Libraries & frameworks

Moderne webapplicaties functioneren heel anders; er worden server-sided geen pagina's in elkaar geknutseld en geserveerd, de applicatie – vaak aangeduid met **SPA (Single Page Application)** – bestaat feitelijk alleen uit JavaScript-code, JavaScript-modules, en HTML-templates en haalt de benodigde data pas op, op het moment dat de applicatie in de browser van de client geladen is. De techniek die daarvoor gebruikt wordt, noemen we **AJAX (Asynchronous JavaScript And XML)** en wordt later besproken. Alles speelt zich vervolgens af in de client, het tonen van de data, het navigeren tussen delen van de applicatie. Pas als data gewijzigd wordt, is er weer communicatie met de webserver – vak in de vorm van een web API, waarover later meer – om de gewijzigde data op te slaan. Dit gebeurt ook weer met AJAX en de data die verstuurd wordt, is in een specifiek format namelijk JSON (JavaScript Object Notation). Vroeger was dit overigens XML maar tegenwoordig wordt vooral JSON gebruikt omdat dit compacter is én eenvoudiger in JavaScript te interpreteren en te verwerken omdat het – het zit al in de naam – JavaScript Object-notatie is.

Asynchroon

Omdat er geen sprake is van page refreshes en nieuwe pagina's die na elke actie van de gebruiker van de webserver opgehaald worden, is er sprake van asynchroniteit tussen de processen op de client en die op de webserver. Een gebruiker kan ergens op klikken of data wijzigen, de data wordt vervolgens opgehaald of verstuurd naar de webserver maar omdat de User Interface van de SPA gewoon actief blijft (geen page refresh) kan de gebruiker ondertussen ergens ander op klikken of erger nog, iets opnieuw aanpassen. Dan lopen alle processen door elkaar en weet de applicatie op een gegeven moment niet meer wat de juiste data is. Ook kan een gebruiker niet zonder meer navigeren naar een vorige weergave, dit is immers geen pagina die opgeslagen is in de history van de browser.

Om dit allemaal mogelijk te maken en in goede banen te leiden hebben we allerlei technieken nodig zoals 'state management', 'data binding', 'change detection' en nieuwe functies van JavaScript zoals de fetch() API, promises en observables. Hier komen de moderne libraries en frameworks om de hoek kijken omdat de genoemde technieken op een elegante en consistente manier in deze oplossingen zijn verwerkt.

Library vs. framework

Een library is een verzameling 'predefined' scripts die een gebruiker in de HTML-pagina kan aanroepen en naar believen kan gebruiken. Een goed voorbeeld is jQuery, een populaire library die gebruikt wordt door eigenlijk alle bekende CMS-systemen. Als jQuery eenmaal geladen is, kan elke jQuery-functie gebruikt worden náást 'vanilla' JavaScript (zie volgend onderwerp voor een voorbeeld).

Een framework werkt heel anders en heeft een veel dwingerder karakter en daardoor een veel steilere leercurve. Een goed voorbeeld is Angular, een zeer robuust en krachtig framework. Het starten van een Angular-applicatie begint met het installeren van Node.Js en NPM op de lokale machine vervolgens gebeurt het meeste via de command line (terminal). De Angular Command Line Interface moet eerst geïnstalleerd worden, maar dan kan via deze interface eenvoudig een project aangemaakt worden. Eenmaal aangemaakt, kan dit project geopend worden in bijvoorbeeld VSC, maar dan wordt de verse

Angular ontwikkelaar geconfronteerd met een intimiderende hoeveelheid mappen en bestanden waarin men de weg moet leren vinden. De ontwikkelaar moet zich het werken met Angular helemaal eigen maken, het lijkt op het leren besturen van een ruimteschip terwijl je alleen maar je rijbewijs hebt. Een Angular-project bouwen lijkt in geen enkel opzicht op het bouwen van een traditionele JavaScript-applicatie.

Grote voordelen van het werken met libraries en frameworks zijn:

- Tijdsbesparing. Eenmaal vertrouwd met de manier van werken kunnen complexe applicaties ontwikkeld worden én getest in een fractie van de tijd die het zou kosten met vanilla JavaScript.
- Compatibiliteit. De moderne libraries en frameworks garanderen browsercompatibiliteit (uiteraard beperkt tot de versies die nog officieel door de leveranciers ondersteund worden. Oudere versie van Internet Explorer (voor versie 11) vallen daar bijvoorbeeld al niet meer onder). Dit scheelt enorm veel tijd in het testen, er hoeft niet meer in elke browser op elk platform getest te worden.
- Consistentie en kwaliteit. Vooral frameworks dwingen een consistente manier van werken af en dit komt de kwaliteit van de code ten goede. Een project kan zonder problemen door iemand anders overgenomen worden, de manier van werken is eenduidig en de opbouw van de applicatie snel te doorgronden, hoe complex ook.
- Unit testing en end-to-end (e2e) testing. Moderne frameworks zijn vaak voorzien van externe libraries (Mocha, Jasmine, Protractor etc.) om zowel componenten als de complete applicatie automatisch te kunnen testen. Testers en ontwikkelaars kunnen zo samenwerken aan foutloos werkende applicaties.
- Veiligheid. Frameworks als Angular zijn relatief veilig omdat het nagenoeg onmogelijk is onveilige code te genereren die gevoelig is voor code injection. Dit komt er tijdens het builden, runnen en testen direct uit. Dit wordt 'sanitization' genoemd.

Nadelen van libraries en frameworks zijn er ook:

- Libraries zijn groot (als in veel Megabytes). Als slechts een beperkt aantal scripts uit een library gebruikt worden, kan het verstandiger zijn de library niet te gebruiken en alles met vanilla JavaScript op te lossen, zeker omdat er steeds meer JavaScript-functies verschijnen die nieuwere en betere functionaliteit bieden.
- Libraries kunnen 'verouderen'. Veel ontwikkelaars zijn van mening dat jQuery zijn beste tijd heeft gehad en eigenlijk verouderd is. Maar als veel applicaties ontwikkeld zijn m.b.v. jQuery is het lastig van deze library af te komen.
- Het kan lastig zijn personeel te vinden dat ervaring heeft met de library of het framework dat u gebruikt.
- De performance van de applicatie kan onder het laden van meerdere frameworks en libraries te lijden hebben.

De populariteit van JavaScript libraries verandert in de loop van de tijd. Er was een tijd dat jQuery de meest gebruikte library was. Maar met de ontwikkeling van frameworks zoals Angular, React en Node is de wereld veranderd. U kunt bij het ontwikkelen van uw project elke library selecteren die uw doel dient en op lange termijn wordt ondersteund.

Houd bij het evalueren van libraries rekening met de volgende factoren:

- Welke libraries gebruikt uw winkel (d.w.z. team, bedrijf, organisatie)?
- Is de library robuust genoeg en heeft deze voldoende plug-ins om aan uw langetermijnbehoeften te voldoen?
- Hoe volwassen is de library? Als het relatief nieuw is, wacht u mogelijk om het te gebruiken totdat het volwassen is.

- Hoe gemakkelijk is het om de library te gebruiken? Kunt u binnen enkele uren werkende modellen maken van voorbeelden, of is de library tijdrovend en vervelend? Het hele doel van het gebruik van een library is om tijd en energie te besparen en zo kostbare ontwikkelingstijd te besparen.

Copyrightkwesties en JavaScript

De meeste mensen zijn bekend met het concept van auteursrechten, die het gebruik beperken van elk type inhoud dat u niet zelf hebt gemaakt, zonder uitdrukkelijke toestemming van de maker. Net als schrijven, kunst, muziek en afbeeldingen, wordt elk type code dat wordt gebruikt om een webpagina, script of applicatieprogramma te maken, beschouwd als iemands creatie en is daarom onderworpen aan de regels van het auteursrecht.

Copyright geeft de maker van een werk het recht om het gebruik van het werk te specificeren: of het kan worden gebruikt, hoe het kan worden gebruikt en of mensen die het willen gebruiken een royalty moeten betalen en/of de auteur de eer moeten geven. Maar hoeveel code vormt een 'werk' waarop auteursrecht kan rusten?

De elementen van een codetaal zijn niet meer auteursrechtelijk beschermd dan de woorden van de Engelse taal – iedereen is vrij om ze te gebruiken om respectievelijk te coderen, schrijven of spreken. Maar – vergelijkbaar met woorden – zodra stukjes code worden gecombineerd om bepaalde dingen te produceren, worden ze een werk dat eigendom is van de eigenaar. Voor code zijn de vereisten voor het in aanmerking komen van auteursrechten als volgt:

- De code moet een volledige functie vormen. U kunt geen JavaScript-fragment als uw eigen fragment claimen. Het moet een specifieke taak uitvoeren.
- Het script of programma moet een uniek werk zijn en mag geen combinatie zijn van eerder gebruikte JavaScript-functies in een enkele plug-in.
- Het werk moet inkomsten genereren.

Als aan deze vereisten is voldaan, kunnen andere ontwikkelaars het werk niet gebruiken zonder toestemming van de maker.

Gelukkig delen ontwikkelaars in de JavaScript-wereld graag. Het delen van code verbetert de ontwikkelingsgemeenschap door:

- Ontwikkeling sneller en gemakkelijker te maken voor iedereen.
- Programma's te openen voor opbouwende kritiek en verbetering.
- Goede code te verspreiden op het web in plaats van minder ervaren ontwikkelaars te dwingen slechte code te gebruiken.

Copyleft en copycenter

Copyleft en copycenter spelen op het woord copyright om alternatieve praktijken te beschrijven voor het delen van werken zoals scripting en programmeercode. De meeste, zo niet alle, code die in JavaScript-libraries wordt gevonden, gebruikt copyleft- of copycenter-licenties.

In tegenstelling tot het auteursrecht, dat anderen verbiedt het werk van een auteur te gebruiken of te distribueren, geeft een auteursplichtlicentie mensen algemene toestemming om het werk te gebruiken, te wijzigen en te verspreiden zonder royalty's te betalen. De belangrijkste vereiste is dat iedereen die dit doet, zijn variaties op het werk op dezelfde manier moet verspreiden: royalty vrij. Sommige makers hebben ook toeschrijving nodig; dat wil zeggen, als u hun code gebruikt in uw eigen afgeleide werken, geeft u een formele erkenning waarin de auteur van het origineel wordt vermeld.

Copyleft vereist ook dat de broncode die nodig is voor het reproduceren en wijzigen van het programma gratis ter beschikking wordt gesteld. De broncode bevat over het algemeen deze licentievoorwaarden en vermeldt de auteur (s) van het werk.

Enkele voorbeelden van copyleft-licenties zijn GNU GPL (GNU General Public License) en MIT (Massachusetts Institute of Technology). GPL was de eerste copyleft-licentie die op grote schaal werd gebruikt en blijft domineren als de meest gebruikte licentie voor vrije software. Klik hier voor meer informatie over de GNU GPL.

De makers van het copycenter bij BSD (Berkeley Software Distribution) maken een grapje dat in tegenstelling tot copyright (wat het gebruik beperkt) en copyleft (wat gratis gebruik mogelijk maakt), copycenter in het midden staat en iedereen toestaat om het naar het kopieercentrum te brengen en maak zoveel kopieën als je wilt. Een licentie voor een copycenter legt geen beperkingen op aan de manier waarop u uw afgeleide werkt distribueert of in licentie geeft.

Klik hier voor een lijst met gratis licenties en hun sjablonen.