Browser detectie veiligheid en compatibiliteit

Inleiding

Als je gaat programmeren in JavaScript, zal de code meestal gebruikt worden in websites en uitgevoerd worden op de browser van de eindgebruiker. Het is dan goed om rekening te houden met de enorme diversiteit aan apparaten, besturingssystemen en browsers die invloed hebben op hoe JavaScript wordt uitgevoerd. Het opvallendst is het verschil in schermformaat, bijvoorbeeld tussen een grote gaming-monitor en een mobiele telefoon. Maar ook de beschikbaarheid van functies en libraries kan verschillen. Gelukkig kun je er wel van uitgaan dat de basis, zoals het doen van een berekening, op alle platformen dezelfde uitkomst zal geven. Hiervoor zijn standaarden afgesproken.

Browser standaarden en website engines

Wanneer je internetfora afzoekt naar tips en trucs voor JavaScript zul je merken dat Internet Explorer 6 heel vaak genoemd wordt als uitzondering. 'Deze code werkt goed, maar op Internet Explorer 6 moet het net iets anders.' Gelukkig is Internet Explorer 6 ondertussen bijna uitgestorven, maar verschillen bestaan nog steeds.

Het verschil in de werking van de diverse browsers is ontstaan omdat de fabrikanten van de browsers moesten kiezen tussen algemene en specifieke functionaliteiten. De algemene eigenschappen waren vastgelegd in de afspraken over de Gecko Engine. Het bedrijf Mozilla, bekend van browser Firefox, probeerde daarmee een open platform te bieden voor alle browsers. Toch wilden andere leveranciers zoals Apple, Microsoft en Google daar nog zaken aan toevoegen waardoor de engines van Safari, Internet Explorer en Chrome toch allemaal weer net iets anders werken. Beschrijvingen van deze verschillen zijn op het internet wel te vinden, maar het maakt het toch lastig voor ontwikkelaars om code te schrijven die altijd (goed) werkt.

De (on)mogelijkheden van browserdetectie

Het liefst zou je als ontwikkelaar met je code kunnen vaststellen welke browser door de eindgebruiker wordt gebruikt om je website te bekijken. Je kunt dan codevariaties aanbieden zodat de code op meer platformen werkt. Helaas ben je als ontwikkelaar afhankelijk van de mogelijkheden die de browsers bieden, en dat is zowel beperkt als weinig betrouwbaar. Je moet er dus altijd rekening mee houden dat de informatie die een browser verstrekt nadelen heeft:

- Verkeerde identificatie van browsers.
- Onbekende of toekomstige browsers.
- Exclusief voor mobiele apparaten.

Je merkt al snel dat het onbegonnen werk is om iedere browser op alle platformen tevreden te stellen. De kans is dat je meer browserdetectiecode nodig hebt dan dat je eigen code lang is.

Het beste advies is dan ook een aantal regels aan te houden.

Tips voor het verhogen van browser compatibiliteit

- Werk vanuit standaarden en probeer platformspecifieke code te vervangen door algemene oplossingen.
- Test je code zelf op zo veel mogelijk platformen. Als je zeker wilt weten of iets er goed uitziet, zul je het zelf moeten testen. Ga er niet blind van uit dat browserdetectiecode altijd zal werken.
- Stel een minimumaantal platformen vast, afhankelijk van de doelgroep. Het maakt een groot verschil of je een gaming platform aan het maken bent voor wereldwijd gebruik, of een informatieplatform voor ontwikkelingslanden. De eerste groep zal hoogstwaarschijnlijk de modernste browsers gebruiken, terwijl de laatste groep nog vertrouwd op technologie van 10 jaar geleden.
- Vertrouw scripts die gebruikmaken van de user-agent string niet automatisch. Deze string is makkelijk aan te passen en geeft snel verouderde of verkeerde informatie.

Webbrowsers en scriptveiligheid

Omdat JavaScript op een apparaat van een eindgebruiker wordt uitgevoerd, is het goed om altijd rekening te houden met beperkingen die te maken hebben met de veiligheid van dat apparaat. Een computer in een openbare bibliotheek kan bijvoorbeeld beter zijn beveiligd dan een thuiscomputer. Het kan dus zo zijn dat JavaScript minder goed of zelfs helemaal niet werkt.

Door in je HTML code een <noscript> tag op te nemen, kan er een melding worden getoond wanneer JavaScript helemaal is uitgeschakeld.

Een andere beperking is het aanroepen van andere bronnen vanuit JavaScript. Bijvoorbeeld het lezen van JSON- of XML-bestanden voor databasegegevens of het importeren van code van een ander onderdeel van de website kan door een browser worden geblokkeerd.

Recentelijk zijn browsers kritischer geworden door alleen gegevens van exact dezelfde bron toe te staan. Dit heet de same origin policy.

Same origin policy en cross-origin resource sharing

Onder de belangrijkste browsers is het restrictiebeleid gevarieerd en in de loop der jaren strenger geworden. Met HTML5 hebben de meeste browserleveranciers overeenstemming bereikt over een 'same origin'-beleid waarmee een document alleen door een ander document kan navigeren naar 'ancestor'-pagina's, waaronder parent, sibling en child, maar niet naar pagina's van andere sites. De regels voor het concept van dezelfde oorsprong zijn ingewikkeld, maar in het algemeen kan pagina X de locatie van pagina Y alleen in een document wijzigen als:

- de locaties van X en Y dezelfde oorsprong hebben (d.w.z. hetzelfde protocol/poort en host) en op een bepaald niveau een venster delen van een ouder of grootouder
- Y een venster op het hoogste niveau is en X genest is binnen Y (direct kind, kleinkind enz.)
- Y een venster is dat is geopend met window.open, en X de locatie kan wijzigen van het venster dat Y opende (d.w.z. Y is een pop-up die is geopend door X, of Y is geopend door een pop-up die is geopend door X).

Over het algemeen moeten kwalificerende voorouderpagina's hetzelfde protocol/dezelfde poort en host gebruiken. Dus als we het volgende voorbeeld bekijken, mogen de volgende pagina's de locatie van pagina Y.htm in http://www.mijnwebsite.com niet wijzigen:

- https://www.mijnwebsite.com/secure_X.htm: (ander protocol/poort)
- http://intranet.mijnwebsite.com/X.htm: (verschillende host)
- http://mijnwebsite.com/X.htm: (andere host)
- ftp://mijnwebsite.com/X.htm: (ander protocol/poort, ook file://).

Dit browserbeveiligingsmodel is verder geëvolueerd dan het was ontworpen en er bestaan nog steeds inconsistenties tussen browsers. Je zult echter merken dat JavaScript een rol speelt in het beleid, omdat scriptrelaties (zoals pop-ups) de mogelijkheid van een pagina om door een document te navigeren, kunnen beïnvloeden.

Overtredingen van de same origin policy zijn zichtbaar in de browser als CORS (cross-origin resource sharing) errors.

CORS (cross-oigin resource sharing) errors

De meest eenvoudige manier om een CORS-overtreding te veroorzaken, is gebruik te maken van het file:// protocol. Als je met de browser een index.html opent op je eigen computer (dus niet via een server), gelden er beperkingen als je bijvoorbeeld vanuit een JavaScript in de HTML een ander bestand probeert te openen. Er ontstaat dan een conflict tussen het file:// protocol en het http:// protocol. Test je code dus altijd op een webserver.

Conclusie

Wanneer je werkt aan code voor een website is het altijd aan te raden te testen op meerdere platformen en browsers. Maak goede afspraken met de opdrachtgever over een selectie, want het is onmogelijk in alle omgevingen goed te testen.

Mocht er iets niet, of niet goed werken, dan heb je de keuze tussen het implementeren van uitzonderingen voor specifieke browsers, of een melding op te nemen dat die browser niet wordt ondersteund. Overleg dit altijd goed met je opdrachtgever voordat er te veel tijd wordt gestoken in de haast onmogelijke taak een website overal te laten werken.