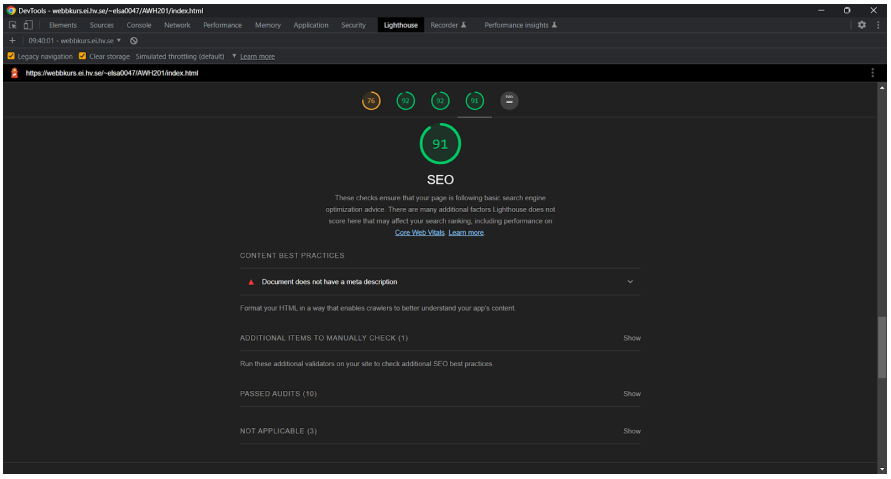
Tillgänglighet ger ingen tidsbesparing, men förbättrar upplevelsen av hemsidan för funktionshindrade personer.

Best practises



Under rubriken Best Practises fanns en varning om att dokumentet helt saknar doctype. Patrick Hulce (2019) rapporterar i en felsökningstråd på Google Chromes github-sida att detta beror på att Lighthouse inte är byggt med XHTML i åtanke. Att åtgärda denna varning är därför enkelt löst genom att byta doctype till den mer moderna HTML5.

SEO



Även SEO-rubriken lämnar ett mindre klagomål på att det saknas en hemsidebeskrivning i metataggarna. Beskrivningen är viktig i SEO-syfte eftersom det är den beskrivande text som ibland syns under rubriken i sökmotorresultat (Toonen, 2023).

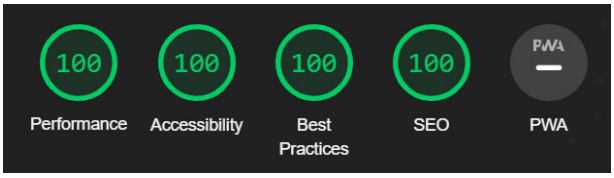
Adderandet av en beskrivande meta-tag ger ingen tidsmässig förbättring, men att ha en beskrivning - speciellt en genomtänkt och välformulerad beskrivning i rätt längd - kan locka fler nya besökare till hemsidan genom sökmotor-resultaten.

4

Slutresultat

Landningshastigheten är en viktig aspekt av användarupplevelsen på internet. Så lite som 0.1 sekunders fördröjning riskerar att minska omvandlingsfrekvensen i en webbshop med 7% och för varje två sekunder extra hemsidan tar att ladda ökar bortfallet av besökare med 103% (Cosper, 2019).

Hur fort hemsidan laddas är också en faktor i Googles rankningssystem vid sökningar. En sida som inte möter kriterierna landar lägre i sökresultaten och får därför inte samma trafik som en väloptimerad hemsida.



Redan från början konstaterade Lighthouse-rapporten att jag gjort mycket rätt, och med ett mindre antal korrigeringar når nu sidan full poäng i alla kategorier, vilket resulterar i en tillgänglig och väl optimerad hemsida som bjuder in, och välkomnar besökare att stanna.

Inga ändringar var tidskrävande, och passar väl in i en webbutvecklares flöde. Användandet av dem bör ses som standard i varje projekt.

MODUL 2 - CSSOPTIMERING

Metoder för att organisera CSS blir viktigare ju större projekt det jobbas med. Modulär CSS skapar ett system med bestämda regler som är enkla att hålla koll på, och för större team att jobba med.

Vilket man väljer verkar inte ha ett självklart svar på internet. Ian Holden (2021) skriver att alla system har positiva och negativa sidor, samt att vilket system man väljer inte spelar någon roll så länge det finns ett regelverk att hålla sig till innan CSS-koden börjar skrivas.

- BEM står för Block-Element-Modifier och lägger strikt fokus på namngivning och modulära komponenter för att skapa kod som är enkel att underhålla och återanvända (ibid).
- Scalable and Modular Architecture for CSS (SMACSS) framförhåller kategorisering baserat på funktion för att organisera koden, ofta i olika filer (Abuelgasim, 2021). Detta skapar kod som är lätt kan underhållas och byggas på över tid (Holden, 2021).
- OOCSS (Object Oriented CSS) separerar struktur och utseende, samt behållare från innehåll. Metoden skapar återanvändningsbar kod som är enkel att bygga på (ibid).

Mitt val föll på att använda OOCSS för att skriva om min hemsida. I mina ögon var alla tre alternativ lika i tänket och alla erbjuder i grunden samma slutresultat, om än med olika tillvägagångssätt. Jag håller därför med Holden om att det är svårt att argumentera för eller emot något system, och att vad som passar beror dels på projektet men också personlig smak.

Vissa, som till exempel Matt Stauffer (2014) kombinerar aspekter från samtliga system. Han menar då att om man blandar filsystemet från SMACSS med strukturen från OOCSS och namngivningen från BEM får man det bästa från varje system.

En kritik som lyfts med OOCSS är att man tvingas använda klassnamn som inte är semantiska. Semantiskt skriven CSS är den andra skolan av hur folk anser att CSS bör skrivas. Ofta menar man då att fokus bör ligga på innehållssemantiken (Strachan, 2014).

Man kan dock också vända på det, som Strachan (ibid) själv gör: OOCSS handlar om att bjuda in det visuellt semantiska. Att fästa ett visst utseende till ett innehåll gör koden

förlorar sin återanvändningsbarhet och påbyggningsbarhet.

Endera gör dock inte den andra obetydlig, utan man bör alltid sträva efter att ge så mycket information som möjligt i sin kod. Klassatributet är flexibelt att användas så som det behövs i situationen (ibid).

Ytterligare en nackdel som lyfts med samtliga av ovan metoder är att HTML-koden blir tjockare och mer oläslig (Arsenault, 2019). Mycket av CSS-ens funktionalitet flyttar från stilmallarna till HTML-taggarnas klassattribut.

Nelson Michael (2022) ställer BEM och SMACSS mot varandra i en artikel på LogRocket där slutsatsen är densamma som Holdens; båda har för och nackdelar men i slutändan beror valet på personlig preferens.

Han hävdar att BEM är långsammare att jobba med än SMACSS, i alla fall utan att blanda in användandet av en CSS preprocessor som Sass eller Less. Detta på grund av den strikta namngivningen som tvingar användaren att skriva ibland långa klassnamn (ibid).

Nackdelen för SMACSS sida är att med friare rekommendationer kring namngivning kan det vara svårare att greppa, speciellt som ny i ett existerande team med sin egen namnstandard (ibid).

Till ett projekt som detta, där det dels redan finns existerande kod, och som är väldigt litet ser jag få fördelar med att använda OOCSS. Det gör inte heller Ben Marshall (2012) som påpekar att OOCSS i små projekt saknar mening eftersom det inte finns ett behov av påbyggnadsbar kod. Han påpekar precis som Arsenault ovan att man flyttar bördan från CSS:en till HTML-koden.

Dessutom påpekar Marshall att OOCSS har en inlärningskurva som kan påverka tidsaspekten om man sedan tidigare inte har kännedom i ämnet.

REFERENSER

Abuelgasim, A. 2021. *A web designer's guide to CSS methodologies.* Tillgänglig: https://www.creativebloq.com/features/a-web-designers-guide-to-css-methodologies. [2023-02-10].

Arsenault, C. 2019. *OOCSS - The Future of Writing CSS.* Tillgänglig: https://www.keycdn.com/blog/oocss. [2023-02-10].

Chrome Developers. 2019a. *Properly size images.* Tillgänglig: https://developer.chrome.com/docs/lighthouse/performance/uses-responsive-images/. [2023-02-06].

Chrome Developers. 2019b. *Serve images in modern formats.* Tillgänglig: https://developer.chrome.com/docs/lighthouse/performance/uses-webp-images/. [2023-02-06].

Cosper, J. 2019. *How to Fix a Slow Website.* Tillgänglig: https://www.dreamhost.com/blog/how-to-fix-slow-website/. [2023-02-09].

Holden, I. 2021. *CSS Architecture (BEM, OOCSS, SMACSS & ACSS) and why we need it.* Tillgänglig: https://www.ianholden.co.uk/blog/css-architecture-bem-oocss-smacss-acss-and-why-we-need-it. [2023-02-10].

Hulce, P. 2019. *XHTML doctype.* Tillgänglig: https://github.com/GoogleChrome/lighthouse/issues/9660#issuecomment-530871237. [2023-02-08].

Marshall, B. 2012. *OOCSS - modular theming*. Tillgänglig: https://www.benmarshall.me/oocss-object-oriented-css/. [2023-02-11].

Michael, N. 2022. *BEM vs. SMACSS: Comparing CSS methodologies*. Tillgänglig: https://blog.logrocket.com/bem-vs-smacss-comparing-css-methodologies/. [2023-02-11].

Stauffer, M. 2014. *Organizing CSS: OOCSS, SMACSS, and BEM*. Tillgänglig: https://mattstauffer.com/blog/organizing-css-oocss-smacss-and-bem/. [2023-02-10].

Strachan, J. 2014. *In Defence of OOCSS.* Tillgänglig: https://medium.com/@jamiestrachan/in-defence-of-oocss-869632e56d8e. [2023-02-10].

Toonen, E. 2023. *How to create the right meta description*. Tillgänglig: https://yoast.com/meta-descriptions/. [2023-02-08].