Afstudeerplan

Informatie afstudeerder en gastbedrijf

Afstudeerblok: 2015-1.1 (start uiterlijk 9 februari 2015)
Startdatum uitvoering afstudeeropdracht: 9 februari 2015
Inleverdatum afstudeerdossier volgens jaarrooster: 5 juni 2015

Studentnummer: 11023449
Achternaam: dhr Gravekamp
Voorletters: T.N.
Roepnaam: Thomas
Adres: Dr.R.J.Fruinplantsoen 49
Postcode: 2552 LJ
Woonplaats: Den Haag
Telefoonnummer: 070 3976310
Mobiel nummer: 06 81211084
Privé emailadres: tomtom@ziggo.nl

Opleiding: Informatica
Locatie: Den Haag
Variant: voltijd

Naam studieloopbaanbegeleider: B.M. Derks
Naam begeleidend examinator: E.M. van Doorn
Naam tweede examinator: G.M. Tuk

Naam bedrijf: Liones
Afdeling bedrijf: FontoXML
Bezoekadres bedrijf: Polakweg 7
Postcode bezoekadres: 2288 GG
Postbusnummer:
Postcode postbusnummer:
Plaats: Rijswijk
Telefoon bedrijf: 070 3191923
Telefax bedrijf:
Internetsite bedrijf: http://www.liones.nl/ en http://www.fontoxml.com/

Achternaam opdrachtgever: dhr Willems
Voorletters opdrachtgever: Bert
Titulatuur opdrachtgever:
Functie opdrachtgever: Product Architect
Doorkiesnummer opdrachtgever: -
Email opdrachtgever: bert.willems@fontoxml.com

Achternaam bedrijfsmentor: dhr Middel
Voorletters bedrijfsmentor: Martin
Titulatuur bedrijfsmentor:
Functie bedrijfsmentor: Developer
Doorkiesnummer bedrijfsmentor:
Email bedrijfsmentor: martin.middel@fontoxml.com

Doorkiesnummer afstudeerder:
Functie afstudeerder (deeltijd/duaal):

Titel afstudeeropdracht:
Uitvoeren van onderzoek naar de performance van de rendering engines in browsers en voorstellen van verbetering bij FontoXML.

Opdrachtomschrijving

1. **Bedrijf**

FontoXML is een bedrijf dat een gelijknamig product ontwikkeld en verkoopt. Dit product is een gebruiksvriendelijke editor voor het maken en bewerken van gestructureerde content (XML). Daarnaast adviseert FontoXML partijen met uitgeefvraagstukken. FontoXML werkt onder andere voor Wolters Kluwer, Thieme Meulenhoff, de Verenigde Naties en het RILM.

FontoXML is begin 2014 afgesplitst van Liones, een internetbureau dat al 12 jaar actief is in de uitgeverij. Daardoor is FontoXML zelf een jong bedrijf.

Op dit moment werken er 10 full-time werknemers, een aantal stagair(e)s en afstudeerders bij FontoXML. Daarnaast werken er bij Liones nog eens 25 medewerkers.

Begin 2014 is gestart met de ontwikkeling van FontoXML .Versie 1 wordt op dit moment afgerond en uitgeleverd aan een viertal early adoptors. Binnen het team wordt met Scrum ontwikkeld.

2. **Probleemstelling**

Omdat FontoXML een high performance web-applicatie is, is het belangrijk dat deze snel reageert op acties van de gebruiker. Daarom moet er slim gebruik gemaakt worden van de resources die beschikbaar zijn binnen de browser. 60 FPS (Frames Per Second) is een must voor een goede en soepele gebruikerservaring. Het is niet de bedoeling dat een gebruiker moet wachten voordat een toetsaanslag op het scherm staat.

FontoXML kan op dit moment prima overweg met documenten van enkele tientallen pagina's. Bij meer pagina's begint de applicatie traag aan te voelen. Echter is er de behoefte ontstaan om complete boekwerken te openen die bestaan uit enkele honderden pagina's. Het openen van grote bestanden is een unique selling point voor FontoXML, omdat concurrerende partijen moeite hebben met het openen en bewerken van dergelijk grote bestanden.

Uit een aantal metingen die door het FontoXML team intern zijn uitgevoerd, is gebleken dat de rendering performance van de browser uiteindelijk de belangrijkste bottleneck is.

De verwachting is dat de optimalisaties van het renderen op een aantal vlakken liggen. De twee belangrijkste vlakken zijn de volgende:

- Het minimaliseren van de DOM (Document Object Model) in de browser.
  Het gaat hierbij bijvoorbeeld om viewport culling, wat inhoudt dat alleen het stuk HTML in de DOM opgenomen wordt, welke op dat moment zichtbaar is voor de gebruiker. Hierdoor hoeft de browser de rest van het bestand, wat op dat moment geopend is, niet te renderen.
- CSS stijlkenmerken die een grote negatieve impact hebben op het renderen van de pagina.

Aangezien er een aantal oplossingsrichtingen is, wil het FontoXML team laten onderzoeken welke optimalisatiestrategie de meeste snelheidswinst oplevert, zodat er gerichte optimalisaties uitgevoerd kunnen worden.

3. **Doelstelling van de afstudeeropdracht**

Het doel van deze afstudeeropdracht is het uitbrengen van advies over de meest effectieve optimalisaties voor de webapplicatie FontoXML op basis van een onderzoek naar deze optimalisaties. Daarnaast zal een een proof of concept gebouwd worden, waarmee wordt bewezen dat de geadviseerde optimalisatie daadwerkelijk een snelheidswinst oplevert tijdens het renderen van het document.

4. **Resultaat**

Aan het eind van de stageopdracht zal er een adviesrapport opgeleverd worden waarin een advies wordt uitgebracht over de meest nuttige optimalisaties. Op basis van dit advies zullen de optimalisaties die worden genoemd door FontoXML op de roadmap worden geplaatst. Uiteindelijk zullen deze optimalisaties na het afronden van de afstudeeropdracht worden geïmplementeerd door het team van FontoXML.

5. **Uit te voeren werkzaamheden, inclusief een globale fasering, mijlpalen en bijbehorende activiteiten**

Mijlpaal	Bijbehorende activiteiten	Aantal dagen
Plan van Aanpak	● Methoden selecteren ● Op te leveren producten inplannen ● Planning maken ● Opdracht zo nodig verder formuleren	3
Testrapport (Baseline Profile)	● Testmethode selecteren ● Benchmarks draaien met de bestaande situatie	5
Onderzoeksrapport	● Onderzoek naar viewport culling ● Onderzoek naar CSS stijlkenmerken ● Resultaten verwerken	20
Ontwerp proof of concept	● Ontwerpen structuur proof of concept	10
Bouw proof of concept	● Implementeren van optimalisaties en verbeteringen in een proof of concept	15
Aanvulling Testrapport	● Benchmarks draaien met de nieuwe situatie (proof of concept) ● Oude en nieuwe situatie vergelijken	7
Adviesrapport	● Advies op basis van voorgaande activiteiten formuleren en onderbouwen	10

6. **Op te leveren (tussen)producten**

- Plan van Aanpak
- Testrapport (Baseline Profile en vergelijking met nieuwe situatie)
- Onderzoeksrapport
- Ontwerp proof of concept
- Proof of concept
- Adviesrapport

7. **Te demonstreren competenties en wijze waarop**

1.1 Selecteren methoden, technieken en tools
Deze beroepstaak wil ik binnen deze opdracht invullen door te onderzoeken welke optimalisatietechnieken er beschikbaar zijn en hoeveel performancewinst deze kunnen opleveren. Het is de bedoeling dat ik de technieken met de meeste performancewinst selecteer, zodat ik hierover een advies kan uitbrengen.

3.2 Ontwerpen systeemdeel
Deze beroepstaak wil ik binnen deze opdracht invullen door het ontwerpen van een proof of concept waarmee kan worden aangetoond dat de optimalisatietechniek daadwerkelijk werkt. Hierbij moet in het ontwerp rekening gehouden worden met de bestaande applicatie.

3.3 Bouwen applicatie
Deze beroepstaak wil ik binnen deze opdracht invullen door het bouwen van een proof of concept waarmee kan worden aangetoond dat de optimalisatietechniek daadwerkelijk werkt.

3.5 Uitvoeren van en rapporteren over het testproces
Deze beroepstaak wil ik binnen deze opdracht invullen door de performance van de bestaande situatie te vergelijken met de nieuwe situatie (proof of concept). Hiervoor dient de performance van de bestaande en de nieuwe situatie (namelijk die van het proof of concept) getest te worden. Hiermee kan worden aangetoond dat de optimalisatietechniek daadwerkelijk performancewinst oplevert en deze daarmee voldoet aan de eisen die aan de performance van de applicatie gesteld worden.

FontoXML

Plan van aanpak

Bijlage B

Thomas Gravekamp
09-02-2015

Inhoud

Bedrijf en opdrachtsomschrijving

Bedrijf

FontoXML is een bedrijf dat een gelijknamig product ontwikkelt en verkoopt. Dit product is een gebruiksvriendelijke editor voor het maken en bewerken van gestructureerde content (XML). Daarnaast adviseert FontoXML partijen met uitgeefvraagstukken. FontoXML werkt onder andere voor Wolters Kluwer, Thieme Meulenhoff, de Verenigde naties en het RILM.

FontoXML is begin 2014 afgesplitst van Liones, een internetbureau dat al 12 jaar actief is in de uitgeverij. Daardoor is FontoXML zelf een jong bedrijf.

Op dit moment werken er 10 full-time werknemers, een aantal stagair(e)s en afstudeerders bij FontoXML. Daarnaast werken er bij Liones nog eens 25 medewerkers.

Begin 2014 is gestart met de ontwikkeling van FontoXML. Versie 1 wordt op dit moment afgerond en uitgeleverd aan een viertal early adoptors. Binnen het team wordt met Scrum ontwikkeld.

Probleemstelling

FontoXML is een high performance web-applicatie. Daarom is het belangrijk dat deze snel reageert op de acties van de gebruiker. Hierdoor moet er slim gebruik gemaakt worden van de resources die beschikbaar zijn binnen de browser. Om een soepele gebruikerservaring te bieden is 60 FPS (Frames Per Seconde) een must. Het is niet de bedoeling dat een gebruiker moet wachten totdat een toetsaanslag op het scherm staat.

FontoXML kan op dit moment prima overweg met documenten van enkele tientallen pagina's. Bij grotere aantallen pagina's begint de applicatie traag aan te voelen. Echter is er de behoefte ontstaan om complete boekwerken te openen die bestaan uit enkele honderden pagina's. Het openen van grote documenten is een unique selling point voor FontoXML, omdat concurrerende partijen moeite hebben met het openen en bewerken van dergelijk grote documenten.

Doelstelling

Het doel van deze afstudeeropdracht is het uitbrengen van advies over de meest effectieve optimalisaties voor de web-applicatie FontoXML op basis van een onderzoek naar deze optimalisaties. Daarnaast zal een proof of concept gebouwd worden, waarmee wordt bewezen dat de geadviseerde optimalisatie daadwerkelijk een snelheidswinst oplevert tijdens het gebruik van FontoXML. Dit proof of concept is een onderdeel van het onderzoek wat uitgevoerd zal worden.

Aan het eind van de afstudeeropdracht zal een adviesrapport opgeleverd worden waarin een advies zal worden uitgebracht over de meest effectieve optimalisaties. Op basis van dit advies zullen de optimalisaties die worden geadviseerd door het team van FontoXML op de roadmap worden geplaatst. Uiteindelijk zullen deze optimalisaties na het afronden van de afstudeeropdracht worden geïmplenteerd door het team van FontoXML.

Scope

Project scope

Om het project tot een succesvol einde te brengen moeten de volgende activiteiten uitgevoerd worden:

Activiteit	**Specificatie**
Bepalen baseline profile (testrapport)	● Bepalen hoe de snelheid van FontoXML betrouwbaar gemeten kan worden ● Benchmarks draaien met de bestaande situatie
Uitvoeren onderzoek	● Opstellen van een hoofdvraag en een deelvraag ● Uitvoeren van vooronderzoek en vergaren van bronnen ● Uitvoeren literatuuronderzoek ● Verwerken bevindingen literatuuronderzoek ● Uitvoeren experimenten voor het beantwoorden deelvragen ● Conclusie trekken om de hoofdvraag te kunnen beantwoorden
Ontwerp proof of concept	● Door middel van verschillende diagrammen in kaart brengen hoe het proof of concept gebouwd zal worden
Bouw proof of concept	● Bouwen van het proof of concept op basis van de opgestelde ontwerpen
Opstellen testrapport	● Benchmarks draaien met het prototype ● Vergelijken van benchmarks
Opstellen adviesrapport	● Opstellen adviesrapport ● Opstellen advies over de te implementeren optimalisaties

Product scope

De product scope zal worden bepaald door de uitkomst van het onderzoek.

Afbakening opdracht

- Het onderzoek zal zich richten op optimalisaties welke betrekking hebben op het optimaliseren van HTML of CSS.
- Het proof of concept hoeft geen andere functionaliteit te bevatten dan de optimalisaties die uit het onderzoek zijn gekomen.
- De kwaliteit van het proof of concept hoeft niet van productieniveau te zijn.

Risicofactoren

- **Het onderzoek kan niet worden afgerond binnen de geplande tijd vanwege de complexiteit van het onderwerp.**
  Dit is een risico omdat het project moeilijk af te ronden is met een incompleet onderzoek. Dit risico zal worden beperkt door het afbakenen van het onderzoek en het maken van een planning voor het onderzoek. Wanneer het onderzoek toch langer gaat duren dan gepland, moet het onderzoek minder uitgebreid afgerond worden. Hierbij moet rekening gehouden worden met de tot dan toe vergaarde informatie.
- **De te implementeren optimalisaties blijken te complex voor een proof of concept.**
  Dit is een risico omdat er twee keer performancetests uitgevoerd dienen te worden. Een keer aan het begin van dit project en een keer aan het eind van dit project. Wanneer een optimalisatie te complex blijkt om de kunnen implementeren in de geplande tijd, kan een tweede keer tests niet uitgevoerd worden. Hierdoor kan de nieuwe situatie niet vergeleken worden met de bestaande situatie.
- **Het blijkt dat er geen literatuur beschikbaar is over het onderwerp.**
  Dit is een risico voor de uitvoer van het onderzoek. Dit risico was niet te beperken omdat de hoeveelheid literatuur vanzelfsprekend niet te beïnvloedden is. Als blijkt dat er geen literatuur beschikbaar is over het onderwerp, moet het onderzoek uitgevoerd worden aan de hand van kleine tests om te bepalen met welke optimalisatiestrategie de meeste performancewinst oplevert.

Projectorganisatie

Werken aan het project vindt plaats op het kantoor van Liones/FontoXML. Het project zal worden uitgevoerd vanaf 09-02-2015 tot 05-06-2015 en duurt 17 weken.

Er is gekozen om te werken met een op SCRUM gebaseerde softwareontwikkelmethode.

Projectleden

Naam	**06-nummer**	**Email**
Thomas Gravekamp	06 81 21 10 84	thomas.gravekamp@liones.nl

Team

Naam	**Email**
Jan Benedictus	jan.benedictus@liones.nl
Bert Willems (opdrachtgever)	bert.willems@liones.nl
Stef Busking	stef.busking@liones.nl
Vincent Smedinga	vincent.smedinga@liones.nl
Thomas Brekelmans	thomas.brekelmans@liones.nl
Wybe Minnebo	wybe.minnebo@liones.nl
Youri Bosselaar	youri.bosselaar@liones.nl
Michael de Vreugd	michael.de.vreugd@liones.nl
Martin Middel (bedrijfsbegeleider)	martin.middel@liones.nl

Verslaglegging

Tijdens het afstuderen wordt aan de Haagse Hogeschool verslag gelegd door middel van een bedrijfsbezoek, een voortgangsverslag en het afstudeerdossier.

Gedurende het onderzoek zal er wekelijks een gesprek zijn met Bert Willems en Martin Middel. In dit gesprek worden de uitgevoerde werkzaamheden besproken en kan er feedback gegeven worden op de uitgevoerde werkzaamheden.

Planning

Globale planning

	Product	Activiteiten	Week
1	Plan van aanpak	● Methoden selecteren ● Op te leveren producten inplannen ● Planning maken ● Opdracht zo nodig verder formuleren	1
2	Testrapport (baseline profile)	● Testmethode selecteren ● Benchmarks draaien met de bestaande situatie	1-2
3	Onderzoeksrapport	● Onderzoek naar viewport culling ● Onderzoek naar CSS stijlkenmerken ● Resultaten verwerken	2-6
4	Ontwerp proof of concept	● Ontwerpen structuur proof of concept	6-8
5	Bouw proof of concept	● Implementeren van optimalisaties en verbeteringen in een proof of concept	8-14
6	Testrapport	● Benchmarks draaien met de nieuwe situatie (proof of concept)	14-15
7	Adviesrapport	● Advies op basis van voorgaande activiteiten en producten forumleren en onderbouwen	15-16

Detailplanning

Vanaf week 2 zal er elke week één dag worden besteed aan het schrijven van het afstudeerverslag.

	Product	**Uitvoerdatum**	**Duur**
1	Plan van aanpak	09-02-2015 t/m 11-02-2015	3 dagen
2	Baseline profile - Testmethode bepalen - Benchmarks oude situatie	12-02-2015 t/m 19-02-2015	7 dagen - 5 dagen - 2 dagen
3	Onderzoeksrapport - Vooronderzoek - Literatuuronderzoek - Verwerken bevindingen - Conclusie	20-02-2015 t/m 26-03-2015	18 dagen - 4 dagen - 8 dagen - 4 dagen - 2 dagen
4	Ontwerp proof of concept	27-03-2015 t/m 03-04-2015	5 dagen
5	Bouw proof of concept	06-04-2015 t/m 08-05-2015	20 dagen
6	Testrapport - Benchmarks nieuwe situatie - Oude en nieuwe situatie vergelijken	11-05-2015 t/m 22-05-2015	8 dagen - 3 dagen - 5 dagen
7	Adviesrapport	25-02-2015 t/m 05-06-2015	9 dagen

Mijlpaalproducten

In de volgende subhoofdstukken worden de mijlpaalproducten van dit project beschreven.

Plan van aanpak

Het plan van aanpak moet in combinatie met de planning houvast bieden tijdens de uitvoer van het project. In het plan van aanpak wordt de opdracht nader beschreven en uitgewerkt. Tevens wordt in het plan van aanpak vastgelegd welke methoden en technieken tijdens de opdracht gebruikt gaan worden.

Testrapport

In het testrapport wordt beschreven hoe de snelheid van de applicatie gemeten kan worden. Daarnaast zullen de resultaten van de tests met de bestaande situatie en de resultaten van de tests met het proof of concept worden opgenomen in het testrapport.

Onderzoeksrapport

In het onderzoeksrapport wordt het proces van het onderzoek en de onderzoeksresultaten beschreven.

Ontwerp proof of concept

Het ontwerp voor het proof of concept wordt bepaald hoe de in het onderzoek gevonden optimalisaties toegepast moeten worden in het proof of concept. Hierbij moet rekening gehouden worden met de bestaande architectuur en de eigenschappen van de optimalisaties.

Proof of concept

De bedoeling van het proof of concept is om aan te tonen dat de optimalisaties die gevonden zijn tijdens het onderzoek daadwerkelijk effect hebben op de snelheid van de applicatie. Het proof of concept wordt gebruikt om te testen of de toegepaste optimalisaties het beoogde effect hebben op de snelheid van de applicatie.

Adviesrapport

In het adviesrapport wordt een advies geformuleerd op basis van de voorgaande activiteiten en producten die daaruit voortgekomen zijn.

FontoXML

Testrapport

Bijlage C

Thomas Gravekamp
12-02-2015

Inhoud

Inleiding

Het doel van dit testrapport is het beschrijven van de gebruikte testmethode waarmee de performance van de applicatie FontoXML is bepaald. Daarnaast zijn ook de resultaten van de performancetests met de applicatie en het proof of concept in dit rapport opgenomen.

De testresultaten die samen de baseline profile vormen, zijn als basis gebruikt voor het onderzoek naar optimalisaties binnen FontoXML. Als onderdeel van dit onderzoek is ook een proof of concept gebouwd waarin een in het onderzoek gevonden optimalisatie geïmplementeerd is. De performancetests zijn ook uitgevoerd met het proof of concept om te bepalen of de geïmplementeerde optimalisatie het gewenste effect heeft.

Testmethode

Het doel van de performancetests is het bepalen van de baseline profile van de applicatie. Deze baseline profile is later gebruikt om te vergelijken met de resultaten van de performancetests met het proof of concept. Hiermee kan bepaald worden of de in het proof of concept geïmplementeerde optimalisatie het gewenste effect heeft.

Software

Omdat FontoXML in de browser draait is de performance van de applicatie binnen de browser gemeten. In overleg met het ontwikkelteam van FontoXML is ervoor gekozen deze performancetests uit te voeren in Google Chrome. Daarnaast heeft Google Chrome, vergeleken met andere browsers, de meest uitgebreide profiler.

Met behulp van de ingebouwde profiler van Google Chrome is het mogelijk de prestaties van een webapplicatie te meten. De profiler kan meten hoe lang de browser bezig is met het uitvoeren van scripts, het opnieuw renderen van een webpagina en de browser niets aan het doen is. De informatie van deze profiler is gebruikt voor het bepalen van de performance van de applicatie.

Om de tests reproduceerbaar te maken, zijn de tests uitgevoerd in een virtuele machine. Op deze machine is Windows 8.1 geïnstalleerd om later de tests mogelijk te kunnen herhalen in Internet Explorer en Firefox.

Om invoer van een gebruiker te simuleren is er gebruik gemaakt van AutoHotkey. Dit programma kan op constante snelheid toetsaanslagen genereren waardoor elke tests met dezelfde snelheid uitgevoerd kan worden. Dit is belangrijk om de tests te kunnen vergelijken en reproduceerbaar te maken.

Testapplicaties

De applicaties waarin de performancetests zijn uitgevoerd zijn de retail editie van FontoXML en de HTML en CSS versie van de retail editie. De retail editie van FontoXML bevat alleen de standaard DITA elementen.

De HTML en CSS versie van de retail editie van FontoXML bestaat uit de HTML en CSS verantwoordelijk voor het opbouwen en stylen van het editorgedeelte van de applicatie. Hiermee kan de invloed van de browser op de performance van de applicatie gemeten worden zonder ruis van de scripts van de applicatie en de garbage collector van de browser zelf.

Testdocumenten

De retail editie van FontoXML bevat een standaard document waarmee de applicatie uitgeprobeerd kan worden. Dit bestand zelf heeft een beperkte omvang en veroorzaakt geen merkbaar performanceverlies tijdens het bewerken van het document in de applicatie. Dit document wordt wel meegenomen in de performancetests als vergelijkingsmateriaal.

Naast het standaard document zijn twee andere documenten gemaakt. Deze documenten moeten tijdens het bewerken in de applicatie voor een merkbaar performanceverlies zorgen. De resultaten van de performancetests met deze twee documenten moeten duidelijkheid scheppen over de bron van het performanceverlies.

Het eerste document is een document met 600 topic-elementen. Elke tien topics worden de volgende elementen herhaald om de gemiddelde inhoud van een document te simuleren:

- Topic 1: 5 standaard paragrafen
- Topic 2: 5 standaard paragrafen met terms
- Topic 3: 5 standaard paragrafen met **bold** woorden
- Topic 4: 5 standaard paragrafen met *italics* woorden
- Topic 5: 5 standaard paragrafen
- Topic 6: standaard paragraaf + 5 subtopics met een standaard paragraaf
- Topic 7: standaard paragraaf + tabel
- Topic 8: standaard paragraaf + unordered list
- Topic 9: standaard paragraaf + ordered list
- Topic 10: standaard paragraaf

Het tweede document is een document met tien geneste topic-elementen. Hierdoor ontstaat een diepe HTML- en XML-DOM. De performancetests met dit document moeten inzicht geven in de performance tijdens het werken met een diep genest document.

Testgevallen

Om de performance van de applicatie te meten zijn een aantal testgevallen opgesteld. Deze testgevallen zijn per testdocument uitgevoerd. Het invoeren van de tekst is met 300 aanslagen per minuut uitgevoerd. Deze snelheid kan door typisten makkelijk behaald worden.

Testgeval zin 1: Invoeren zin aan het begin van de eerste paragraaf
Tijdens de het uitvoeren van de tests met dit testgeval is er een zin aan het begin van de eerste paragraaf in het document ingevoerd. Dit testgeval en testgeval zin 2 moesten samen inzicht geven in de performance tijdens het invoeren van een zin en het eventuele verschil van de performance tussen beide testgevallen.

De zin welke gebruikt is voor dit testgeval en testgeval zin 2: *Lorem ipsum dolor sit amet, consectetur adipiscing elit.*

Testgeval zin 2: Invoeren zin aan het eind van de eerste paragraaf
Zie testgeval zin 1.

Testgeval paragraaf 1: Invoeren paragraaf na de eerste paragraaf
Tijdens het uitvoeren van de tests met dit testgeval is er een paragraaf na de eerste paragraaf van het document ingevoerd. Dit testgeval en testgeval paragraaf 2 moesten samen inzicht geven in de performance tijdens het invoeren van een paragraaf aan het begin van het document en aan het eind van het document. Daarbij moest ook duidelijk worden wat het eventuele verschil in performance tussen beide testgevallen zou zitten.

De paragraaf welke gebruikt is voor dit testgeval, testgeval paragraaf 2 en testgeval paragraaf 3: *Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas sed mi ante. Aliquam id turpis vestibulum, imperdiet arcu vel, faucibus sapien. Sed ultricies laoreet egestas. Nunc orci tellus, semper vel commodo vitae, blandit vel ante. Vestibulum id eros eu urna varius cursus et eu mauris.*

Testgeval paragraaf 2: Invoeren paragraaf aan het eind van het document
Zie testgeval paragraaf 1.

Testgeval paragraaf 3: Invoeren paragraaf na de eerste paragraaf met 600 aanslagen per minuut
Dit testgeval moest inzicht geven in het verschil tussen het invoeren van tekst met 300 aanslagen per minuute en het invoeren van tekst met 600 aanslagen per minuut.

Metingen bestaande situatie (Baseline profile)

In dit hoofdstuk zijn de resultaten van de performancetests voor de baseline profile opgenomen. Deze resultaten vormen samen de baseline profile.

Testgeval zin 1: Invoeren zin aan het begin van een paragraaf

Document	Onderdeel	Retail editor	HTML editor
Standaard	idle	70 %	98 %
	scripts	22 %	0 %
	garbage collector	2 %	0 %
	program	6 %	2 %
Diep	idle	64 %	96 %
	scripts	25 %	0 %
	garbage collector	5 %	0 %
	program	6 %	4 %
Groot	idle	10 %	51 %
	scripts	49 %	0 %
	garbage collector	6 %	0 %
	program	35 %	49 %

Testgeval zin 2: Invoeren tekst aan het eind van een paragraaf

Document	Onderdeel	Retail editor	HTML editor
Standaard	idle	73 %	98 %
	scripts	21 %	0 %
	garbage collector	2 %	0 %
	program	4 %	2 %
Diep	idle	47 %	97 %
	scripts	39 %	0 %
	garbage collector	8 %	0 %
	program	6 %	3 %
Groot	idle	13 %	55 %
	scripts	52 %	0 %
	garbage collector	7 %	0 %
	program	28 %	45 %

Testgeval paragraaf 1: Invoeren paragraaf na de eerste paragraaf

Document	Onderdeel	Retail editor	HTML editor
Standaard	idle	67 %	98 %
	scripts	26 %	0 %
	garbage collector	2 %	0 %
	program	5 %	2 %
Diep	idle	58 %	96 %
	scripts	31 %	0 %
	garbage collector	5 %	0 %
	program	6 %	4 %
Groot	idle	3 %	39 %
	scripts	67 %	0 %
	garbage collector	9 %	0 %
	program	21 %	61 %

Testgeval paragraaf 2: Invoeren paragraaf aan het eind van het document

Document	Onderdeel	Retail editor	HTML editor
Standaard	idle	63 %	99 %
	scripts	29 %	0 %
	garbage collector	3 %	0 %
	program	5 %	1 %
Diep	idle	57 %	96 %
	scripts	31 %	0 %
	garbage collector	5 %	0 %
	program	7 %	4 %
Groot	idle	41 %	93 %
	scripts	35 %	0 %
	garbage collector	9 %	0 %
	program	15 %	7 %

Testgeval paragraaf 3: Invoeren paragraaf na de eerste paragraaf met 600 APM

Document	Onderdeel	Retail editor	HTML editor
Standaard	idle	57 %	97 %
	scripts	32 %	0 %
	garbage collector	3 %	0 %
	program	8 %	3 %
Diep	idle	44 %	93 %
	scripts	40 %	0 %
	garbage collector	5 %	0 %
	program	10 %	7 %
Groot	idle	5 %	11 %
	scripts	79 %	0 %
	garbage collector	9 %	0 %
	program	6 %	89 %

Metingen nieuwe situatie

In dit hoofdstuk worden de resultaten van de tests met het proof of concept beschreven. De tests met het proof of concept zijn alleen uitgevoerd met het standaard en het grote document omdat de culling op geneste elementen nog niet af was.

Testgeval zin 1: Invoeren zin aan het begin van een paragraaf

Standaard	idle	98 %
	scripts	0 %
	garbage collector	0 %
	program	2 %
Groot	idle	98 %
	scripts	0 %
	garbage collector	0 %
	program	2 %

Testgeval zin 2: Invoeren tekst aan het eind van een paragraaf

Standaard	idle	98 %
	scripts	0 %
	garbage collector	0 %
	program	2 %
Groot	idle	98 %
	scripts	0 %
	garbage collector	0 %
	program	2 %

Testgeval paragraaf 1: Invoeren paragraaf na de eerste paragraaf

Standaard	idle	98 %
	scripts	0 %
	garbage collector	0 %
	program	2 %
Groot	idle	98 %
	scripts	0 %
	garbage collector	0 %
	program	2 %

Testgeval paragraaf 2: Invoeren paragraaf aan het eind van het document

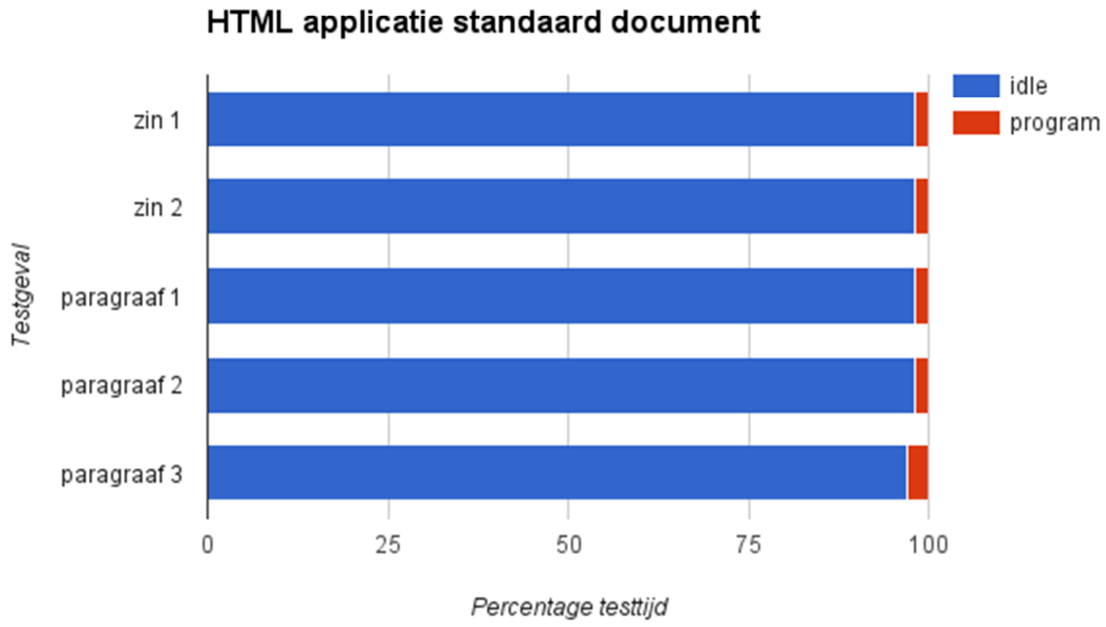
Standaard	idle	98 %
	scripts	0 %
	garbage collector	0 %
	program	2 %
Groot	idle	98 %
	scripts	0 %
	garbage collector	0 %
	program	2 %

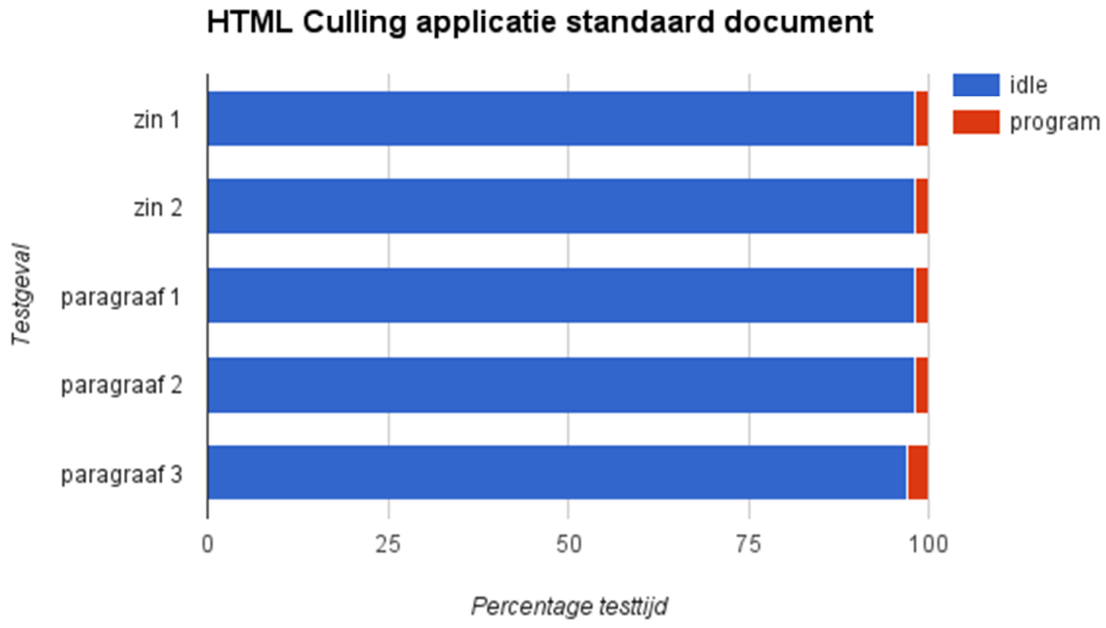
Testgeval paragraaf 3: Invoeren paragraaf na de eerste paragraaf met 600 APM

Standaard	idle	97 %
	scripts	0 %
	garbage collector	0 %
	program	3 %
Groot	idle	97 %
	scripts	0 %
	garbage collector	0 %
	program	3 %

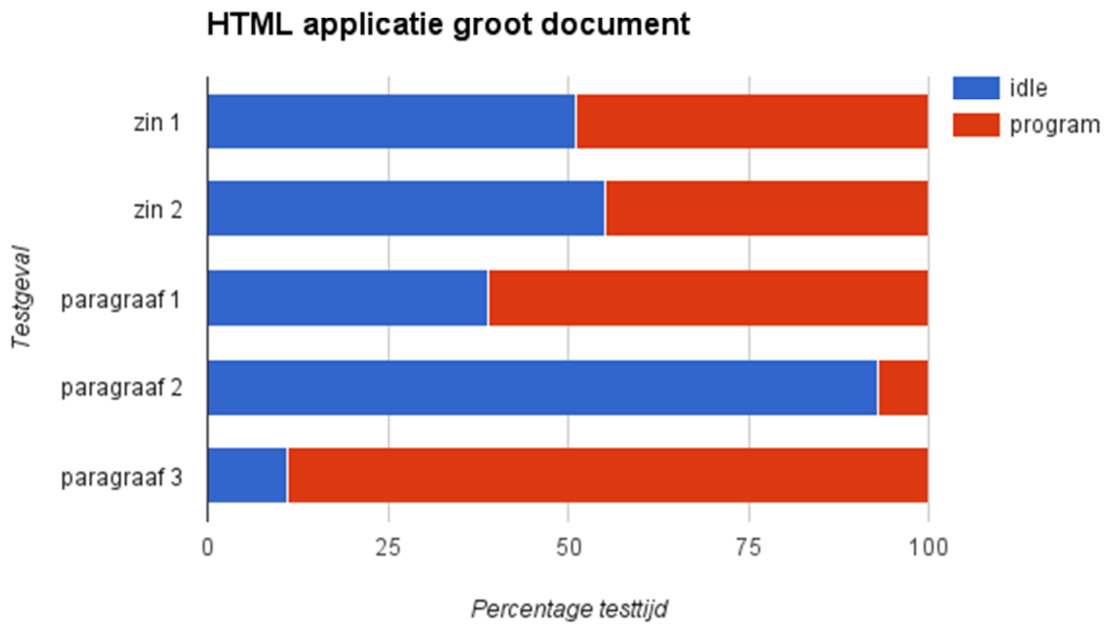
Vergelijking

Tussen de HTML applicatie en de HTML Culling applicatie zat tijdens de tests met het standaard document geen verschil in performance. De performance van de applicaties was in beide gevallen hetzelfde. Dit is te zien in onderstaande diagrammen.

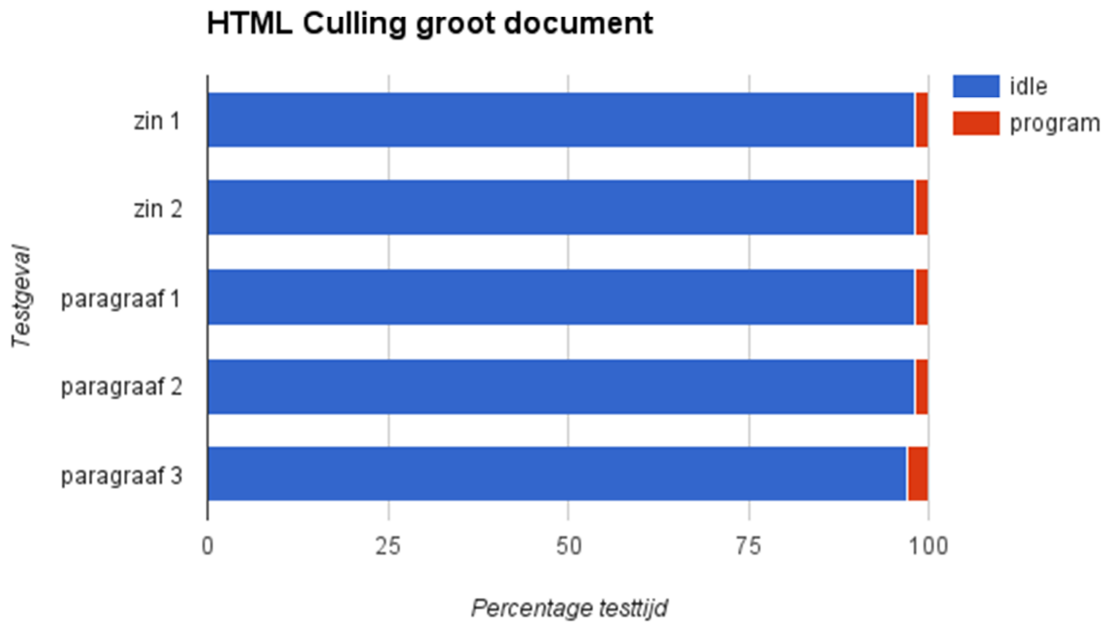




Tussen de HTML applicatie en de HTML Culling applicatie zat tijdens de tests met het grote document een groot verschil. Tijdens de tests met het grote document in de HTML editor was de browser een groot deel van de tijd bezig met het renderen van de webpagina. Dit is terug te zien in de resultaten van de performancetests in het onderstaande diagram.



De resultaten van de tests met het grote document in de HTML Culling editor waren hetzelfde als de resultaten van de tests met het standaard document in de HTML en HTML Culling editor. Hieraan is te zien dat het cullen van HTML-elementen een positief effect heeft op de performance van de editor.



FontoXML

Onderzoeksplan

Bijlage D

Thomas Gravekamp
12-03-2015

Inhoud

Aanleiding

De aanleiding van dit onderzoek is het performanceprobleem wat optreedt tijdens het bewerken van grote documenten in FontoXML. Het performanceprobleem treedt op in de vorm van een merkbare vertraging tussen een toetsaanslag en het verschijnen van het ingevoerde teken op het scherm.

Probleemstelling

De probleemstelling van dit onderzoek luidt als volgt: "Welke optimalisaties op het gebied van het renderen van een webapplicatie hebben een positief effect op de performance van FontoXML?".

Doelstelling

Het doel van dit onderzoek is een inzicht geven in de mogelijke optimalisatiestrategieën die een positief effect kunnen hebben op de performance van FontoXML. De uitkomst van dit onderzoek zal worden ondersteund door een proof of concept en zal daarnaast worden gebruikt om een advies te formuleren over de uit te voeren optimalisaties.

Vooronderzoek

Uit het vooronderzoek bleek dat er meerdere technieken beschikbaar zijn voor het optimaliseren van een webpagina of een uitgebreidere webapplicatie zoals FontoXML. Deze technieken hebben invloed op verschillende categorieën van een webpagina of webapplicatie. De grootte van deze invloed is afhankelijk van de opbouw van de webpagina of webapplicatie en het categorie waarop de optimalisatie invloed heeft.

De technieken voor het optimaliseren van een webpagina of webapplicatie zijn in te delen in een aantal categorieën. Deze categorieën zijn:

- Netwerk & webserver
- HTML DOM
- CSS
- Javascript

Het optimaliseren van het netwerkgebruik en de snelheid van de webserver vallen buiten de scope van dit onderzoek. Optimalisaties die betrekking hebben op Javascript vallen ook buiten de scope van dit onderzoek.

Optimalisaties die betrekking hebben op de HTML en de CSS van de pagina vallen wel binnen de scope van dit onderzoek. Het onderzoek zal zich dan ook voornamelijk richten op optimalisatietechnieken die te maken hebben met HTML en CSS. Dit betekent niet dat Javascript niet gebruikt zal worden om deze optimalisaties te gebruiken in het proof of concept.

Onderzoeksontwerp

In dit hoofdstuk wordt omschreven welke hoofdvraag (probleemstelling) en bijbehorende deelvragen tijdens dit onderzoek aan bod komen en wat het nut van deze vragen is.

Afbakening

Dit onderzoek zal zich alleen richten op het optimaliseren van de HTML DOM en CSS. De snelheid van het netwerk, de snelheid van de webserver en de snelheid van de aanwezige scripts vallen buiten de scope van dit onderzoek. Optimalisaties met betrekking tot HTML DOM en CSS kunnen wel met behulp van Javascript geïmplementeerd worden in het proof of concept.

Hoofdvraag

De hoofdvraag van dit onderzoek luidt: "Welke optimalisaties op het gebied van het renderen van een webapplicatie hebben een positief effect op de performance van FontoXML?". Deze vraag zal aan de hand van de antwoorden op de deelvragen beantwoord kunnen worden.

Om de hoofdvraag van dit onderzoek te beantwoorden zijn de volgende deelvragen opgesteld:

- **Deelvraag 1: Waardoor wordt het performanceverlies in FontoXML momenteel veroorzaakt?**
  Deze deelvraag wordt gesteld om te inventariseren wat momenteel de oorzaak van het performanceverlies is.
- **Deelvraag 2: Welke optimalisatiestrategieën kunnen worden toegepast op het gebied van HTML en CSS?**
  Deze deelvraag moet inzicht geven in de optimalisatiestrategieën die toegepast kunnen worden op het gebied van HTML en CSS.
- **Deelvraag 3: Welke optimalisatiestrategieën zullen (de meeste) performancewinst opleveren?**
  Deze deelvraag wordt gesteld om te bepalen welke optimalisatiestrategieën een performancewinst op zullen leveren. Dit wordt gedaan aan de hand van een proof of concept.

Planning

Hieronder is de planning te zien van het onderzoek. Voor het onderzoek zijn in totaal achttien dagen ingepland in de globale planning. Deze achttien dagen zij als volgt onderverdeeld:

Activiteit	**Duur**
Vooronderzoek + onderzoeksplan	6 dagen
Deelvraag 1	2 dagen
Deelvraag 2	5 dagen
Deelvraag 3	3 dagen
Conclusie	2 dagen

FontoXML

Onderzoeksrapport

Bijlage E

Thomas Gravekamp
18-03-2015

Inhoud

Inleiding

Dit onderzoeksrapport is opgesteld om inzicht te geven in het onderzoek wat gedaan is naar de optimalisatiemogelijkheden binnen FontoXML. In dit rapport is beschreven hoe het onderzoek is uitgevoerd en wat de resultaten waren. Dit rapport wordt afgesloten met een conclusie.

Hoofdvraag

De hoofdvraag van dit onderzoek luidt: "Welke optimalisaties op het gebied van het renderen van een webapplicatie hebben een positief effect op de performance van FontoXML?".

Deelvragen

De deelvragen die binnen dit onderzoek beantwoord zullen worden zijn:

- Deelvraag 1: Waardoor wordt het performanceverlies in FontoXML momenteel veroorzaakt?
- Deelvraag 2: Welke optimalisatiestrategieën kunnen worden toegepast op het gebied van HTML en CSS?
- Deelvraag 3: Welke optimalisatiestrategieën zullen (de meeste) performancewinst opleveren?

Deelvragen

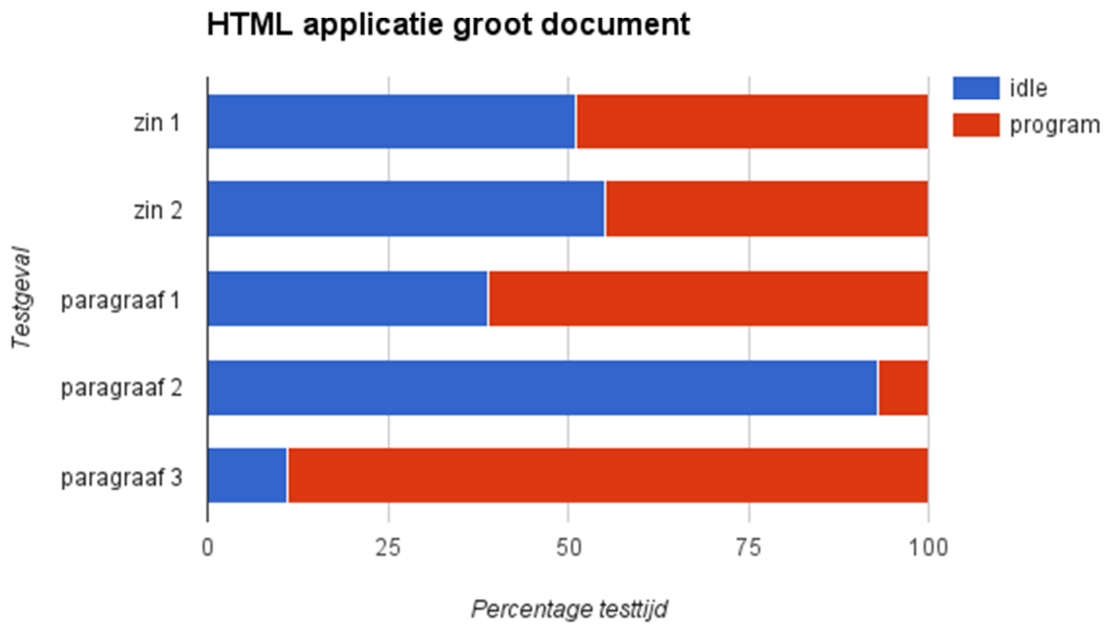
Deelvraag 1: Waardoor wordt het performanceverlies in FontoXML momenteel veroorzaakt?

Om te bepalen waardoor het performanceverlies binnen FontoXML wordt veroorzaakt, is gekeken naar de testresultaten welke samen de baseline profile vormen. Uit deze resultaten kan worden afgeleid welke onderdelen binnen de applicatie het performanceverlies veroorzaken.

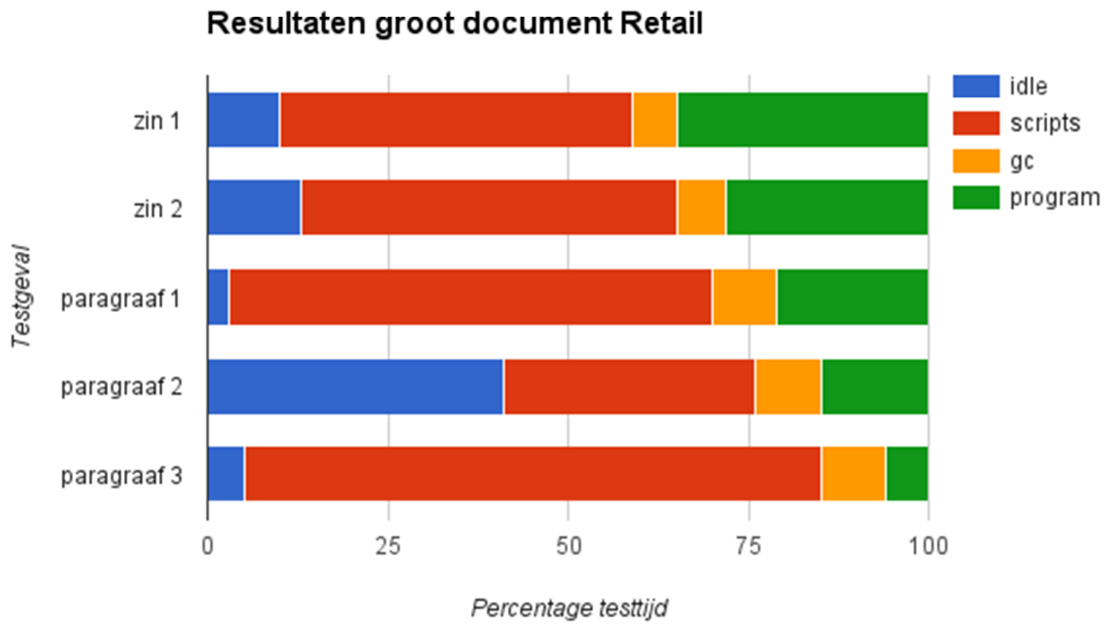
De performancetests voor het bepalen van de baseline profile zijn uitgevoerd in de retail editie van FontoXML en de HTML versie van diezelfde applicatie. Uit de tests met de HTML versie van de applicatie bleek dat de browser een groot deel van de totale tijd gebruikt voor het reflowen en repainten van de pagina wanneer tekst wordt ingevoerd.

Uit de resultaten van de performancetests met de retail editie van de applicatie bleek dat de scripts van de applicatie de meeste tijd in namen. Naast de scripts van de applicatie was de browser een groot deel van de tijd bezig met het renderen van de webpagina.

Het grootste performanceverlies was te meten tijdens de tests met het grote document. Tijdens vier van de vijf tests met het grote document in de HTML editor was de browser meer dan 50% van de totale tijd bezig met het renderen van de webpagina. De enige test waarbij de browser minder dan 50% van de totale tijd bezig was met het renderen van de webpagina was tijdens het invoeren van een paragraaf aan het eind van het document. Tijdens deze test was de browser slechts 7,41% van de totale tijd bezig met het renderen van de webpagina. Dit is goed te zien in het volgende diagram:



Tijdens de performancetests met de retail editie van FontoXML was de browser vooral bezig met het uitvoeren van de aanwezige scripts. De rest van de testtijd werd grotendeels ingevuld met het renderen van de applicatie. Dit is te zien in het onderstaande diagram:



Conclusie

De hoeveelheid tijd die de browser besteed aan het renderen van de webpagina tijdens het testen met de HTML versie van de applicatie wijst erop dat het renderen van de pagina na elke toetsaanslag een zware operatie is voor de browser. Daarnaast is duidelijk te zien dat het invoeren van tekst aan het begin van een document zwaarder is dan aan het eind van een document.

Naast het performanceverlies door het renderen van de pagina treed er in de volledige applicatie ook performanceverlies op door de scripts die op de achtergrond draaien. Deze scripts vallen echter buiten de scope van dit onderzoek.

Deelvraag 2: Welke optimalisatiestrategieën kunnen worden toegepast op het gebied van HTML en CSS?

Zoals al uit het vooronderzoek bleek, zijn de optimalisatiestrategieën voor het optimaliseren van webpagina's in te delen in vier categorieën: HTML, CSS, Javascript en netwerk.

De optimalisatiestrategieën die gericht zijn op Javascript en netwerk vallen buiten de scope van dit onderzoek. Welk kan het voorkomen dat Javascript gebruikt zal (moeten) worden om een optimalisatie op het gebied van HTML of CSS toe te kunnen passen.

Vanuit het ontwikkelteam van FontoXML waren voor de aanvang van dit onderzoek al twee mogelijke oplossingsrichtingen opgesteld:

- Het minimaliseren van de DOM (Document Object Model) in de browser door middel van HTML DOM Culling
- CSS stijlkenmerken met een grote negatieve impact op het renderen van de webpagina vermijden

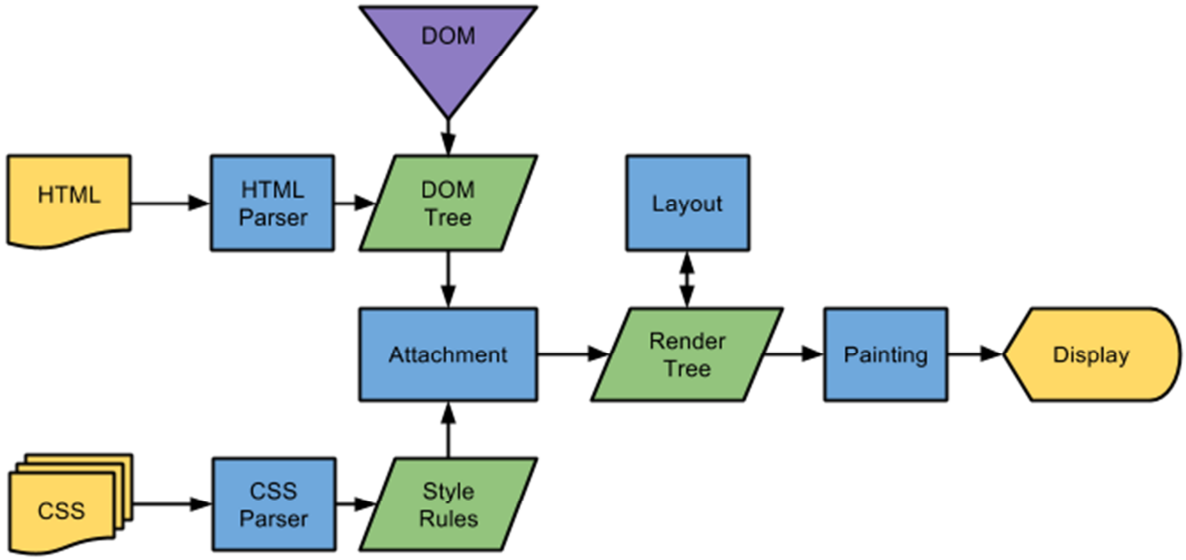
Om deze deelvraag goed te kunnen beantwoorden zijn de volgende analyse vragen gesteld en beantwoord:

- Hoe gaat de browser om met HTML- en CSS-bestanden?
- Hoe kan de HTML van een webpagina worden geoptimaliseerd?
- Hoe kan de CSS van een webpagina worden geoptimaliseerd?

Hoe gaat de browser om met HTML- en CSS-bestand?

De manier waarop de browser HTML- en CSS-bestanden verwerkt en intern representeert geeft inzicht in welke optimalisaties toe te passen zijn en waarom deze optimalisaties een positief effect kunnen hebben op de performance van de applicatie.

In onderstaand stroomdiagram is te zien hoe de browser de HTML- en CSS-bestanden intern representeert. Ook is in dit stroomdiagram te zien welke acties de browser uit moet voeren om van de bronbestanden tot een uitvoer op scherm te komen (Garsiel, 2011) (Stefanov, 2009).

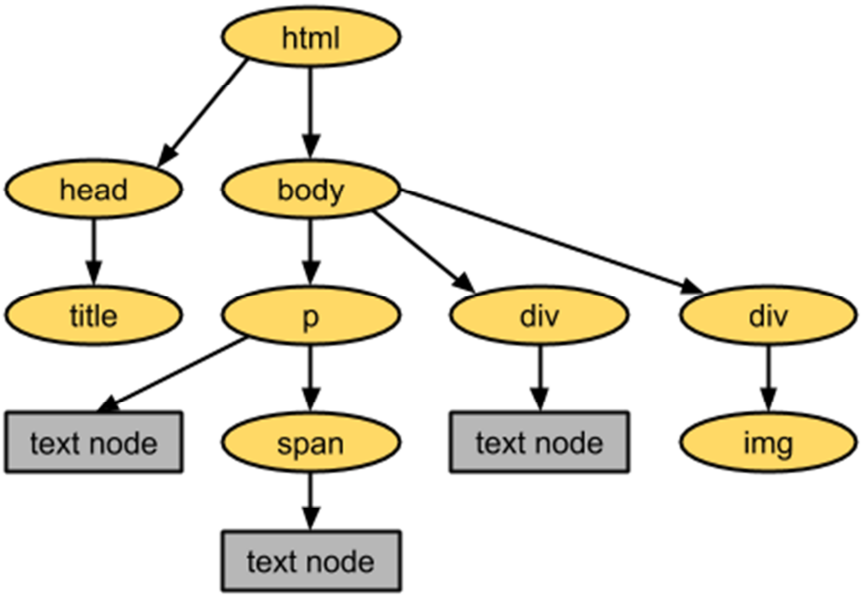


Na het ontvangen van het HTML-bestand begint de browser met het opbouwen van de webpagina. Eerst wordt het HTML-bestand door de HTML parser verwerkt en omgezet in een DOM Tree.

Voorbeeld HTML-bestand

```
<html>
  <head>
    <title>HTML webpagina</title>
  </head>
  <body>
    <p>Lorem <span>ipsum</span> dolor sit amet elit.</p>
    <div style="display: none">
      Onzichtbare tekst
    </div>
    <div>
      <img src="" />
    </div>
  </body>
</html>
```

Hieronder staat een voorbeeld van een DOM Tree zoals een browser deze opbouwt op basis van het bovenstaande bestand (Grigorik, 2014):

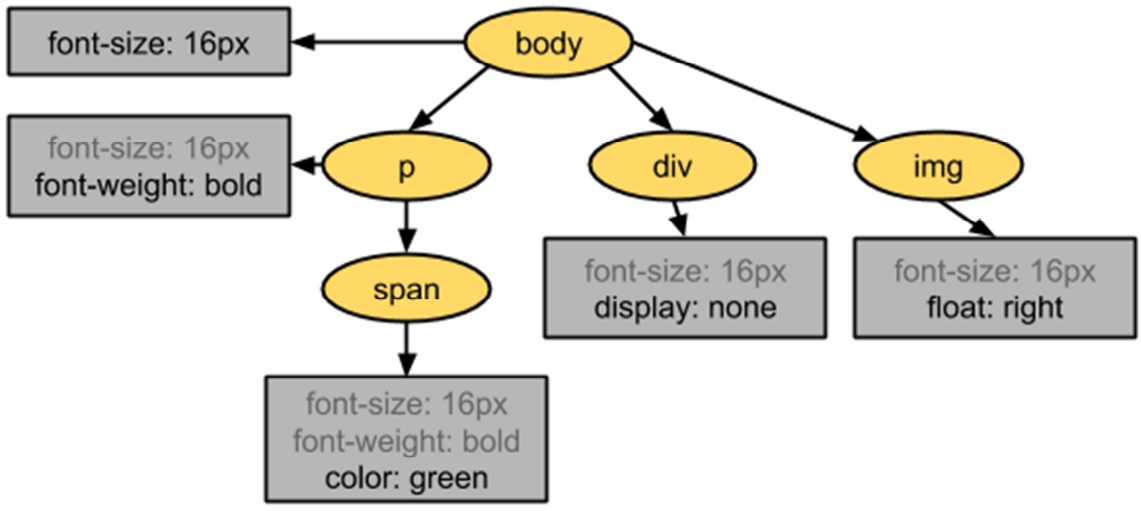


Na het verwerken van het HTML-bestand wordt het CSS-bestand verwerkt. Dit bestand wordt net als het HTML-bestand verwerkt door een parser. De CSS Parser wordt omgezet in een CSSOM Tree (Grigorik, 2014).

Voorbeeld CSS-bestand

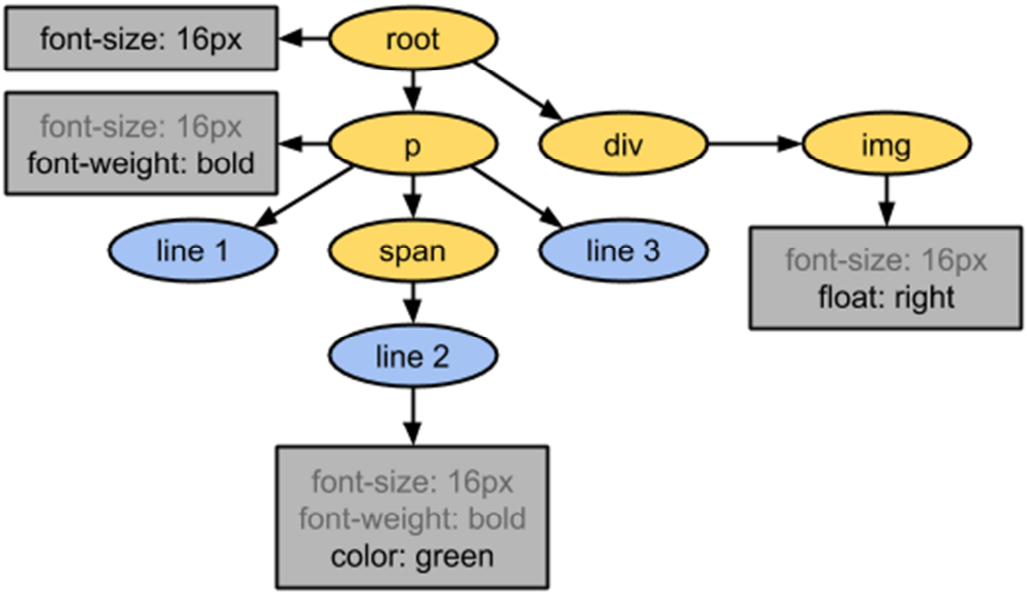
```
body {
  font-size: 16px;
}

p {
  font-weight: bold;
}

p span {
  color: green;
}

img {
  float: right;
}
```

Hieronder staat een voorbeeld van een CSSOM Tree zoals een browser deze opbouwt op basis van het bovenstaande voorbeeldbestand:



Nadat de DOM Tree en de CSSOM Tree zijn opgebouwd worden deze twee Trees samengevoegd tot een Render Tree (Grigorik, 2014). De Render Tree bevat alleen de elementen die volgens de toegepaste stijlen zichtbaar zijn. Hierbij horen daarom ook de elementen die buiten de viewport van de browser liggen.

Hieronder staat een voorbeeld van een Render Tree op basis van de voorgaande DOM en CSSOM Trees:



Na het opbouwen van de Render Tree wordt de layout van de pagina berekend. Hierna wordt de pagina gepaint en wordt het resultaat op het scherm getoond.

Wanneer de layout van de pagina veranderd wordt er een reflow uitgevoerd. Tijdens een reflow berekent de browser de layout van de pagina opnieuw. Een reflow is noodzakelijk wanneer er bijvoorbeeld een nieuw element ingevoerd wordt. Hierdoor moet de positie en eventueel de grootte van andere elementen opnieuw berekend worden.

Een reflow wordt veroorzaakt door een van de volgende acties (Sullivan, 2009) (Lew, 2012):

- Formaat van het venster wijzigen
- Font veranderen
- Toevoegen of verwijderen van een stylesheet
- Wijzigingen aan de inhoud, bijvoorbeeld een gebruiker die typt in een input element
- Activeren van CSS pseudo classes zoals :hover
- Wijzigen class-attrubuut
- Wijzigen van een element in de DOM
- Berekenen van offsetWidth, offsetHeight of getComputedStyle
- Toevoegen van een property aan het style-attribuut
- Scrollen op de pagina

Na een reflow moet de pagina wordt er altijd een repaint uitgevoerd. Een repaint wordt ook uitgevoerd wanneer het uiterlijk van een element veranderd, zoals bijvoorbeeld de tekstskleur van een element. Hiervoor hoeft geen reflow uitgevoerd te worden omdat de positie of de grootte van het beïnvloedde element niet veranderd.

Hoe kan de HTML van een webpagina worden geoptimaliseerd?

In de voorgaande analysevraag is te lezen dat een HTML-bestand wordt omgezet in een DOM Tree. De enige twee manieren om de DOM te optimaliseren zijn het minimaliseren van de DOM Tree en het minimaliseren van de diepte van de DOM Tree.

In de volgende subhoofdstukken worden deze twee optimalisaties verder uitgelegd.

Minimaliseren DOM

De DOM Tree kan geoptimaliseerd worden door de gehele DOM zo minimaal mogelijk te houden. Een kleinere DOM Tree is voor een browser sneller en makkelijker op te bouwen. Daarnaast is een kleinere DOM Tree sneller bij te werken wanneer er een verandering in de DOM plaatsvindt.

Het minimaliseren van de DOM Tree heeft ook tot gevolg dat de Render Tree beperkter in omvang wordt. Een kleinere Render Tree heeft tot gevolg dat de browser de Render Tree sneller kan updaten en de webpagina sneller kan renderen.

Minimaliseren diepte DOM

De HTML DOM kan naast kleiner, ook minder diep worden gemaakt (Simon, 2015). Een minder diepe DOM Tree zorgt voor een minder diepe Render Tree. Waneer er een reflow uitgevoerd wordt door de browser, moeten de elementen in de Render Tree opnieuw berekend worden. Wanneer er een reflow wordt veroorzaakt in een element met een diepe interne structuur, moeten alle subelementen ook opnieuw berekend worden. Dit zal in een ondiepe DOM veel minder vaak voorkomen.

Conclusie

De HTML DOM kan simpel gezegd worden geoptimaliseerd door de HTML DOM zo klein en zo ondiep mogelijk te houden.

Hoe kan de CSS van een webpagina worden geoptimaliseerd?

In de eerste analysevraag is te lezen dat de browser een CSSOM Tree opbouwt van het CSS-bestand. Voor het optimaliseren van de CSSOM Tree zijn een aantal optimalisaties beschikbaar.

In de volgende subhoofdstukken de optimalisaties verder uitgelegd.

Gebruik de meest efficiënte CSS selector

In een CSS-bestand worden style rules gedefinieerd. Een style rule begint altijd met een selector welke aangeeft op welke elementen de style rule toegepast moet worden. Er zijn een aantal verschillende selectors en elk type selector heeft een andere invloed op de performance van het gebruik van de selector.

In het onderstaande lijstje staan de CSS selectors van meest efficiënt tot het minst efficiënt (Roberts, 2011) (Coyier, 2010):

- ID selector: `#header`
- Class selector: `.item`
- Type selector: `div`
- Adjacent selector: `h1 + p`
- Child selector: `ul > li`
- Descendant selector: `ul a`
- Universal selector: `*`
- Attribute selector: `[type="text"]`
- Pseudo-classes: `a:hover`

Vermijd het gebruik van complexe CSS selectors

Naast het gebruik van de meest efficiente selector helpt het ook om de selectors zo simpel mogelijk te houden. Dit heeft ermee te maken dat een browser de selectors van rechts naar links interpreteert. Hieronder staat een voorbeeld van een selector:

Voorbeeld CSS selector

```
div .content div p
```

Deze CSS selector zou de volgende HTML structuur moeten matchen:

Voorbeeld HTML structuur

```
<div>
  <div class="content">
    <div>
      <p>
```

De reden dat de voorbeeldselector meer performanceverlies oplevert is dat de browser bij elk p-element moet nagaan of deze selector daarop van toepassing is. Wanneer de browser een p-element tegen komt, controleert de browser of het p-element een child is van een div-element. Wanneer dit niet zo is wordt deze style rule niet toegepast op dit p-element. Wanneer het p-element wel een child is van een div-element wordt er gecontroleerd of het div-element een child is van een element met de class "content". Wanneer dit niet zo is wordt deze style rule niet toegepast op het p-element. Wanner het div-element wel een child is van een element met de class "content" wordt er gecontroleerd of het element met de class "content" een child is van een div-element. Elke controle zorgt ervoor dat het toepassen van de style-rule langer duurt (Roberts, 2011) (Zbarsky, 2011).

Conclusie

Het optimaliseren van CSS is vooral gericht op het gebruiken van de snelste selectors en het minder toepassen van complexe CSS selectors. Echter zijn browsers tegenwoordig al goed geoptimaliseerd voor het toepassen van style rules. Hierdoor zal het optimaliseren van de gebruikte CSS weinig tot geen effect hebben op de rendersnelheid van de webpagina (Coyier, 2010) (Sullivan, CSS Selector Performance has changed! (For the better), 2011).

Conclusie

De browser heeft een vaste manier voor het verwerken van HTML- en CSS-bestanden. Dit proces kan alleen versneld worden door de hoeveelheid te verwerken informatie te minimaliseren. Dit geld voor het inladen van een webpagina, maar ook voor dynamisch wijzigen van de DOM op een webpagina. Daarom zijn de gevonden optimalisaties vooral gericht op het minimaliseren van de HTML DOM het minimaliseren van de diepte van de HTML DOM.

De optimalisaties op het gebied van CSS zijn vooral voor het gebruik van simpele en meest efficiënte selectors. Echter zijn browsers tegenwoordig al zodanig goed geoptimaliseerd dat het richten op CSS niet veel meer snelheidswinst zal opleveren.

Hierdoor blijft alleen het minimaliseren van de HTML DOM over als optimalisatiestrategie. Deze conclusie ondersteunt ook de optimalisatiestrategie welke het FontoXML team al opgesteld had, namelijk het toepassen van viewport culling. De andere optimalisatiestrategie welke het FontoXML team opgesteld had, heeft minder effect op de performance van de applicatie.

Deelvraag 3: Welke optimalisatiestrategieën zullen (de meeste) performancewinst opleveren?

Uit deelvraag 2 blijkt dat weinig optimalisatiestrategieën beschikbaar zijn voor het optimaliseren van HTML en CSS van een webpagina.

De gevonden optimalisatiestrategieën komen neer op het zo klein en ondiep mogelijk houden van de HTML DOM en het gebruik van zo simpel en efficient mogelijke CSS selectors. Deze conclusie wijst erop dat HTML DOM Culling, de oplossingsrichting welke het FontoXML ontwikkelteam al opgesteld had, zou kunnen zorgen voor een significante performanceverbetering.

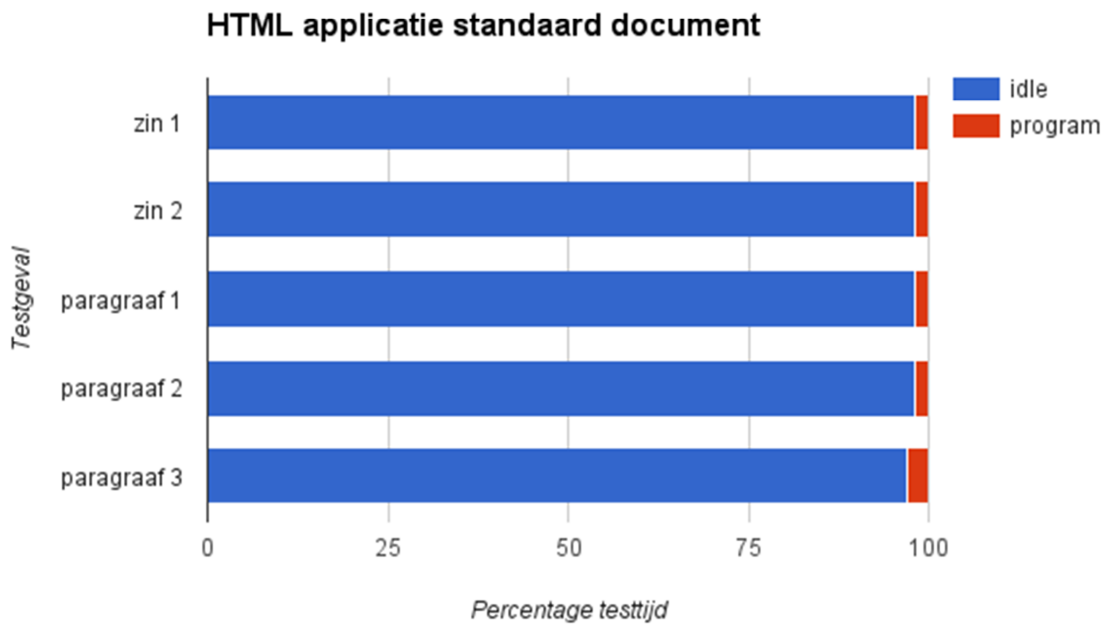
Om te kunnen bepalen of HTML DOM Culling een positieve invloed heeft op de performance van de applicatie is deze optimalisatiestrategie geïmplementeerd in een proof of concept.

Na de bouw van het proof of concept zijn de performancetests waarmee de baseline profile zijn bepaald opnieuw uitgevoerd met het proof of concept. De testresultaten hiervan zijn vergeleken met de testresultaten van de baseline profile.

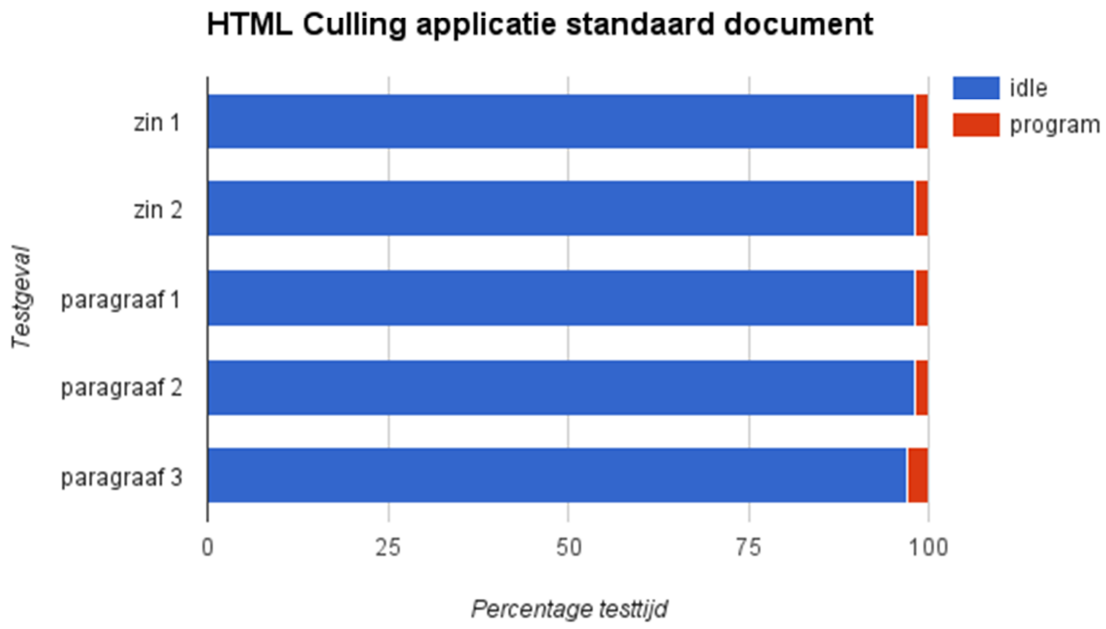
Testresultaten standaard document

Tijdens de performancetests met het standaard document in de HTML versie van de applicatie was er geen performanceverlies merkbaar. De resultaten van de performancetests met het standaard document in de HTML Culling versie van de applicatie zijn hetzelfde als de resultaten van de tests met de HTML versie van de editor.

Hieronder staan de resultaten van de performancetests met het standaard document in de HTML versie van de applicatie. Hieraan is te zien dat tijdens alle performancetests de browser voor het grootste gedeelte idle was en een klein deel van de testtijd bezig was met het renderen van de webpagina.



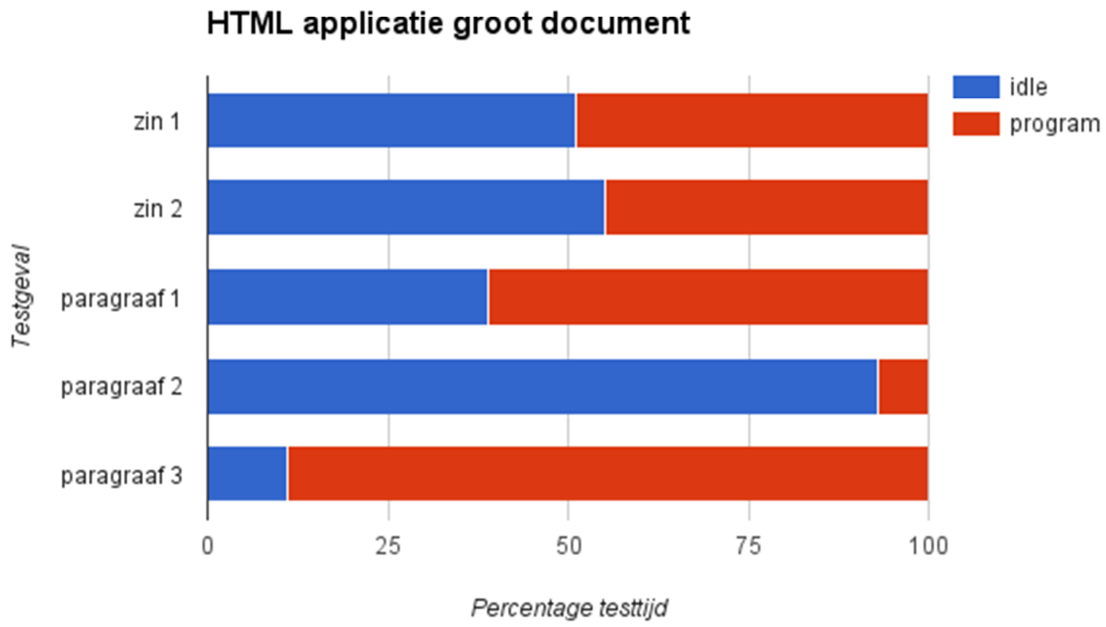
Hieronder staan de resultaten van de performancetests met het standaard document in de HTML Culling versie van de applicatie. Hieraan is te zien dat tijdens alle performancetests de browser voor het grootste gedeelte idle was en een klein deel van de testtijd bezig was met het renderen van de webpagina.



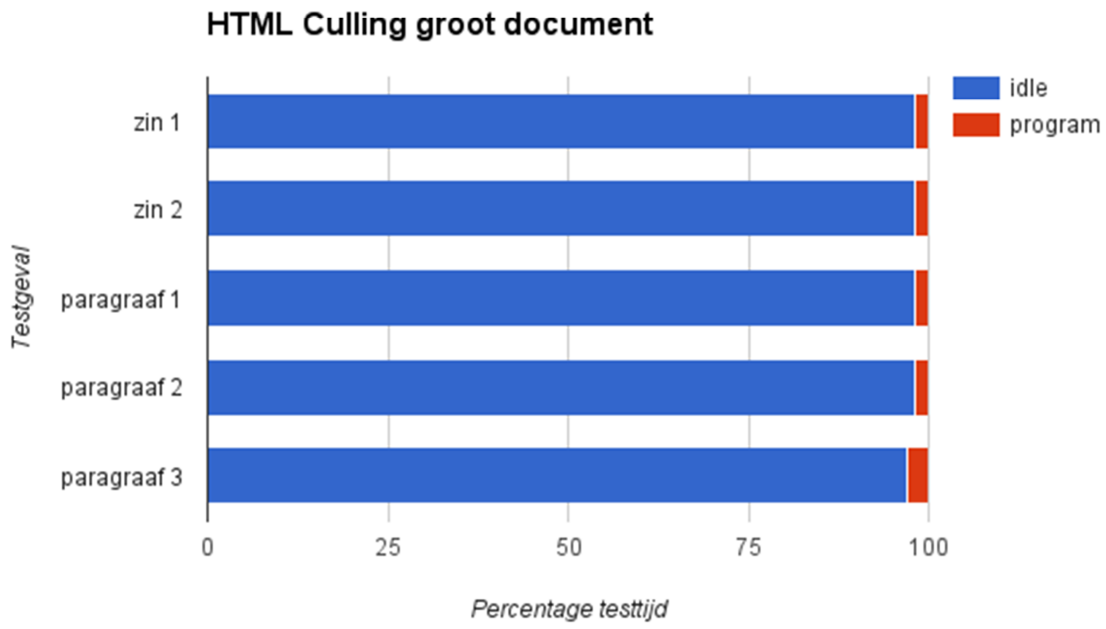
Testresultaten groot document

Tijdens de performancetests met het grote document in de HTML versie van de applicatie was er performanceverlies merkbaar. Dit is duidelijk terug te zien in de resultaten van de performancetests met het grote document in de HTML versie van de applicatie.

Hieronder staan de resultaten van de performancetests met het grote docuemtn in de HTML versie van de applicatie. Hieraan is te zien dat tijdens de performancetests de browser een groot deel van de testtijd bezig was met het renderen van de webpagina. Alleen tijdens de test waarbij een paragraaf aan het eind van het document wordt ingevoerd was de browser het grootste deel van de totale testtijd idle.



Hieronder staan de resultaten van de performancetests met het grote document in de HTML Culling versie van de applicatie. Hieraan is te zien dat tijdens alle performancetests de browser voor het grootste gedeelte idle was en een klein deel van de testtijd bezig was met het renderen van de webpagina.



Conclusie

Uit de testresulaten van de performancetests die zijn uitgevoerd met de HTML versie en de HTML Culling versie van de applicatie blijkt dat HTML DOM Culling een positief effect heeft op de performance van de applicatie. De resultaten van de test met het grote document in de HTML Culling versie van de applicatie geven hetzelfde resultaat als de tests met het standaard document in de HTML en HTML Culling versie van de applicatie. Hieruit blijkt dat HTML DOM Culling een positief effect heeft op de performance van de applicatie en het zinnig is deze optimalisatie toe te passen.

Bibliografie

Coyier, C. (2010, Mei 24). *Efficiently Rendering CSS*. Opgeroepen op 2015, van CSS-Tricks: https://css-tricks.com/efficiently-rendering-css/

Garsiel, T. (2011, Augustus 5). *How Browsers Work: Behind the scenes of modern web browsers*. Opgeroepen op 2015, van HTML5 Rocks: http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/

Grigorik, I. (2014, April 1). *Render-tree construction, Layout, and Paint*. Opgeroepen op 2015, van Google Developers: https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-tree-construction?hl=en

Lew, L. (2012, Augustus 28). *Repaints and Reflows: Manipulating the DOM responsibly*. Opgeroepen op 2015, van temp.blogname: http://blog.letitialew.com/post/30425074101/repaints-and-reflows-manipulating-the-dom

Roberts, H. (2011, September 17). *Writing efficient CSS selectors*. Opgeroepen op 2015, van CSS Wizardry: http://csswizardry.com/2011/09/writing-efficient-css-selectors/

Simon, L. (2015, April 8). *Minimizing browser reflow* . Opgeroepen op 2015, van Google Developers: https://developers.google.com/speed/articles/reflow

Stefanov, S. (2009, December 17). *Rendering: repaint, reflow/relayout, restyle*. Opgeroepen op 2015, van phpied.com: http://www.phpied.com/rendering-repaint-reflowrelayout-restyle/

Sullivan, N. (2009, Maart 27). *Reflows & Repaints: CSS Performance making your JavaScript slow?* Opgeroepen op 2015, van Stubbornella: http://www.stubbornella.org/content/2009/03/27/reflows-repaints-css-performance-making-your-javascript-slow/

Sullivan, N. (2011, December 29). *CSS Selector Performance has changed! (For the better)*. Opgeroepen op 2015, van Performance Calendar: http://calendar.perfplanet.com/2011/css-selector-performance-has-changed-for-the-better/

Zbarsky, B. (2011, April 26). *Why do browsers match CSS selectors from right to left?* Opgeroepen op 2015, van stackoverflow: http://stackoverflow.com/questions/5797014/why-do-browsers-match-css-selectors-from-right-to-left

FontoXML

Adviesrapport

Bijlage F

Thomas Gravekamp
25-05-2015

Managementsamenvatting

Tijdens het bewerken van documenten van enkele honderden pagina's in de FontoXML applicatie is een performanceverlies merkbaar. Dit performanceverlies is hinderlijk voor de gebruiker van de applicatie.

Uit dit probleem is de volgende hoofdvraag voortgekomen: "Welke optimalisaties op het gebied van het renderen van een webapplicatie hebben een positief effect op de performance van FontoXML?".

Tijdens het onderzoek is gezocht naar een antwoord op deze hoofdvraag. Hiervoor zijn eerst de resultaten van de baseline profile onderzocht. De baseline profile bestaat uit de testresultaten van de performancetests met de bestaande situatie. Hieruit bleek dat de browser een groot deel van de tijd innam om de webpagina opnieuw te renderen.

Hierna is gezocht naar optimalisatiestrategieën die kunnen worden toegepast op het gebied van de HTML en CSS van een webpagina. Uit deze optimalisatiestrategieën is één strategie overgebleven om een proof of concept van te bouwen.

Met dit proof of concept zijn de performancetests herhaald. De resultaten van de performancetests met de beginsituatie en het proof of concept zijn vergeleken. Hieruit bleek dat het toepassen van HTML DOM Culling een positief effect heeft op de performance van de applicatie.

Het advies is om HTML DOM Culling toe te passen op de applicatie voor een verbetering van de performance.

Inhoud

Inleiding

Dit adviesrapport is bedoeld voor het ontwikkelteam van FontoXML. Het doel van dit adviesrapport is het adviseren van een optimalisatiestrategie voor FontoXML om de performance van FontoXML mee te verbeteren.

De aanleiding van het uitgevoerde onderzoek is het performanceverlies in FontoXML wanneer een groot document van enkele honderden pagina's wordt geopend. Dit performanceverlies is voor de gebruiker van FontoXML hinderlijk. Daarbij kwam de wens vanuit de klanten van FontoXML om documenten van enkele honderden pagina's te kunnen openen.

Het gegeven advies is gebaseerd op het uitgevoerde onderzoek naar de mogelijke optimalisatiestrategieën op het gebied van HTML en CSS.

Onderzoek

De hoofdvraag die tijdens het onderzoek beantwoordt is, luidt: "Welke optimalisaties op het gebied van het renderen van een webapplicatie hebben een positief effect op de performance van FontoXML?".

Om deze hoofdvraag te kunnen beantwoorden zijn de volgende deelvragen gesteld:

- Deelvraag 1: Waardoor wordt het performanceverlies in FontoXML momenteel veroorzaakt?
- Deelvraag 2: Welke optimalisatiestrategieën kunnen worden toegepast op het gebied van HTML en CSS?
- Deelvraag 3: Welke optimalisatiestrategieën zullen (de meeste) performancewinst opleveren?

Tijdens het beantwoorden van deelvraag 1 is gezocht naar de oorzaak van het performanceverlies in FontoXML. Uit de testresultaten van de baseline profile bleek dat in de het grootste performanceverlies werd veroorzaakt door de scripts van de applicatie zelf. Echter vielen optimalisaties op het gebied van scripts buiten de scope van dit onderzoek. Uit de performancetests met de HTML versie van de applicatie bleek dat het renderen van de webpagina het performanceverlies veroorzaakte.

Tijdens het beantwoorden van deelvraag 2 is gezocht naar optimalisatiestrategieën die kunnen worden toegepast op het gebied van de HTML en CSS van een webpagina. Hiervoor is eerst de werking van browsers onderzocht. Daarna is verder onderzocht welke optimalisatiestrategieën er zijn voor het optimaliseren van HTML en CSS.

De twee optimalisatiestrategieën voor HTML zijn de volgende:

- Verminderen van de grootte van de HTML DOM
- Verminderen van de diepte van de HTML DOM

Het verminderen van de diepte van de HTML DOM is lastig en levert in verhouding tot het toepassen van deze optimalisatiestrategie weinig op.

De twee optimalisatiestrategieën voor CSS zijn de volgende:

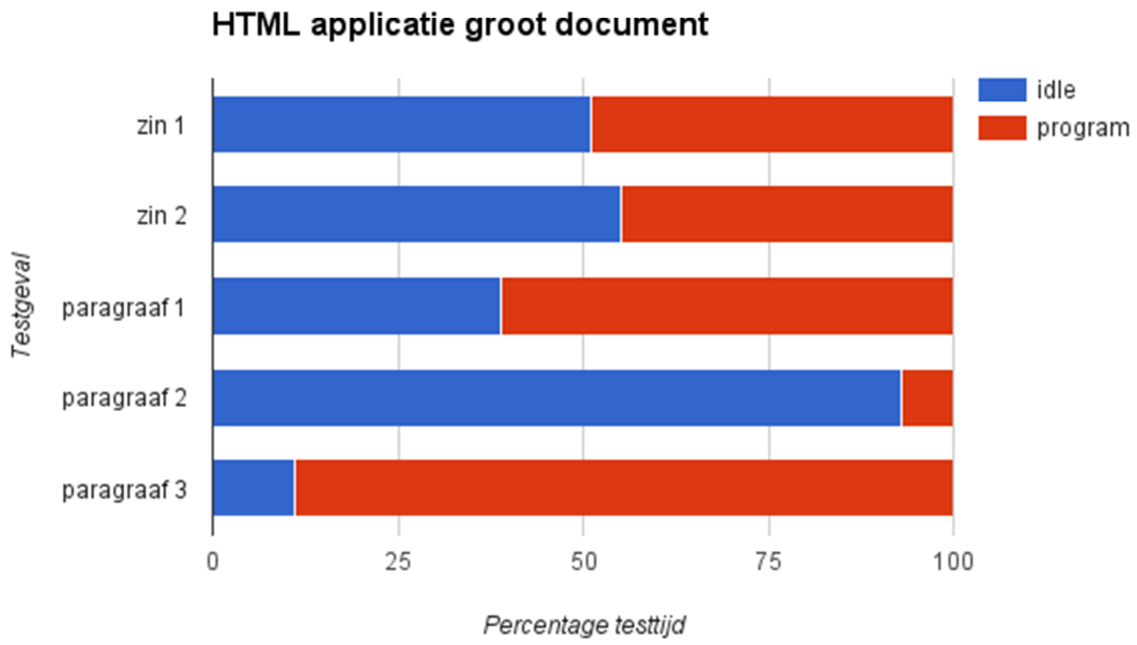
- Gebruik de meest efficiente CSS selector
- Vermijdt het gebruik van complexe CSS selectors

Echter zijn browsers tegenwoordig zodanig geoptimaliseerd dat het optimaliseren van CSS haast geen effect meer heeft op de performance van een webpagina. Hierdoor bleven alleen de twee optimalisatiestrategie

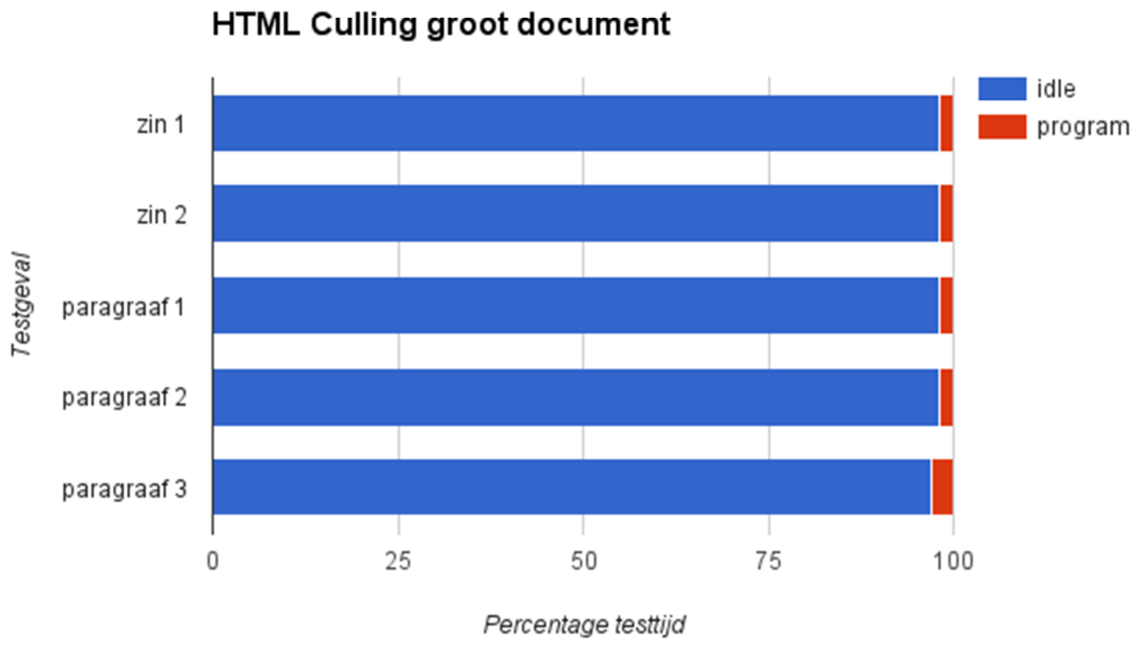
De uiteindelijk gevonden optimalisatiestrategie was het verminderen van de grootte van de HTML DOM.

Tijdens het beantwoorden van deelvraag 3 is een proof of concept gebouwd waarin de in deelvraag 2 gevonden optimalisatiestrategie is geïmplementeerd. Deze optimalisatiestrategie is HTML DOM Culling.

Na het implementeren van deze optimalisatiestrategie zijn de performancetests van de baseline profile herhaald met het proof of concept. Hieruit bleek dat het toepassen van viewport culling een grote positieve impact heeft op de performance van de applicatie. Dit is te zien aan de volgende twee diagrammen.



Dit diagram laat de resultaten van de performancetests met het grote document zonder HTML DOM Culling zien.



Dit diagram laat de resultaten van de performancetests met het grote document met HTML DOM Culling zien.

Alternatieven

De enige oplossingsrichting is het toepassen van HTML DOM Culling. Deze optimalisatiestrategie laat in het proof of concept duidelijk zien dat het nuttig is om deze optimalisatiestrategie toe te passen.

Andere oplossingsrichtingen zullen gezocht moeten worden op het gebied van Javascript.

Advies

Het advies is om HTML DOM Culling toe te gaan passen in FontoXML. Op het gebied van HTML en CSS zijn geen andere oplossingsrichtingen die evenveel of meer positieve invloed zullen hebben op de performance van de applicatie.