Afstudeerverslag Viradj Jainandunsing

Viradj Jainandunsing - 413300
Saxion Hogeschool Enschede - Informatica
Avisi B.V. te Arnhem

17 juni 2019

Samenvatting

Dit document betreft het afstudeerverslag van mij, Viradj Jainandunsing. De afstudeeropdracht is uitgevoerd bij Avisi B.V. te Arnhem, van 11 februari 2019 tot 12 juli 2019.

Achtergrondinformatie

Avisi is een softwareontwikkelbedrijf dat gespecialiseerd is in maatwerksoftware. Eén van de ontwikkelteams van Avisi ontwikkelt een softwarepakket voor Nobilex. Dit is een systeem waarmee notarissen al hun dagelijkse werkzaamheden moeten kunnen doen. Het Nobilex-team werkt volgens de agile SCRUM methodiek, waarbij uiteraard in iedere sprint ook getest wordt. Dit zijn voornamelijk functionele testen. Omdat het product *Nobilex* een grafische gebruikersinterface in de vorm van een webapplicatie heeft, is het wenselijk dat de eindgebruikers geen onverwachte visuele veranderingen te zien krijgen. Deze wens heeft ertoe geleid dat deze afstudeeropdracht is bedacht en uitgevoerd.

Afstudeeropdracht

De afstudeeropdracht had in brede zin betrekking op het verbeteren van de front-end testen van Avisi en kende in eerste instantie drie facetten: visueleregressietesten (VRT), functionele testen en toegankelijkheidstesten. Deze aspecten hebben geleid tot een opdracht waarin onderzocht moest worden hoe Avisi de kwaliteit van haar webapplicaties beter kan waarborgen door middel van front-end testen. Hierbij heeft de nadruk vooral gelegen op VRT. Wanneer er een visuele aanpassing wordt gedaan op een pagina kan dit ongewenst invloed hebben op het uiterlijk van andere pagina's. Bijvoorbeeld wanneer twee pagina's eenzelfde component gebruiken en het component ten behoeve van slechts één van deze pagina's aangepast wordt. Met een oplossing voor VRT, kan de ongewenste verandering op de andere pagina ontdekt worden en kan dit opgelost worden, voordat de wijziging naar productie gaat. De opdracht was om onderzoek te doen naar de beste manier waarop Avisi VRT kon toepassen en om hier een proof of concept bij te maken.

Management

Tijdens het project is er gewerkt volgens een variatie op de Rational Unified Process (RUP) softwareontwikkelmethode. De afstudeerperiode en de op te leveren producten zijn verdeeld in vijf fases met alle een doel waaraan is gewerkt om naar de volgende fase te gaan. Omdat ik in het Nobilex team heb gewerkt (weliswaar aan mijn eigen opdracht), heb ik de daily standups bijgewoond die bij hun SCRUM proces horen. Hiernaast heb ik minstens één keer per week met de bedrijfsbegeleider een voortgangsoverleg gehad.

Onderzoek

Het onderzoeksgedeelte van deze opdracht is opgedeeld in vier onderdelen: de inventarisatie van bestaande oplossingen, het opstellen van requirements, het toetsen van bestaande oplossingen aan de hand van de opgestelde requirements en een aanbeveling van de beste keuze. Uit het onderzoek kwam naar voren dat er geen pasklare bestaande producten zijn die aan alle requirements voldoen. Zodoende is er gekozen voor een composietoplossing, bestaande uit de testrunner Cypress.io en de afbeeldingvergelijker Pixelmatch.

Proof of concept

De composietoplossing bestaande uit een testrunner die screenshots kan maken en een tool die screenshots kan vergelijken, leidde tot de vraag waar deze afbeeldingen opgeslagen moesten worden. Dit was interessant, omdat de uiteindelijke locatie, namelijk in de repository zelf, anders was dan aanvankelijk gedacht. Eerst werd er gedacht aan de afbeeldingen op aparte git repository op te slaan of in de cloud. Door te testen met de afbeeldingen in de repo werd ontdekt dat Bitbucket (versiebeheersysteem) ook kijkt naar veranderde afbeeldingen. Uiteindelijk speelt dit mechanisme een grote rol in de proof of concept.

Inhoudsopgave

1. Versiegeschiedenis

Versienummer	Datum	Aanpassingen	Bewerker
0.1	20/03/'19	Eerste opzet	Viradj Jainandunsing
0.2	19/05/'19	Feedback verwerkt	Viradj Jainandunsing
1.0	03/06/'19	Concept oplevering	Viradj Jainandunsing
2.0	17/06/'19	Definitieve oplevering	Viradj Jainandunsing

2. Theoretisch kader

2.1 Begrippen

Regressietesten

Regressie is het verschijnsel dat de kwaliteit van een systeem als geheel terugloopt als gevolg van individuele aanpassingen. Om schade als gevolg van deze softwarefouten te voorkomen wordt een regressietest uitgevoerd. De regressietesten hebben als doel om vast te stellen of alle systeemonderdelen goed blijven functioneren na het doorvoeren van één of meerdere wijzigingen (“Regressietesten, De tien uitgangspunten. - Testersuite”, 2019).

Visueleregressietesten

Visueleregressietesten zijn bedoeld om regressie op het uiterlijk van een systeem te detecteren. Dit wordt vooral gedaan door screenshots te vergelijken van schermen vóór een wijziging en na een wijziging. (“Visual Regression Testing. Wat is het en wat heb je eraan? – Performance architecten”, 2018).

Functionele testen

Met functionele testen wordt de werking van software getest. Dit kan zo klein zijn als een unit-test waarbij de kleinste stukken uit software zoals methodes getest worden, tot end to end testen waarbij functionaliteit van een systeem getest wordt en waarbij zelfs rekening gehouden wordt met externe services, zoals REST API's.

Image-diffing

Met behulp van image-diffing kunnen twee afbeeldingen van dezelfde afmeting vergeleken worden. Dit wordt gedaan door iedere pixel van de ene afbeelding te vergelijken met de pixel op de overeenkomstige positie van de andere afbeelding.

Baseline

De verzameling van afbeeldingen die zijn geaccepteerd als een correcte weergave van een scherm en gezien wordt als *Single Source of Truth*.

2.2 Afkortingen

Afkorting	Betekenis
VRT	visueleregressietesten
RUP	Rational Unified Proces
PvA	Plan van aanpak
PoC	Proof of Concept
VCS	Versiebeheersysteem (Version Control System)
CI/CD	Continuous Integration en Continuous Delivery
CLI	Command Line Interface
AI	Artificial Intelligence
UI	User-interface

3. Inleiding

Dit document betreft het afstudeerverslag van mij, Viradj Jainandunsing. De afstudeeropdracht wordt uitgevoerd bij Avisi B.V. te Arnhem, van 11 februari 2019 tot 12 juli 2019. In dit verslag wordt de uitvoering van de afstudeeropdracht toegelicht.

3.1 Documentoverzicht

Achtergrondinformatie

In dit hoofdstuk wordt achtergrondinformatie gegeven over het bedrijf waar de afstudeeropdracht is uitgevoerd en wordt uitgelegd wat de probleemstelling van de opdracht is.

Afstudeeropdracht

Hier wordt de definitie en het doel van de afstudeeropdracht verder toegelicht en wordt uitgelegd wat mijn persoonlijke doelen waren omtrent de afstudeerperiode.

Beoogde aanpak

In de eerste weken is gewerkt aan een plan van aanpak (PvA). Dit hoofdstuk is erop gericht om een beeld te schetsen van hoe ik het verloop van het project voor me zag voordat ik begon aan de daadwerkelijke uitvoering.

Verloop proces

In het hoofdstuk *Beoogde aanpak* is aan bod gekomen hoe ik het proces en aanpak van de afstudeeropdracht voor me zag. In dit hoofdstuk vertel ik over het daadwerkelijke verloop van de afstudeerperiode. Hierbij wordt verteld waar ik ben afgeweken van het originele plan en waarom ik hiervoor gekozen heb. Tot slot geef ik per fase van het project aan wat mijn bevindingen waren en wat ik anders zou doen als het over mocht.

Opgeleverde resultaten

In dit deel van het verslag komt naar voren wat mijn onderzoeksplan was om kennis op te doen over het onderwerp van de afstudeeropdracht, de resultaten van het onderzoek worden benoemd en ook wordt er een conclusie van het onderzoek gegeven. Nadat de requirements van Avisi en bestaande oplossingen in kaart waren gebracht in het onderzoek, is er gewerkt aan een proof of concept. In dit hoofdstuk zullen zodoende ook het ontwerp en de werking van deze proof of concept toegelicht worden, met een onderbouwing van gemaakte keuzes.

Conclusie en aanbevelingen

Tot slot komt in dit onderdeel een conclusie van de afstudeeropdracht. Ik benoem wat de mogelijke vervolgstappen zijn en ik licht toe welke doelen er zijn behaald.

Graag wil ik hier ook van de gelegenheid gebruik maken om een aantal personen te bedanken die mij geholpen hebben bij de uitvoering van dit afstudeerproject:

In de eerste plaats wil ik graag mijn bedrijfsbegeleider Jeroen Schonenberg en mijn afstudeerdocent Willem Prakken bedanken voor hun begeleiding waar ik altijd op kon rekenen.

Ook wil ik Avisi B.V. bedanken voor het aanbieden van de afstudeeropdracht en beschikbaar maken van de aanwezige kennis van collega's die mij gedurende de afstudeerperiode hebben geholpen, hierbij de testers-guild en het Nobilex Spot team in het bijzonder.

4. Achtergrondinformatie

4.1 Bedrijf

Na het winnen van een Award Winning Vision afstudeerprijs hebben drie studievrienden, Jan Bakker, Barri Jansen en Gert-Jan van de Streek, Avisi in 2000 opgericht op de zolderkamer van Jan Bakker. Zij hebben een grote passie voor techniek en hebben het doel om in 2030 "Het beste softwarebedrijf van Nederland" te zijn. Avisi levert maatwerksoftware aan een aantal grote klanten, waaronder:

NXP

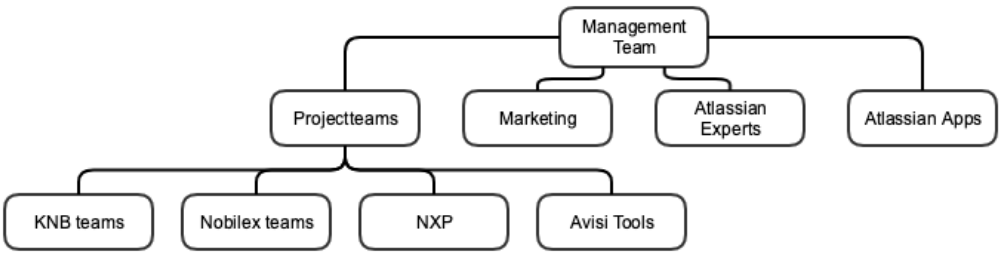
NXP is een chipfabrikant gevestigd in Nijmegen. Bij het gebruiken van de productiemachines die de chips fabriceren worden een groot aantal omgevingsvariabelen gemeten waarvoor Avisi zorgt dat deze gemonitord kunnen worden. Op deze manier kan bijvoorbeeld predictive maintenance worden toegepast.

KNB

De Koninklijke Notariële Beroepsorganisatie is een organisatie die moet zorgen voor een goede beroepsuitoefening door notarissen. Voor de KNB heeft Avisi meer dan tien applicaties gemaakt, waaronder een aantal centrale registers.

Nobilex

Nobilex ontwikkelt een systeem voor notarissen dat meer functionaliteiten biedt dan huidige oplossingen. Veel van Nobilex heeft samenhang met het PEC platform dat beheerd wordt door één van de KNB-teams binnen Avisi.



Afbeelding 1: Organogram van Avisi

4.2 Probleemstelling

Softwareontwikkelbedrijven testen doorgaans de producten die zij maken. Binnen Avisi worden al uitgebreid functionele regressietesten uitgevoerd die controleren of de software naar behoren functioneert. Deze regressietesten worden telkens uitgevoerd na de ontwikkeling van een nieuwe feature, om te garanderen dat alle onderdelen die goed werkten nog steeds naar behoren werken na de verandering.

De problemen komen echter kijken bij het testen van visuele veranderingen in de front-end van web-applicaties. Dit wordt namelijk nog niet gedaan. Naast de garantie dat de software functioneel nog goed werkt, wil Avisi ook dat een webpagina er niet onverwacht compleet anders uitziet na een aanpassing van bijvoorbeeld een losstaand onderdeel.

Ook is er de wens om de toegankelijkheid van webpagina's inzichtelijk te maken middels geautomatiseerde testen, zodat mensen met beperkingen ook zo goed mogelijk gebruik kunnen maken van de front-end.

De afstudeeropdracht wordt uitgevoerd in het Nobilex team, maar de te ontwikkelen oplossing zal bij meerdere teams toepasbaar zijn.

5. Afstudeeropdracht

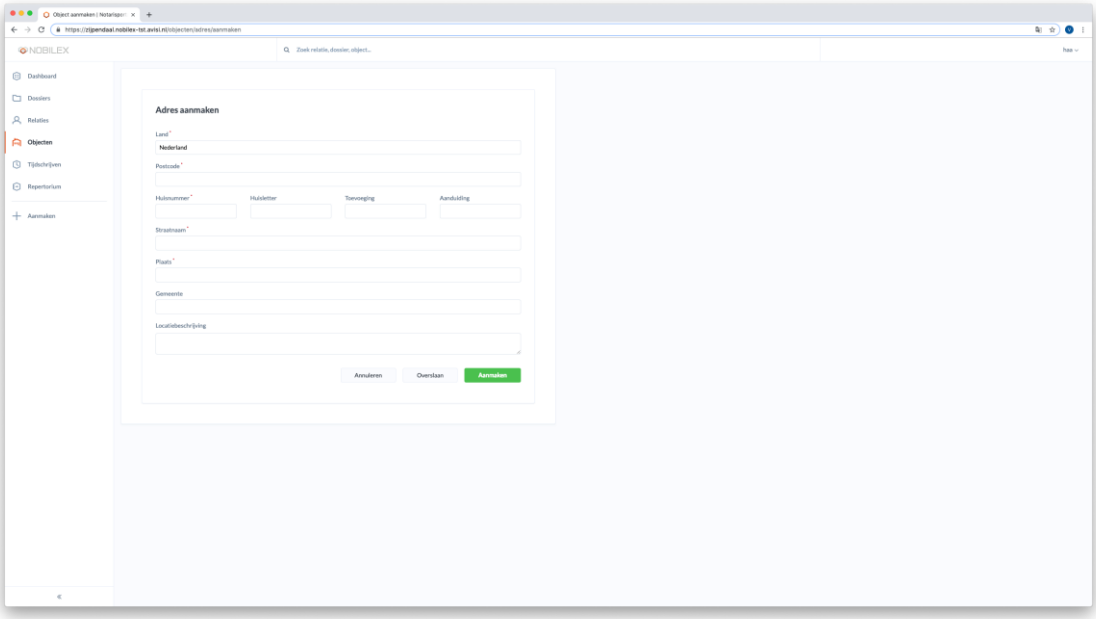
5.1 Context

Binnen Avisi worden door verschillende teams webapplicaties ontwikkeld. Deze webapplicaties worden geautomatiseerd functioneel getest, al gebruikt niet ieder team dezelfde test-frameworks.

De afstudeeropdracht heeft betrekking op het verbeteren van de front-end testen en heeft drie facetten die hieronder in aflopende prioriteit beschreven worden aan de hand van een voorbeeld dat de probleemstelling illustreert.

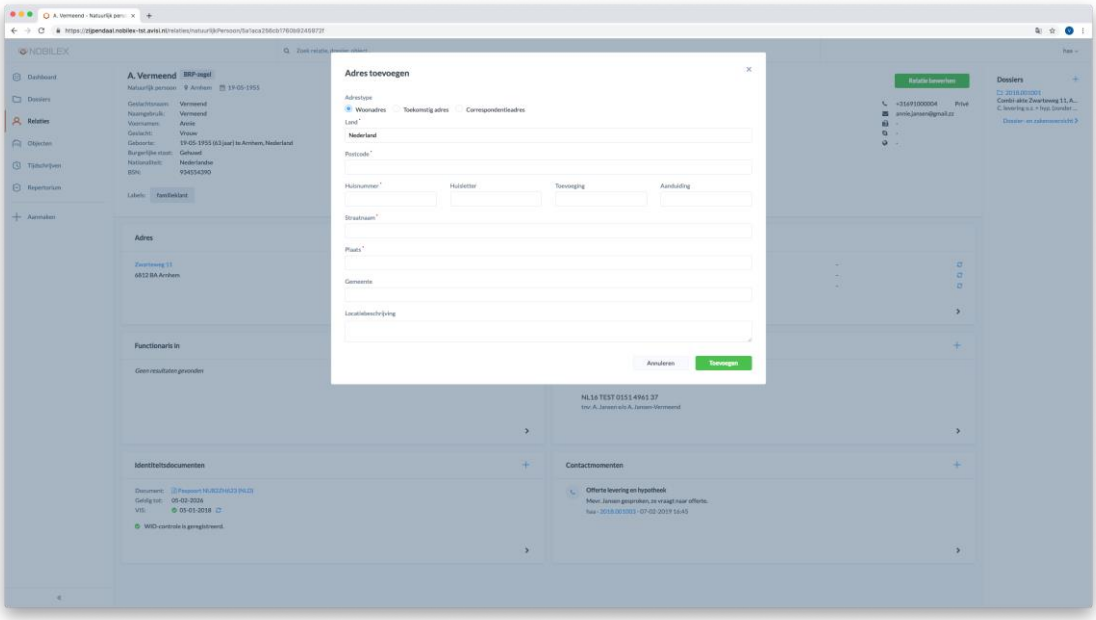
1) Front-end testen op visuele regressies

Wanneer er visuele aanpassingen worden gedaan aan een pagina, kan dit invloed hebben op andere pagina's. Hierdoor is het uitvoeren van regressietesten na het doorvoeren van deze aanpassingen belangrijk. In het volgende voorbeeld is te zien hoe een pagina een formulier bevat, maar dat deze een ongewenste border heeft:



Afbeelding 2: formulier met overbodige border

Dit komt doordat het formulier een component is die op een ander scherm in de vorm van een modal weergeven wordt, waar de border wel nodig was:



Afbeelding 3: modal met gewenste border

Omdat de border later aan het component is toegevoegd toen er voor de modal behoefte aan was, is over het hoofd gezien dat dit ook effect had op het scherm in *Afbeelding 2: formulier met overbodige border*. Een visueleregressietest had in dit geval deze verandering opgemerkt en de developers hiervan op de hoogte gesteld, zodat de container waar het component in zit weggehaald had kunnen worden.

2) Onderzoek doen naar de huidige manier van functionele testen en kritisch kijken wat de beste optie is

Omdat er in de verschillende teams op verschillende manieren functioneel getest wordt, zal er onderzocht worden wat de beste manier is. Dit houdt ook in dat er onderzoek gedaan moet worden naar frameworks en technieken die nog niet toegepast worden binnen het bedrijf. Het kan zijn dat het goed is dat er verschillen bestaan, omdat bepaalde frameworks beter aansluiten bij de behoeftes van een team, maar misschien bestaat er wel één oplossing waar alle teams mee uit de voeten kunnen.

3) Het testen van toegankelijkheid aan de hand van richtlijnen

Dit aspect is het best te begrijpen in de context van een oud ontwikkelteam van Avisi, namelijk: Meddex (Medical Data Exchange). Meddex ontwikkelde een portal waarop zorginstanties medische gegevens van patiënten met elkaar konden delen. Uiteindelijk moesten de patiënten zelf ook toegang krijgen tot het systeem, maar bij deze groep gebruikers zal het vaak voorkomen dat ze een beperking hebben. Gelukkig bestaan er lijsten met richtlijnen waaraan websites moeten voldoen om toegankelijk te zijn voor zoveel mogelijk mensen en verschillende user-agents. Er zijn bijvoorbeeld de Web Content Accessibility Guidelines (WCAG) en Accessible Rich Internet Application (ARIA), waarin richtlijnen staan zoals: "alle non-text content moet tekstuele alternatieven hebben zodat dit bijvoorbeeld automatisch voorgelezen kan worden" en "alle functies moeten bereikbaar zijn met een toetsenbord" (“WAI-ARIA Overview”, z.d.).

Samengevat komen deze aspecten neer op de volgende hoofdvraag: 'Hoe kan Avisi de kwaliteit van haar webapplicaties beter waarborgen door middel van verbetering van de front-end testen?'.

Omdat er al automatische functionele testen worden gedaan moet er in de eerste plaats onderzocht worden hoe Avisi visueleregressietesten zou kunnen toepassen. Omdat de visueleregressietesten deels hetzelfde moeten doen als functionele testen, wordt aan de functionele testen in de tweede plaats aandacht besteed. Tot slot is het de wens om onderzoek te doen naar een manier om toegankelijksheidstesten uit te voeren binnen het bedrijf. Uiteindelijk moet er een proof of concept gemaakt worden voor de visueleregressietesten die in de build-pipeline van het Nobilex team ingezet kan worden.

5.2 Doelstellingen

Een vereiste voor het succesvol afsluiten van het afstudeerproject is het aan kunnen tonen dat de afstudeerder in het HBO-I 25-vlaksmodel ("HBO-i-domeinbeschrijving", z.d.) in minstens drie facetten die bij zijn/haar profiel horen voldoende competenties bezit. Dit betekent voor een afstudeerder met het Software Engineering profiel dat in de architectuurlaag "Software" drie van de competenties uit: beheren, analyseren, adviseren, ontwerpen en realiseren op niveau drie moeten zijn aan het eind van het afstuderen. Globaal gezien betekent niveau drie dat men zelfstandig een complex probleem kan oplossen, waarbij eigen initiatief en een innovatieve houding worden getoond. De volgende competenties worden aangetoond met deze opdracht:

Analyseren is een competentie die in dit project hoe dan ook aan bod komt, omdat eerst onderzocht moet worden wat de belanghebbenden van het project willen en tegen welke problemen zij aanlopen. Daarnaast zal er veel onderzoek worden gedaan naar bestaande oplossingen om uit te zoeken of (delen van) deze een oplossing voor Avisi bieden.

Ontwerpen komt ook aan bod, omdat de gekozen oplossingen moeten integreren in een bestaande CI/CD pipeline (in dit geval Bitbucket i.c.m. Bamboo van Atlassian) en de developer workflow. Over de precieze positie binnen het systeem moet nagedacht worden en hier moet een ontwerp voor gemaakt worden.

Afhankelijk van de requirements en de bestaande oplossingen kan er geconcludeerd worden of er een oplossing bestaat die de wensen van Avisi vervuld waarna er een **advies** gegeven kan worden, of dat er nog geen oplossing bestaat en er dus een product **gerealiseerd** moet worden.

6. Beoogde aanpak

In deze paragraaf komt de beoogde aanpak van procesgerelateerde zaken aan bod. In het volgende hoofdstuk worden de afwijkingen van het originele plan benoemd en toegelicht.

6.1 Softwareontwikkelmethode

Bij de meeste schoolprojecten en mijn stage heb ik gewerkt volgens de agile softwareontwikkelmethode SCRUM. Gezien de organisatie van de meeste afstudeeropdrachten heb ik gekozen om nog even verder te kijken dan SCRUM zomaar te willen gebruiken. Omdat SCRUM voornamelijk is ontwikkeld voor het werken in teamverband (Wat is Scrum? Simpele Uitleg van de Methode (+ Infographic). (2018, 27 november)) en dit afstudeerproject door één persoon wordt uitgevoerd, is gekeken naar een methodologie die beter bij deze opdracht past. De keuze hierin is gevallen op een variatie op Rational Unified Process (RUP), vanwege de structuur die de fasering biedt en de onbekendheid van de exacte requirements. RUP is agile en er wordt net als bij SCRUM incrementeel gewerkt ("Rational Unified Process - Wikipedia", 2019). Er zal worden gewerkt in vijf fases, die gebaseerd zijn op de fases die RUP kent:

Voorbereiden (inceptie)

In de voorbereiding wordt gewerkt aan het plan van Aanpak (PvA). Het doel is om de haalbaarheid, scope en grenzen te bepalen en om uiteindelijk een GO/NOGO te krijgen. Deze fase komt in RUP overeen met de *inceptiefase*.

Onderzoeken (elaboratie)

In de onderzoeksfase worden de requirements van het project vastgesteld en wordt kennis opgedaan die nodig is voor de implementatiefase. Bij RUP is de tweede fase *elaboratie*, waarin een projectplan gemaakt wordt ("Rational Unified Process - Wikipedia", 2019), maar omdat deze in de voorbereidingsfase in de vorm van een PvA al gemaakt is, wordt hierin afgeweken van RUP. Tijdens deze fase kunnen de strategieën uit het DOT-model ("ICT research methods", z.d.) aan bod komen, zoals ook het hands-on testen van bestaande producten. Het resultaat van deze fase is een onderzoeksrapport waarin voldoende kennis (zoals de huidig gebruikte frameworks en manieren om visuele tests uit te voeren) staat om te kunnen beginnen met de implementatiefase. Dit rapport kan echter worden bijgewerkt wanneer er meer onderzoek nodig is in volgende fases.

Implementeren (constructie)

Nadat de requirements in de onderzoeksfase een concretere vorm hebben gekregen en onderzoek is gedaan naar benodigde technieken, kan er worden begonnen aan deze fase. Hierin wordt een Proof of Concept (PoC) gemaakt die voldoet aan de eerder opgestelde eisen van het project.
De *constructiefase* van RUP is bedoeld om een systeem te ontwikkelen dat compleet genoeg is om te testen, hierin wijkt deze fase niet af.

Testen (transitie)

Wanneer de PoC af is kan deze worden getest. Gedurende de implementatiefase worden losse onderdelen getest, maar in de testfase wordt het complete systeem getest. RUP kent voor het testen de *transitiefase* waarin ook activiteiten worden uitgevoerd zoals deployment, overdracht en het opleiden van gebruikers. Dit zijn geen activiteiten die vallen binnen de scope van dit project, hoewel er wel een gebruikershandleiding geschreven zal worden waarin staat hoe de PoC te gebruiken is.

Afronden

De laatste fase van het afstudeerproject is gefocust op de afronding van het geheel. Alle op te leveren documenten worden in orde gemaakt, er wordt een presentatie / verdediging voorbereid en er wordt een reflectieverslag geschreven voor het hele project.

6.2 Wekelijkse evaluatie

Iedere week op maandagochtend wordt er samen met de bedrijfsbegeleider geëvalueerd hoe de afgelopen week is verlopen. Ook worden plannen voor de komende week gedeeld en wordt hier feedback op gegeven.

6.3 Maandelijkse oplevering tussentijdse documenten

Zoals aangeraden in de afstudeerhandleiding en door verschillende collega's wordt iedere maand de tussentijdse documentatie opgeleverd. De maandag voor het inleveren wordt met de bedrijfsbegeleider tijdens de evaluatie de tussentijdse documentatie doorgenomen zodat er voor het inleveren nog tijd is om feedback te verwerken. In de week van het inleveren is er ook een afspraak met de afstudeerdocent om het ingeleverde werk te bespreken.

6.4 Testers-guild

Avisi kent een aantal groepen van mensen met een interesse in een bepaald onderwerp, een zogeheten *guild*. Iedere twee weken komt een guild bijeen om bijvoorbeeld kennis te delen over het onderwerp of om een nieuwe techniek te demonstreren. Zo is er ook de *testers-guild* die gezamenlijk nadenken over testvraagstukken of een testwerkwijze die voor alle teams bruikbaar is. Gedurende de afstudeerperiode worden de meetings van deze guild bijgewoond om dichtbij de uiteindelijke gebruikers van de opdracht te zitten, zodat inzichtelijk gemaakt kan worden wat hun wensen zijn met betrekking tot de oplossing waaraan gewerkt wordt.

7. Verloop proces

7.1 Softwareontwikkelmethode

In de onderstaande paragrafen staat het daadwerkelijke verloop van iedere fase in het project beschreven. Hierbij ligt de nadruk op afwijkingen ten opzichte van het beoogde plan, met daarbij een toelichting voor deze afwijking.

Opstarten

De opstartfase verliep niet anders dan verwacht. Er werd een plan van aanpak (PvA) opgeleverd en een Go gegeven voor het project.

Onderzoeken

Bij het opstellen van het PvA werd gedacht dat het verstandig was om eerst het gehele onderzoek uit te voeren om daarna te beginnen met het implementeren. Echter, tijdens de onderzoeksfase, is gebleken dat het onderdeel omtrent visueleregressietesten een hogere prioriteit had dan de andere twee onderdelen. Zodoende is er in overleg met de bedrijfsbegeleider afgesproken dat er vooral gefocust zou worden op de visueleregressietesten. De functionele testen en toegankelijkheidstesten zouden pas gedaan worden wanneer hier genoeg tijd voor zou zijn. Qua planning was de scheiding tussen de onderzoeksfase en implementatiefase minder hard dan op papier stond. Zo is er tijdens de onderzoeksfase gewerkt aan een testopzet waarin verschillende oplossingen getest konden worden.

Implementeren

Zoals er in de onderzoeksfase al is begonnen aan een testopzet, is er in de implementeerfase ook nog onderzoek verricht. Waar aanvankelijk gedacht werd dat er aan één oplossing gewerkt zou worden, is er gaandeweg voor gekozen om in de eerder genoemde testopzet meerdere *proof of concepts* te bouwen om zo te kijken wat het beste werkt voor Avisi. Dankzij de testopzet konden mogelijke oplossingen makkelijk uitgeprobeerd worden. Uiteindelijk bleek één oplossing het best aan te sluiten bij de wensen van Avisi en is deze verder uitgewerkt in de definitieve PoC. Ook is er tijdens deze fase besloten dat het project meer op een advies moest richten dan het realiseren van een product, omdat de ontwikkelde oplossing te weinig uitdagend programmeerwerk bevatte om de competentie *realiseren* mee aan te tonen.

Testen

Tijdens de testfase is geen testplan, testrapport of gebruikershandleiding geschreven zoals eerder gepland was. De must-have requirements zijn getest en in het technische ontwerp verantwoord. Er zijn verschillende scenario's nagespeeld om te testen of de PoC naar behoren werkt maar dit is niet in een apart document vastgelegd. In een toelichting van de procesflow is aan de hand van *use-cases* uitgelegd hoe de PoC te gebruiken is, maar hier is geen apart document aan toegewijd.

Afronden

Het afronden van het project verliep zoals verwacht, hoewel er wel veel aandacht besteed moest worden aan de documentatie. Er is bijvoorbeeld onderzoek gedocumenteerd in dit document wat niet in het onderzoeksrapport stond. Dit is later rechtgetrokken.

7.2 Wekelijkse evaluatie

De wekelijkse evaluaties zijn altijd verlopen zoals eerder beschreven. Met een terugblik op de afgelopen week en het delen van de plannen voor de komende week kon de bedrijfsbegeleider goed in de gaten houden of het project de goede richting opging. Hoewel eerder werd voorgenomen om iedere evaluatie te documenteren en in de bijlagen op te nemen, is hier uiteindelijk van afgezien, omdat dit geen toegevoegde waarde meer had. Aanvankelijk werd gedacht dat hiermee het proces gecontroleerd kon worden, maar omdat iedere evaluatie over de periode van één week ging, kon deze controle ook zonder extra documentatie.

7.3 Maandelijkse oplevering tussentijdse documenten

Het ter controle aanbieden van documentatie bij de bedrijfsbegeleider liep niet volgens de geplande dagen, maar veel vloeiender. Wanneer er een stuk klaar voor review was werd dit opgestuurd en werd er feedback gegeven. Dit document werd maandelijks opgeleverd bij de afstudeerdocent. De eerste paar opleveringen was het grootste punt van feedback dat er meer geschreven moest worden. In de periode tussen de concept-oplevering en definitieve oplevering is de afstudeerdocent tweemaal gevraagd om feedback en werd hierin voorzien.

7.4 Testers-guild

Het bijwonen van de testers-guild bijeenkomsten bleek een ontzettend nuttige manier om feedback te krijgen over het project. Dit kwam voornamelijk doordat er iedere sessie tijd beschikbaar werd gesteld om over de werkzaamheden aan dit project te vertellen. Hierbij werd door de testers kritisch gekeken of bepaalde onderdelen van de proof of concept wel gebruikt zouden worden, zodat er voor een andere optie gekozen zou kunnen worden wanneer dit niet het geval was. Een ander bijkomend voordeel was de enthousiasme van de testers wanneer er een demo gegeven kon worden die een nieuw stuk van de realisatie liet zien. Dit enthousiasme is waardevol voor het vertrouwen in de oplossing. Het bevestigt dat er een product/oplossing gemaakt wordt die voldoet aan de wensen en behoeften van de eindgebruiker.

8. Opgeleverde resultaten

In de hoofdstukken hieronder staan de belangrijkste bevindingen per deelproduct dat tijdens de afstudeerperiode is opgeleverd. De volledige documenten zijn in de bijlagen opgenomen waar aangegeven.

8.1 Plan van aanpak

Het plan van aanpak (PvA) is in de eerste drie weken van de afstudeerperiode opgesteld. Het PvA was bedoeld om tot een duidelijke opdrachtomschrijving te komen en om na te denken over hoe de betreffende afstudeeropdracht zou worden uitgevoerd. Het PvA diende ook als basis voor het eerste gesprek van de afstudeerdocent met de afstudeerder en bedrijfsbegeleider. Het PvA heeft een helder beeld gegeven van het project en de manier waarop dit gaat worden aangepakt. Dit is gedaan door zaken erin op te nemen als: de aanleiding, probleemstelling, hoofd- en deelvragen, onderzoeksmethodiek en werkwijze. Deze zijn in dit document ook gebruikt ter toelichting van een aantal onderdelen van het project. Het volledige plan van aanpak is opgenomen in bijlage A.

8.2 Onderzoeksplan

Het onderzoeksplan was bedoeld om in kaart te brengen wat de hoofd- en deelvragen waren, waarom deze vragen relevant waren en hoe ze beantwoord diende te worden. Het opgeleverde product gaf een houvast om tot een goed uitgevoerd onderzoek te komen en maakte voor de begeleiders inzichtelijk hoe het verloop van het onderzoek eruit zou zien.

8.2.1 Vraagstelling en aanpak

Hoofdvraag

De hoofdvraag die met dit onderzoek beantwoord diende te worden luidde als volgt: "Hoe kan Avisi de kwaliteit van haar webapplicaties beter waarborgen door middel van visueleregressietesten?". Deze vraag is beantwoord door eerst een aantal deelvragen te beantwoorden en een Proof of Concept te maken.

Deelvragen

Om de hoofdvraag te beantwoorden zijn eerst de hieronder genoemde deelvragen beantwoord. Deze deelvragen waren bedoeld om te analyseren wat de wensen van Avisi zijn en welke mogelijke oplossingen er reeds bestaan.

Het onderzoek was op te delen in de volgende vier deelvragen:

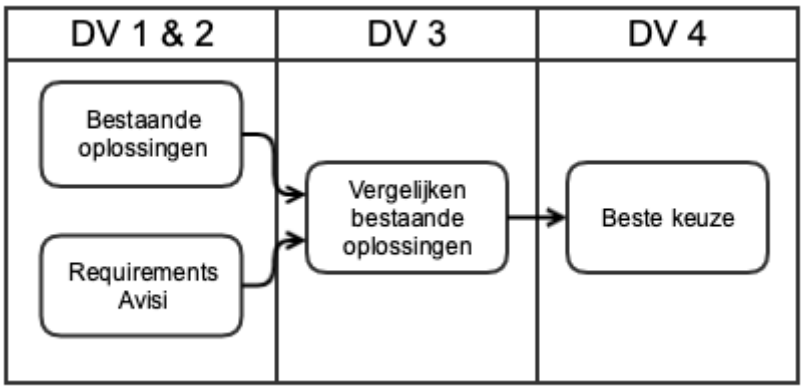
1) Welke frameworks voor het testen van visuele veranderingen zijn er?

2) Welke eisen heeft Avisi met betrekking tot visueleregressietesten?

3) In welke mate voldoen bestaande oplossingen aan de door Avisi gestelde eisen?

4) Wat is voor Avisi de beste oplossing met betrekking tot visueleregressietesten?

Hoewel de opdracht in eerste instantie bestond uit drie facetten is gezien de complexiteit besloten om de scope te verkleinen. In afstemming met de productowner (bedrijfsbegeleider) is besloten om onderzoek naar toegankelijkheidstesten pas te doen als hier genoeg tijd voor blijkt te zijn. Qua functionele testen wordt er geïnventariseerd welke frameworks nu gebruikt worden en of deze in combinatie met oplossingen voor de visueleregressietesten te gebruiken zijn.



Afbeelding 4: Diagram onderzoeksplan

Aanpak

Hieronder staat per deelvraag hoe deze zijn beantwoord met een aantal methodes uit het DOT-framework ("*ICT research methods*". (z.d.)).

1) Op het moment wordt binnen Avisi nog niet getest op visuele veranderingen, daarom moest onderzocht worden op welke manier dit gedaan kan worden en of er frameworks bestaan die dit kunnen. De methodes die hier zijn toegepast zijn: **literatuurstudie**, **available product analysis**.

2) Om in kaart te brengen wat Avisi wil bereiken en wat de wensen zijn met betrekking tot visueleregressietesten, werd met verschillende werknemers overlegd. De methodes die hier zijn toegepast zijn: **explore user requirements**, **interview**, **brainstorm**.

3) Nadat de requirements duidelijk waren en er een aantal mogelijke oplossingen werden gevonden is een matrix opgesteld waarmee duidelijk werd gemaakt welke oplossingen aan de criteria voldoen. De methode die hier is toegepast is: **multi-criteria decision making**.

4) Tot slot kon na het opstellen van de matrix met opties een keuze worden gemaakt voor de best passende oplossing. De methode die hier is toegepast is: **multi-criteria decision making**.

Methodes

Literatuurstudie: Literatuuronderzoek wordt uitgevoerd om je kennis over bepaalde onderwerpen te vergroten. Je maakt een lijst met trefwoorden waarmee je op het internet of in boeken op zoek kan gaan naar theorie.

Available product analysis: Met *available product analysis* doe je onderzoek naar bestaande producten. Zo kun je erachter komen hoe anderen (een deel van) jouw probleem hebben opgelost en of je misschien onderdelen van anderen kan gebruiken in plaats van alles zelf maken.

Explore user requirements: Door met de gebruikers van jouw oplossing in gesprek te gaan kun je achterhalen wat de verwachtingen en wensen zijn waar je oplossing aan moet voldoen.

Interview: Aan de hand van een *interview* kun je meer te weten komen over de stakeholders die betrokken zijn bij het project. Door iedereen een kans te geven om hun mening te geven, krijg je zelf input om je product te verbeteren en weet je beter waar je op moet letten.

Brainstorm: Een *brainstorm* sessie is handig om ideeën op te doen. Iedereen mag ideeën geven en alles moet geaccepteerd worden, het filteren wordt later gedaan. Het fijne aan een brainstorm is dat je gebruik kunt maken van de creativiteit en perspectieven van anderen, omdat deze verschillen van de jouwe.

Multi-criteria decision making: Wanneer er een aantal alternatieve oplossingen zijn, kan er een weging gegeven worden aan een aantal criteria uit de requirements. Vervolgens kunnen de oplossingen kwantitatief getoetst worden, door te tellen hoe zij scoren op de criteria met weging en zo kan de beste oplossing bepaald worden.

8.3 Onderzoeksrapport

In dit hoofdstuk komen de resultaten van het onderzoek naar voren. Het volledige onderzoeksrapport is opgenomen in bijlage B.

8.3.1 Welke frameworks voor het testen van visuele veranderingen zijn er?

Tijdens het onderzoek naar bestaande frameworks werd ontdekt dat er naast integrale oplossingen, die alles met betrekking tot de visueleregressietesten doen, ook composietoplossingen bestaan die voor een aantal verschillende onderdelen van visueleregressietesten losse frameworks / programma's gebruiken ("Visual Regression testing with Cypress.io — Xebia Blog", 2017). Zodoende wordt in deze deelvraag een onderscheid gemaakt in onderzoek naar integrale oplossingen en onderdelen van composietoplossingen.

Voor het beantwoorden van deze deelvraag is er gedurende een week gezocht naar alle frameworks die oplossingen bieden op het gebied van visueleregressietesten. Van de meest interessante oplossingen is een lijst opgesteld met een korte beschrijving van ieder product en wat het doet.

Integrale oplossingen

De integrale VRT-oplossingen zijn complete frameworks die gericht zijn op VRT. Een aantal hiervan komen met een *user-interface* (UI) en zijn direct aan te sluiten op CI/CD systemen.

Applitools Eyes

Applitools Eyes is een product dat gespecialiseerd is in geautomatiseerde visueleregressietesten. Eyes integreert met een groot aantal functionele test frameworks waardoor dit makkelijk geïntegreerd kan worden met functionele testen. Eyes werkt alleen in de cloud, wat ervoor zorgt dat de AI die ze gebruiken voor de vergelijkingen snel kan testen. ("Applitools Eyes", z.d.).

Screenster.io

Screenster.io is een tool die door middel van *record and playback* stappen van een test na één keer doorklikken kan herhalen. Op deze manier worden screenshots van het testproces gemaakt. Deze screenshots worden vergeleken met de screenshots die eerder als *baseline* zijn aangegeven. ("Visual Regression Testing", z.d.).

Argus Eyes

Argus Eyes is een Node *command line interface* (CLI) tool die met behulp van JSON bestanden screenshots maakt van pagina's op aangegeven URLs. Na het maken van screenshots kan Argus Eyes deze vergelijken met eerder gemaakte screenshots om verschillen te markeren. ("Argus Eyes", z.d.).

Composiet: testrunner

Eén van de onderdelen van composietoplossingen is de testrunner. Deze is verantwoordelijk voor het geautomatiseerd besturen van een browser en het maken van screenshots. De volgende testrunners zijn onderzocht:

Cypress.io

Cypress.io is een opensource end-to-end test framework voor "alles wat in een browser draait". Wanneer tests lokaal uitgevoerd worden kan dit in een GUI weergegeven worden, waarin bijvoorbeeld snapshots van losse teststappen getoond kunnen worden. Naast de opensource testrunner biedt Cypress.io ook een betaald dashboard aan dat makkelijk integreerbaar zou moeten zijn in CI/CD systemen . ("JavaScript End to End Testing Framework", z.d.)

Puppeteer

Puppeteer is een high-level Node library die gebruikt kan worden om (headless) Chrome aan te sturen. Puppeteer is geen test-framework op zichzelf, maar kan onder water worden gebruikt door

test-frameworks ter besturing van geautomatiseerde UI testen. (“Puppeteer | Tools for Web Developers | Google Developers”, 2019).

Fitnesse

Fitnesse is een opensource wiki webserver waarin de specificaties van een applicatie beschreven kunnen worden die ook door Fitnesse getest kunnen worden. (“FrontPage”, z.d.)

Composiet: afbeeldingvergelijker

Een ander significant onderdeel van composietoplossingen is de afbeeldingvergelijker. Deze is verantwoordelijk voor het vergelijken van screenshots die door de testrunner zijn gemaakt. Bij het vergelijken van twee screenshots moet een derde afbeelding gegenereerd worden waarin de verschillen tussen de screenshots zijn gehighlight (diff-afbeelding).

Pixelmatch

Pixelmatch is een kleine en snelle JavaScript afbeelding vergelijkende library die op pixel niveau afbeeldingen kan vergelijken. Het heeft geen verdere afhankelijkheden. Pixelmatch kan in Node, browsers en in CLI gebruikt worden. Naast het genereren van een diff-afbeelding print Pixelmatch ook een aantal statistieken van de vergelijking uit, zoals het aantal verschillende pixels en een percentage van verschillende pixels. (“npm: pixelmatch”, 2019)

ImageMagick

ImageMagick is een open-source softwaresuite voor het weergeven, omzetten en bewerken van afbeeldingen. Zo kan het ook gebruikt worden om twee afbeeldingen met elkaar te vergelijken op pixel niveau en een diff-afbeelding te genereren die de gevonden verschillen aangeeft. (“ImageMagick”, z.d.)

Blink-diff

Blink-diff is een JavaScript library die afbeeldingen kan vergelijken (alleen .png formaat). Blink-diff kan als JavaScript object of als CLI programma gebruikt worden. Verder heeft Blink-diff de optie om secties op basis van coördinaten te negeren bij de vergelijking. (“yahoo/blink-diff”, z.d.)

8.3.2 Welke eisen heeft Avisi met betrekking tot visueleregressietesten?

Deze deelvraag is in twee fases beantwoord door middel van twee meetings met een tester en developer van het Nobilex team. In de eerste fase is een MoSCoW lijst opgesteld met de eisen waar de oplossing aan moet voldoen en in de tweede fase is er een weging gegeven aan de verschillende eisen. Dit is gedaan om in het volgende stadium van het onderzoek een rangschikking op te stellen die de beste oplossing moet aangeven. Voor de weging van de requirements wordt een schaal gebruikt van 0 tot 5 met een extra optie ∞, wat staat voor *must have*. Voor de integrale oplossingen gelden alle requirements. Voor de composiet onderdelen slechts een deel (deze staan aangegeven met een X in de betreffende kolom). Hoe belangrijker de requirement hoe hoger het getal. Zoals wellicht opvalt is er één eis met weging 0. Dit komt omdat dit in de eerste ronde een harde wens was, maar in de tweede ronde werd besloten dat deze achterwege kon blijven.

	Requirement	Testrunner	Vergelijker	Weging
1	De oplossing moet testen op basis van screenshots	X	X	∞
2	De oplossing voor VRT moet los van andere testen te draaien zijn	X		∞
3	De oplossing moet data kunnen bewaren			∞
4	De oplossing moet actief onderhouden worden	X	X	∞
5	De oplossing werkt niet in de cloud	X	X	∞

	Requirement	Testrunner	Vergelijker	Weging
6	De oplossing heeft een proces voor het verwerken van screenshots			∞
7	De oplossing moet elementen van een pagina kunnen negeren	X		∞
8	Interactie met webpagina moet mogelijk zijn	X		∞
9	De VRT-oplossing deelt gluecode met functionele testen	X		∞
10	De oplossing geeft een overzicht van alle gemaakte screenshots			∞
11	De VRT moeten te draaien zijn in Google Chrome	X		∞
12	De VRT zijn handmatig te starten in Bamboo	X	X	5
13	De oplossing heeft een UI voor het weergeven van de screenshots			5
14	De oplossing kan screenshots maken van elementgroepen	X		5
15	De VRT moeten te draaien zijn in IE10/11, Edge	X		3
16	De oplossing ondersteunt verschillende viewports	X		3
17	De VRT zijn lokaal te draaien	X	X	2
18	De historie van visuele aanpassingen moet inzichtelijk zijn			1
19	De oplossing heeft een UI voor het verwerken van screenshots			1
20	De VRT zijn te draaien in overige browsers	X		1
21	De oplossing genereert een rapport met impact van visuele aanpassingen			0

Tabel 1: Requirements visueleregressietesten

Toelichting requirements

Hieronder staan de requirements verder uitgewerkt.

1) De oplossing moet testen op basis van screenshots

De oplossingen moeten op basis van screenshots regressietesten uitvoeren. Er zijn bijvoorbeeld ook oplossingen die de DOM en stylesheets vergelijken om veranderingen te vinden, maar de wens is om de regressies op te merken die een gebruiker met het blote oog kan waarnemen (uiterlijk van de webapplicatie, niet de code). Bovendien leidt een wijziging in de DOM en stylesheets niet per se tot een visuele verandering.

2) De oplossing voor visueleregressietesten moet los van andere testen te draaien zijn

De afweging was om de visueleregressietesten in de functionele testen te integreren of om hier een losstaande test van te maken. Het voordeel van geïntegreerde testen is dat beide testsoorten door de applicatie heen moeten lopen en dat je kunt voorkomen dat je dezelfde teststappen meerdere keren uit moet schrijven. Het is echter zo dat je alle testsoorten los van elkaar moet zien als aparte onderdelen. Op die manier kun je er bijvoorbeeld voor kiezen om de VRT niet uit te voeren wanneer deze bijvoorbeeld een te grote doorlooptijd hebben en de front-end van de applicatie niet

is aangeraakt. Zodoende is besloten dat de visueleregressietesten los van andere testen uit te voeren moeten zijn. Deze afweging heeft wel geleid tot eis 9 die later wordt toegelicht.

3) De oplossing moet data kunnen bewaren

Het moet mogelijk zijn om één geaccepteerde versie van de screenshots te bewaren. Hiernaast moet er per run van de VRT de afwijkende screenshots bewaard worden tot deze verwerkt zijn, omdat de optie moet bestaan dat de *baseline-afbeeldingen* geüpdatet moeten kunnen worden om de gemaakte veranderingen als nieuwe baseline te zien.

4) De oplossing moet actief onderhouden worden

Het framework en de onderliggende technologieën (e.g. webdriver) moeten nog onderhouden worden. Dit is om te voorkomen dat er gewerkt wordt met technieken die in de toekomst niet meer compatibel zijn met nieuwere versies van bijvoorbeeld Chrome. Dit betekent dat er niet gekeken wordt naar producten waarvan is aangegeven dat onderhoud gestopt is en producten die in het afgelopen jaar geen wijzigingen hebben gehad.

5) De oplossing werkt niet in de cloud

Met het oog op security en privacy is het gebruik van een oplossing die VRT uitvoert in de cloud uitgesloten.

6) De oplossing heeft een proces voor het verwerken van screenshots

Er moet een proces beschikbaar zijn om na de VRT, gemaakte screenshots te verwerken. Hiermee wordt bedoeld dat er de mogelijkheid moet zijn om een visuele wijziging als verwacht en zodoende geaccepteerd te markeren, waarna dit de nieuwe *baseline* wordt, of om een wijziging af te wijzen en een build te laten falen.

7) De oplossing moet elementen van een pagina kunnen negeren

Dynamische onderdelen van webpagina's die vaak veranderen, e.g. datums, moeten kunnen worden uitgesloten van de test. Bijvoorbeeld door in de html een tag of css class toe te voegen op elementen die *vóór* het maken van de screenshots onzichtbaar worden gemaakt. Wanneer dit niet mogelijk is zullen afbeeldingen met bijvoorbeeld een datum, altijd een verschil opmerken.

8) Interactie met webpagina moet mogelijk zijn

De VRT moeten net als een functionele test interactie met de applicatie kunnen hebben. Dit om bijvoorbeeld modals te openen of verkeerde input te geven om een error-melding te testen.

9) De VRT-oplossing deelt gluecode met functionele testen

Omdat de VRT qua testscenario's en interactie een subset zijn van de functionele testen, moeten ze een gluecode delen die ervoor zorgt dat je teststappen kan hergebruiken.

10) De oplossing geeft een overzicht van alle gemaakte screenshots

Na het uitvoeren van de VRT moet er gerapporteerd worden welke tests uitgevoerd zijn en welke gefaald/geslaagd zijn. Dit kan ook in de logging worden aangegeven.

11) De VRT moeten te draaien zijn in Google Chrome

Omdat het merendeel van het team met Google Chrome werkt en dit één van de moderne browsers is die allen grotendeels hetzelfde renderen is ervoor gekozen dat de VRT minimaal in Chrome te draaien moet zijn.

12) De VRT zijn handmatig te starten in Bamboo

De VRT moeten handmatig te starten zijn in Bamboo. Dit betekent dat de task voor het uitvoeren van de VRT handmatig te starten moet zijn. Dit komt ook goed van pas wanneer de VRT in eerste instantie gefaald zijn en deze na het accepteren van veranderingen, opnieuw uitgevoerd moeten

worden zonder de hele build opnieuw te hoeven doen. De weging 5 is gekozen omdat het de wens is om de tests achteraf uit te kunnen voeren op een build, maar dit is niet onmisbaar.

13) De oplossing heeft een UI voor het weergeven van de screenshots

In een UI worden de baseline screenshots en screenshots van de huidige testrun weergeven met ook een afbeelding die de verschillen highlight. Voor het gebruiksgemak zou het fijn zijn om direct een weergave te krijgen van alle gevonden verschillen, maar dit is niet onmisbaar zolang het VRT proces zelf goed werkt.

14) De oplossing kan screenshots maken van elementgroepen

Er is een mogelijkheid om in een testscenario aan te geven dat er slechts van een specifiek (HTML-)element een screenshot moet worden gemaakt. Zo kunnen bijvoorbeeld de statische onderdelen die over alle pagina's gelijk zijn, e.g. headers, footers buiten de vergelijking gelaten worden.

15) De moeten te draaien zijn in IE10/11, Edge

Naast dat de VRT uit te voeren zijn in Chrome is het interessant om op Internet Explorer 10/11 en Edge te testen, omdat deze browsers nog steeds veel gebruikt worden.

16) De oplossing ondersteunt verschillende viewports

Hoewel de Nobilex applicatie alleen op desktops wordt gebruikt, kan het voor andere teams interessant zijn om ook VRT op mobiele viewports toe te passen.

17) De VRT zijn lokaal te draaien

Naast dat de VRT op Bamboo te draaien zijn kunnen de developers lokaal ook tegen de baseline testen of er visuele regressies zijn opgetreden. Door dit te doen kunnen zij zelf al zien of er dingen veranderd zijn die niet verwacht waren, zonder dat iemand daar ook naar hoeft te kijken.

18) De historie van visuele aanpassingen moet inzichtelijk zijn

De screenshots die als baseline worden geaccepteerd moeten een bepaalde tijd bewaard blijven, ook als ze overschreven worden. Wanneer de oude versies van de *baseline* bewaard blijven, zou er inzichtelijk gemaakt kunnen worden wanneer een aanpassing is goedgekeurd, echter, dit is niet in de eerste plaats het doel van de VRT en daarom heeft dit criterium een lage prioriteit.

19) De oplossing heeft een UI voor het verwerken van screenshots

Het verwerken van de visuele verandering moet in een UI kunnen gebeuren. Dit criterium is er om het proces genoemd in criterium 6 makkelijker te maken en heeft een lage prioriteit omdat het proces zelf belangrijker is.

20) De VRT zijn te draaien in overige browsers

Nadat de VRT in Chrome, IE en Edge gedraaid kan worden kan deze ook in overige browser gedraaid worden, e.g. Safari.

21) De oplossing genereert een rapport met impact van visuele aanpassingen

Na het uitvoeren van de VRT moet een rapportage gemaakt worden die kwantitatief aangeeft hoeveel impact de pull-request heeft op het uiterlijk van de webapplicatie.

8.3.3 In welke mate voldoen de bestaande oplossingen aan de door Avisi gestelde eisen?

In de onderstaande tabel staan de uit de eerste deelvraag gevonden oplossingen gewogen aan de hand van de in de tweede deelvraag opgestelde criteria.

Integrale oplossingen

	Applitools Eyes	Screenster.io	Argus Eyes
De oplossing moet testen op basis van screenshots (∞)	Ja	-∞	Ja
De oplossing voor VRT moet los van andere testen te draaien zijn (∞)	Ja	Ja	Ja
De oplossing moet data kunnen bewaren (∞)	Ja	Ja	-
De oplossing moet actief onderhouden worden (∞)	Ja	Ja	-
De oplossing werkt niet in de cloud (∞)	-∞	Ja	Ja
De oplossing heeft een proces voor het verwerken van screenshots (∞)	Ja	-	-
De oplossing moet elementen van een pagina kunnen negeren (∞)	Ja	-	-
Interactie met webpagina moet mogelijk zijn (∞)	Ja	Ja	-∞
De VRT-oplossing deelt gluecode met functionele testen (∞)	Ja	-∞	-∞
De oplossing geeft een overzicht van alle gemaakte screenshots (∞)	Ja	Ja	-
De VRT moeten te draaien zijn in Google Chrome (∞)	Ja	Ja	-
De VRT zijn handmatig te starten in Bamboo (5)	-	Ja	-
De oplossing heeft een UI voor het weergeven van de screenshots (5)	Ja	Ja	-
De oplossing kan screenshots maken van elementgroepen (5)	Ja	-	-
De VRT moeten te draaien zijn in IE10/11, Edge (3)	Ja	-	-
De oplossing ondersteunt verschillende viewports (3)	Ja	-	-
De VRT zijn lokaal te draaien (2)	Nee	-	-
De historie van visuele aanpassingen moet inzichtelijk zijn (1)	Ja	-	-
De oplossing heeft een UI voor het verwerken van screenshots (1)	Ja	-	-
De VRT zijn te draaien in overige browsers (1)	Ja	-	-
De oplossing genereert een rapport met impact van visuele aanpassingen (0)	-	-	-
Totale weging	**-∞**	**-∞**	**-∞**

Tabel 2: Beoordeling integrale oplossingen

Om niet te veel tijd te besteden aan het onderzoeken van integrale oplossingen die niet voldeden aan must-have requirements is er geen volledige toetsing geweest wanneer aan zulke requirements niet voldaan werd. De conclusie is dat er geen geschikte integrale oplossingen zijn gevonden. De reden dat de bovenstaande oplossingen zijn opgenomen in dit verslag was dat deze oplossingen inspiratie gaven voor VRT in het algemeen of een functionaliteit die in de gebouwde oplossing zit. Zo was de *record and playback* functie van Screenster.io uniek. Argus Eyes had gedocumenteerd hoe zij onderwater afbeeldingen vergelijken. Dit proces komt overeen met de gebouwde oplossing.

Composiet: testrunner

	Cypress.io	Puppeteer	Fitnesse
De oplossing moet testen op basis van screenshots (∞)	Ja	Ja	Ja
De oplossing voor VRT moet op zichzelf te draaien zijn (∞)	Ja	Ja	Ja
De oplossing moet actief onderhouden worden (∞)	Ja	Ja	Ja
De oplossing werkt niet in de cloud (∞)	Ja	Ja	Ja
De oplossing moet elementen van een pagina kunnen negeren (∞)	Ja	-∞	-∞
Interactie met webpagina moet mogelijk zijn (∞)	Ja	Ja	Ja
De VRT-oplossing deelt gluecode met functionele testen (∞)	Ja	-∞	Ja
De VRT moeten te draaien zijn in Google Chrome (∞)	Ja	Ja	Ja
De VRT zijn handmatig te starten in Bamboo (5)	Ja	Ja	Ja
De oplossing kan screenshots maken van elementgroepen (5)	Ja	Ja	-5
De VRT moeten te draaien zijn in IE10/11, Edge (3)	-3	-3	-
De oplossing ondersteunt verschillende viewports (3)	Ja	Ja	-
De VRT zijn lokaal te draaien (2)	Ja	Ja	-
De VRT zijn te draaien in overige browsers (1)	-1	-1	-
Totale weging	**-4**	**-∞**	**-∞**

Tabel 3: Beoordeling composiet testrunner

Omdat Fitnesse op dit moment door Nobilex wordt gebruikt als functionele testframework is onderzocht of deze ook als testrunner voor de VRT in te zetten was. Dit zou het in gebruik nemen van VRT makkelijk maken voor Nobilex. Het bleek echter zo dat Fitnesse niet toereikend was voor alle must-have requirements. Daarom is gezocht naar andere testrunners en kwam Cypress.io als beste uit de vergelijking.

Composiet: afbeeldingvergelijker

	Pixelmatch	ImageMagick	Blink-diff
De oplossing moet testen op basis van screenshots (∞)	Ja	Ja	Ja
De oplossing moet actief onderhouden worden (∞)	Ja	Ja	-∞
De oplossing werkt niet in de cloud (∞)	Ja	Ja	Ja

	Pixelmatch	ImageMagick	Blink-diff
De VRT zijn handmatig te starten in Bamboo (5)	Ja	Ja	Ja
De VRT zijn lokaal te draaien (2)	Ja	Ja	Ja
Totale weging	**+7**	**+7**	**-∞**

Tabel 4: Beoordeling composiet afbeeldingvergelijker

Omdat de drie oplossingen een gelijke score hebben gekregen, zijn er performance testen uitgevoerd waar gemeten is hoe snel ze een verzameling afbeeldingen kunnen vergelijken. Hierbij is ook gekeken naar de schaalbaarheid door te meten hoe lang het duurt om 50, 100 en 200 afbeeldingen te verwerken. Iedere test is vijf keer uitgevoerd en voor iedere run is de map met de door de tool gegenereerde afbeeldingen geleegd. De afbeeldingen zijn afkomstig van unsample.net en hebben allemaal een afmeting van 1920x1080. Op 10% van de afbeeldingen zijn bewerkingen gedaan om regressies te introduceren. Deze tests zijn enkel bedoeld om een beeld te krijgen van welke afbeeldingvergelijker het snelst is. Blink-diff is ook getest omdat deze wellicht het snelst zou zijn. Als dit het geval was kon de requirement "De oplossing moet actief onderhouden worden" mogelijk buiten beschouwing gelaten worden voor de afbeeldingvergelijkers. De gemiddelde tijdsduur van de vijf metingen per verzameling afbeeldingen per vergelijker zijn opgenomen in de volgende tabel:

Run met x afbeeldingen	Pixelmatch	Blink-diff	ImageMagick
x = 50	52,8 sec	84 sec	102,6 sec
x = 100	106,2 sec	167,8 sec	212,8 sec
x = 200	208,4 sec	336,4 sec	410,2 sec

Tabel 5: Gemiddelde tijdsduur van de afbeeldingvergelijker performance testen

Uit deze meetresultaten blijkt dat Pixelmatch het snelst is en deze wordt daarom gebruikt als afbeeldingvergelijker in de gebouwde oplossing. Qua schaalbaarheid lijkt de verwerkingstijd van alledrie de tools lineair te schalen met het aantal afbeeldingen.

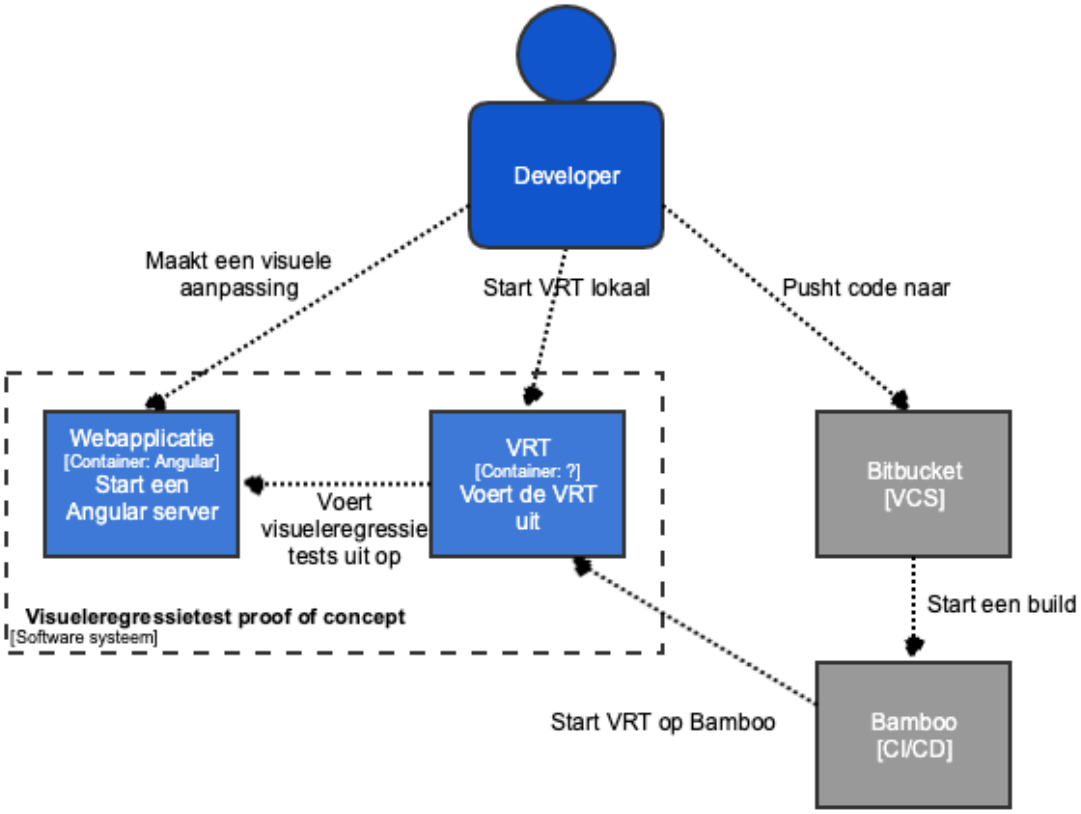
8.3.4 Wat is voor Avisi de beste oplossing met betrekking tot visueleregressietesten?

Omdat er geen integrale oplossing gevonden werd die voldeed aan alle must-have requirements van Avisi is gekozen om een composietoplossing *proof of concept* (PoC) te ontwikkelen. Hierbij is Cypress.io gekozen als testrunner en Pixelmatch als afbeeldingvergelijker. In het volgende hoofdstuk wordt deze PoC toegelicht.

De testrunner van Cypress.io wordt aangeboden onder de MIT open-source licentie (“JavaScript e2e Testing.”, z.d.). Pixelmatch wordt aangeboden onder de ICS open-source licentie (“npm: pixelmatch”, 2019).

8.4 Proof of concept

Gedurende het project is gewerkt aan een proof of concept (PoC) waarmee aangetoond moest worden dat een oplossing voor VRT zou werken in een zelfde soort omgeving als de Nobilex applicatie. Hierbij is een modulair project opgezet waarin bepaalde onderdelen eenvoudig te vervangen zijn om deze afzonderlijk te kunnen testen. Voor de PoC is een repository op Bitbucket ingeregeld die gekoppeld is aan een project op Bamboo. Er is gekozen voor deze opzet om het landschap van de Nobilex applicatie en haar ontwikkelstraat te simuleren.



Afbeelding 5: Container diagram PoC

In het bovenstaande container diagram, die de globale opzet van de PoC weergeeft, staan twee containers in het softwaresysteem *Visueleregressietest proof of concept*. Dit zijn containers voor een Angular2 webapplicatie en een VRT-oplossing. In werkelijkheid zijn in de PoC project meerdere containers voor verschillende oplossingen gebouwd, om een aantal opties afzonderlijk te testen.

8.4.1 Componenten

De PoC bestaat uit de volgende componenten:

Webapplicatie

De webapplicatie is simpelweg de demo-app van Angular2 waaraan twee dingen zijn toegevoegd om bepaalde eisen te kunnen testen:

1. De webapplicatie heeft een paragraaf waarin de huidige datum en tijd weergeven worden. Dit helpt met het toetsen van eis 7 uit de lijst met eisen, omdat de tijd bij iedere screenshot anders zou zijn en zodoende altijd een visuele regressie zou opleveren. Wanneer een oplossing deze paragraaf kan negeren voldoet het aan de eis.
2. De webapplicatie heeft een text-input veld waar de gebruiker c.q. testrunner iets in kan vullen, waarna de tekst onder de input-box in een label weergeven wordt. Dit is gedaan om te toetsen

of een oplossing aan eis 8 voldoet, dus of interactie met de webpagina mogelijk is. Tevens kan de styling van het label eenvoudig aangepast worden om te testen of de oplossing regressies detecteert.

Er is gekozen voor een demo-app in Angular2, omdat Nobilex ook hierin is gemaakt en de VRT die werkt op de demo-app ook zou moeten werken op Nobilex.

VRT

De VRT-container bevat een testrunner die voldoet aan de eisen genoemd in *Tabel 3: Beoordeling composiet testrunner,* en de afbeeldingenvergelijker *Pixelmatch*, omdat deze als beste uit de tests kwam. Wanneer de container van de webapplicatie is opgestart kan de container met de VRT opgestart worden om testen uit te voeren op de webapplicatie. In deze container wordt ook de baseline opgeslagen in de repository. Er zijn op het moment van schrijven twee VRT containers die allebei een andere oplossing testen:

1. cypressVRT: deze oplossing kwam als totaal het beste uit de testen en wordt daarom ook aangeraden. De oplossing heeft de standaard Cypress.io mappenstructuur, uitgebreid met de mappen *screenshot_base* en *screenshot_diff*. De map *screenshot_base* bevat alle baseline-afbeeldingen en de map *screenshot_diff* wordt gebruikt om de diff-afbeeldingen die door Pixelmatch gegenereerd worden naar toe te schrijven. Met deze afbeeldingen kunnen developers lokaal zien welke visuele regressies zijn gevonden.
2. Puppeteer: deze container is in een vroeg stadium gebruikt om te experimenteren met een webdriver en een mogelijke user-interface. Omdat Puppeteer op zichzelf niet gemaakt is als test-framework zou het gebruik van Puppeteer als testrunner omslachtig zijn. Deze conclusie heeft ertoe geleid dat er niet verder is ontwikkeld aan deze oplossing.

Bitbucket

Avisi gebruikt Atlassian Bitbucket als versiebeheersysteem (VCS). Op Bitbucket maken developers een *pull-request* aan wanneer zij een feature afhebben. Bitbucket kan echter ook veranderingen in afbeeldingen detecteren en highlighten. Dit is ontzettend nuttig, omdat er zo baseline-afbeeldingen in de repository van een project opgeslagen kunnen worden en dat aanpassingen op deze afbeeldingen door andere developers/testers/UX-designers gereviewed kunnen worden.

Bamboo

Avisi gebruikt Atlassian Bamboo als CI/CD systeem. Wanneer er voor een branch een pull-request wordt gemaakt start Bamboo een build voor deze branch. In het buildplan kan worden aangegeven dat de VRT ook uitgevoerd moeten worden. Wanneer de developer is vergeten om de VRT lokaal te draaien maar wel visuele aanpassingen heeft, komt Bamboo hierachter en laat het de build falen.

8.4.2 Procesflow

Om het gebruik van de PoC voor VRT te illustreren wordt het proces van drie use-cases in deze paragraaf toegelicht:

Als developer wil ik een baseline-afbeelding toevoegen, zodat deze kan worden meegenomen in de VRT

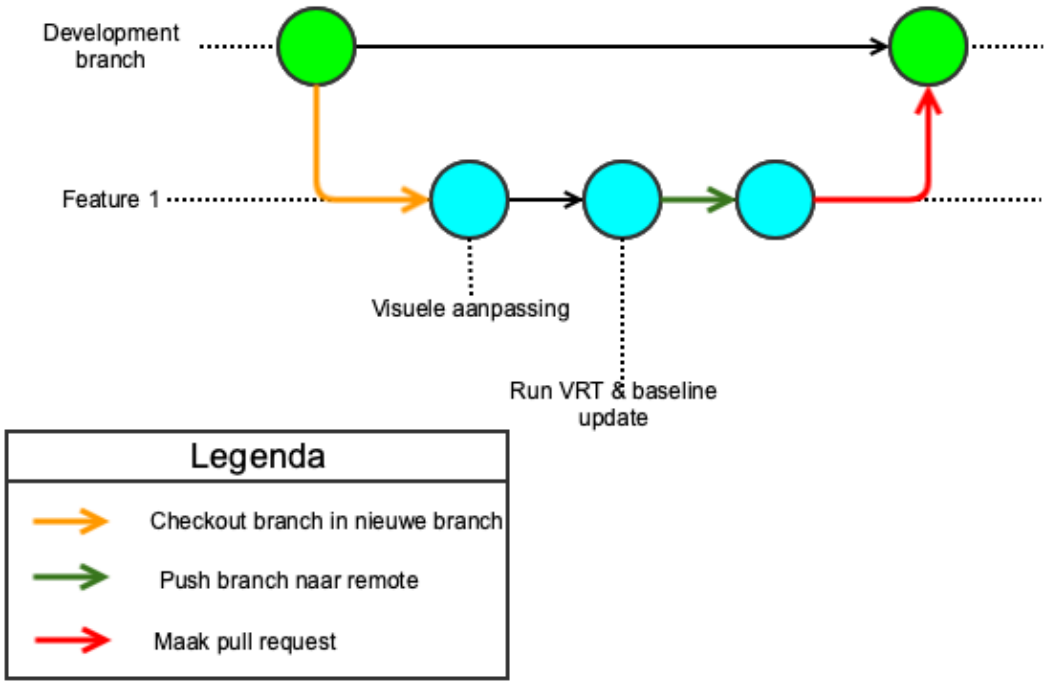
Wanneer een developer een nieuw scherm heeft gemaakt of een VRT wil toevoegen aan een bestaand scherm kan dat als volgt:

1. De developer pullt de development-branch.
2. De developer checkt de lokale development-branch uit naar een feature branch.
3. De developer schrijft de test voor het gewenste scherm.
4. De developer voert VRT uit. Hieruit komen de afbeeldingen die nodig zijn.
5. De developer verplaatst de afbeeldingen naar de baseline map (dit is de map die in de repository zit).
6. De developer commit en pusht de nieuwe afbeeldingen naar de remote feature branch.
7. De developer maakt een pull request.
8. Teamleden reviewen de baseline-afbeeldingen.
9. De developer merged de feature branch in de development-branch.

Als developer wil ik een visuele aanpassing laten accepteren, zodat de baseline *up-to-date* blijft

Wanneer in de toekomst aanpassingen gedaan worden op een scherm dat is opgenomen in de VRT , gaat het bijhouden van een correcte baseline als volgt:

1. De developer pullt de development-branch.
2. De developer checkt de lokale development-branch uit naar een feature branch.
3. De developer maakt de visuele aanpassing.
4. De developer voert VRT uit. Hieruit komen de afbeeldingen die nodig zijn voor de geüpdatete baseline.
5. De developer verplaatst de afbeeldingen met aanpassingen naar de baseline map (dit is de map die in de repository zit) en overschrijft hiermee de oude baseline. Door dit te doen accepteert de developer zijn nieuwe baseline. Als er onverwachte veranderingen zijn gevonden moet de code worden aangepast en stap 4 opnieuw worden uitgevoerd.
6. De developer pusht de code wijzigingen en baseline update naar remote feature branch.
7. De developer maakt een pull request.
8. Teamleden reviewen code en geüpdatete afbeeldingen. Wanneer de aanpassingen worden goedgekeurd accepteren de reviewers met deze stap de nieuwe baseline.
9. Bamboo voert de VRT uit ter controle of alle door de developer gemaakte wijzigingen zijn toegevoegd aan de baseline.
10. De developer merged de feature branch in de development-branch.

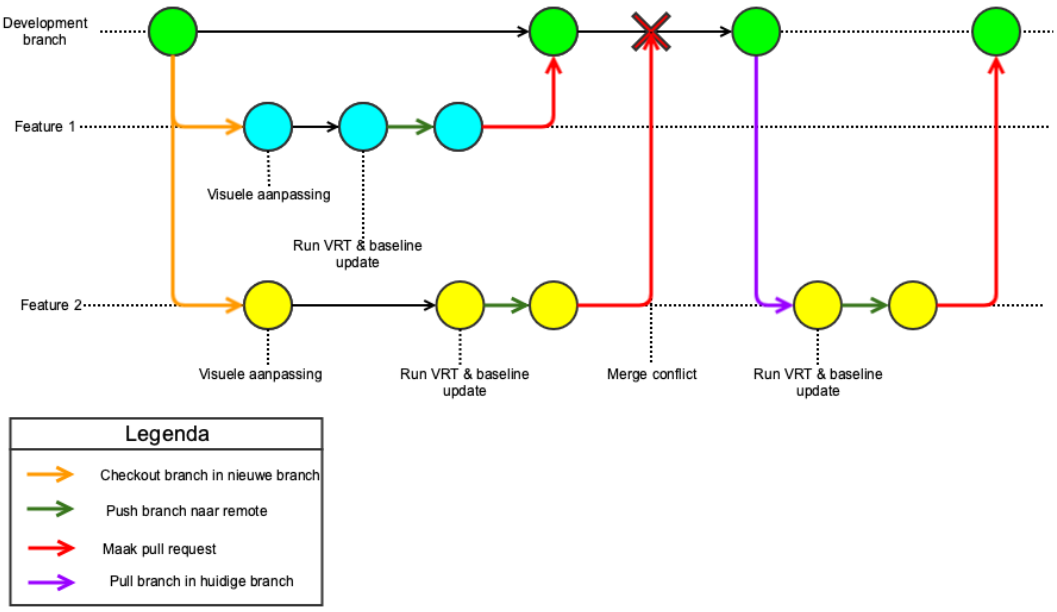


Afbeelding 6: GIT-flow visuele aanpassing op één branch

Als developer wil ik een visuele aanpassing laten accepteren op een scherm waar een andere developer ook visuele aanpassingen heeft gedaan, zodat de baseline *up-to-date* blijft

Het kan voorkomen dat meerdere developers tegelijk aan hetzelfde scherm werken en dus meerdere baseline updates maken. Om dit te laten werken moeten er wat extra stappen worden uitgevoerd. Het uitgangspunt van het volgende voorbeeld is dat *developer A* en *developer B* beide van dezelfde commit van de development-branch zijn afgetakt:

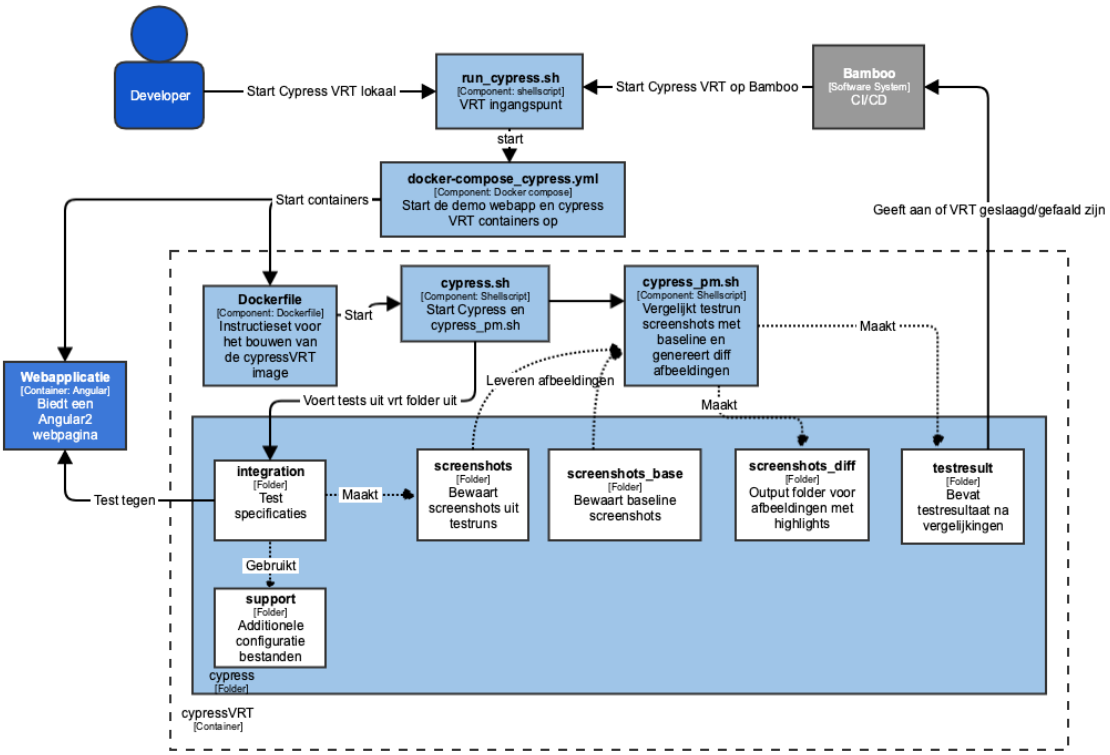
1. Developer A heeft de stappen uit het vorige voorbeeld uitgevoerd.
2. Developer B heeft vanaf dezelfde versie van de development-branch als developer A een aftakking gemaakt.
3. Developer B maakt een visuele aanpassing op hetzelfde scherm als developer A.
4. Developer B voert de VRT uit. Hieruit komen de afbeeldingen met zijn/haar wijzigingen, maar niet met de wijzigingen van developer A.
5. Developer B verplaatst de afbeeldingen naar de baseline map (dit is de map die in de repository zit) en overschrijft hiermee de oude baseline.
6. Developer B pusht de code wijzigingen en baseline update naar remote feature branch.
7. Developer B maakt een pull request.
8. Bitbucket geeft aan dat de afbeelding(en) die door zowel developer A als developer B zijn gewijzigd een merge-conflict opleveren.
9. Developer B pullt de meest recente versie van de development-branch in zijn/haar feature branch. Hierin zitten de wijzigingen van developer A en de meest recente baseline.
10. Developer B voert VRT uit. Doordat developer B nu ook beschikt over de wijzigingen van developer A, bevatten de afbeeldingen de visuele aanpassingen van beide developers.
11. Developer B verplaatst de afbeeldingen naar de baseline map en overschrijft hiermee de oude baseline.
12. Developer B pusht de code wijzigingen en baseline update naar remote feature branch.
13. Het oude pull-request wordt geüpdatet.
14. Teamleden reviewen code en geüpdatete afbeeldingen.
15. Bamboo voert VRT uit ter controle of alle door developer B gemaakte wijzigingen zijn toegevoegd.
16. Developer B merged de feature branch in de development-branch.



Afbeelding 7: GIT-flow visuele aanpassingen op meerdere branches

8.4.3 Technisch ontwerp cypressVRT container

Om een overzicht te geven van de architectuur van de cypressVRT container is het onderstaande *component diagram* gemaakt. Verder worden in deze paragraaf een aantal mappen en bestanden nader toegelicht om de technische opzet van de PoC te verduidelijken.



Afbeelding 8: Component diagram cypressVRT container

run_cypress.sh

Dit bestand is voor zowel Bamboo als de developer die de VRT lokaal wil draaien het startpunt. In de tweede regel worden de in *docker-compose_cypress.yml* gespecificeerde containers gestart. De -d flag zorgt ervoor dat de container op de achtergrond wordt uitgevoerd. Dit houdt de logging overzichtelijker. De until loop zorgt ervoor dat de containers die gestart zijn blijven bestaan tot dat er in de logging van de cypressVRT container "cypressVRT done" voorbij komt. Dit is een teken dat alle tests zijn uitgevoerd en dat de gemaakte screenshots zijn vergeleken met de baseline.

run_cypress.sh

```
1  #!/usr/bin/env bash
2  docker-compose -f docker-compose_cypress.yml up -d
3
4  until docker logs cypressvrt | grep "cypressVRT done"; do
5    sleep 1
6  done
7
8  docker-compose down
```

Code block 1: run_cypress.sh

docker-compose_cypress.yml

Het docker-compose bestand specificeert hoe het landschap van twee services eruit ziet, namelijk: cypressvrt en my-app. Hier is cypressvrt, zoals de naam doet vermoeden, de container die een VRT-oplossing biedt met Cypress.io als testrunner. De service my-app is een container waarin een voorbeeld Angular2 webapplicatie draait. Omdat beide services in hetzelfde compose bestand staan, zitten ze automatisch in hetzelfde netwerk en kan de testrunner de webapplicatie bereiken op http://my-app:4200/. Onder de cypressVRT service is een *depends_on* geconfigureerd die ervoor zorgt dat deze container pas gestart wordt zodra my-app gestart is. Dit is gedaan om te voorkomen dat de testen een pagina proberen te openen die nog niet gestart is. Tot slot worden voor beide services twee volumes ingesteld. Op regel 10 en 21 staan de host-volumes die ervoor zorgen dat de code uit het project in de container gemount worden. Ook worden de baseline-afbeeldingen toegankelijk voor de container en worden vice versa de door de container gegenereerde testrun screenshots en diff-afbeeldingen beschikbaar op de host-machine. Op regel 11 en 22 staat dezelfde anonieme volume. Deze is nodig om te voorkomen dat de *node_modules* die tijdens het bouwen van de images zijn geïnstalleerd worden overschreven met de lokale node-modules wanneer de container wordt opgestart (“Dockerizing an Angular App”, 2019).

docker-compose_cypress.yml

```
version: '3.5'

services:
  cypressvrt:
    container_name: cypressvrt
    build:
      context: ./cypressVRT
      dockerfile: Dockerfile
    volumes:
      - './cypressVRT:/usr/src/app'
      - '/usr/src/app/node_modules'
    depends_on:
      - my-app

  my-app:
    container_name: my-app
    build:
      context: ./my-app
      dockerfile: Dockerfile
    volumes:
      - './my-app:/usr/src/app'
      - '/usr/src/app/node_modules'
    ports:
      - '4200:4200'
```

Code block 2: docker-compose_cypress.yml

Dockerfile

Om de VRT in een docker-container te laten draaien moest er een base-image gekozen worden. Cypress.io heeft op hub.docker.com drie repositories: cypress/base, cypress/included en cypress/browsers ("Docker Hub", z.d.). Cypress/base is een image met alle benodigdheden voor Cypress.io, maar de testrunner en additionele browsers moeten hierop zelf geïnstalleerd worden. Cypress/included heeft de testrunner geïnstalleerd in de image, maar geen extra browsers. De image cypress/browsers gebruikt cypress/base als base image en installeert afhankelijk van de gekozen tag bijvoorbeeld, Chrome en/of Firefox in de image.

Dockerfile cypress

```
FROM cypress/browsers:node11.13.0-chrome73

RUN mkdir /usr/src/app
WORKDIR /usr/src/app

ENV PATH /usr/src/app/node_modules/.bin:$PATH

COPY package.json /usr/src/app/package.json
RUN npm install

COPY . /usr/src/app

CMD ./cypress.sh
```

Code block 3: Dockerfile

Zoals boven te zien is wordt *cypress/browsers:node11.13.0-chrome73* als base-image gebruikt. Er worden een aantal geneste mappen gemaakt waar later de sourcecode van de Cypress.io testen en baseline-afbeeldingen in gekopiëerd worden. De map */usr/src/app* wordt als working directory ingesteld. De bestanden uit de map */usr/src/app/node_modules/.bin* worden aan de variable *PATH* toegevoegd zodat ze vanuit de terminal uit te voeren zijn. Vervolgens wordt het bestand *package.json* naar de image gekopiëerd zodat het programma Pixelmatch en de Cypress.io testrunner in de volgende stap, *RUN npm install*, geïnstalleerd kunnen worden. Zoals boven

genoemd wordt nu de sourcecode, i.e. de testcode en baseline-afbeeldingen in de image gekopiëerd. Tot slot wordt de shellscript *cypress.sh* uitgevoerd.

cypress.sh

Wanneer de Cypress.io container wordt opgestart wordt het volgende script uitgevoerd:

cypress.sh

```
1  #!/usr/bin/env bash
2  npx cypress run --browser chrome --config video=false -s "cypress/integration/vrt/*"
3  ./cypress_pm.sh
4  chmod -R 777 cypress
5  echo "cypressVRT done"
```

Code block 4: cypress.sh

Het commando op regel 2 zorgt ervoor dat Cypress.io testen uitvoert. De *--browser chrome* flag geeft aan dat Chrome als browser gebruikt moet worden, zoals gewenst in requirement 11: "De VRT moeten te draaien zijn in Google Chrome". Deze browser wordt geleverd door de gebruikte base-image en hoeft niet eerst geïnstalleerd te worden. Omdat Cypress.io standaard opnames maakt van testen wanneer deze via de terminal worden uitgevoerd en deze functie niets toevoegt aan de vrt, is dit met de *--config video=false* flag uitgezet.

Met de *-s* flag wordt aangegeven dat alleen de specs uit de map *cypress/integration/vrt/* worden uitgevoerd. Op hetzelfde niveau als de map *vrt*, kan bijvoorbeeld een map gemaakt worden voor functionele testen. Door in een ander script deze map aan te geven met de -s flag kunnen de verschillende testsoorten eenvoudig apart worden uitgevoerd zoals gewenst met requirement 2: "De oplossing voor visueleregressietesten moet los van andere testen te draaien zijn". Het commando op regel 4 is nodig om Bamboo de benodigde rechten te geven om afbeeldingen uit de volumes te kunnen verwijderen. Wanneer de testen zijn uitgevoerd en de afbeeldingen vergeleken zijn wordt met de laatste regel de tekst "cypressVRT done" geëchoed, zodat *run_cypress.sh* weet dat de containers afgesloten kunnen worden.

cypress/integration

In de integration map worden alle testspecificaties geschreven. Dit kan in een zelfgekozen mappenstructuur, wat ervoor zorgt dat verschillende testsoorten (e.g. VRT/functioneel) gegroepeerd kunnen worden. In de PoC heeft deze map bijvoorbeeld een submap *vrt* waarin een testspec staat die de demo webapp opent en daar screenshots maakt. In de map *cypress/support* zit het bestand *index.js*. Deze kan gebruikt worden voor extra configuratie. In de PoC is deze gebruikt om een CSS-class aan te geven die bij het maken van screenshots genegeerd moeten worden, zoals gewenst volgens requirement 7: "De oplossing moet elementen van een pagina kunnen negeren".

cypress/screenshots

Dit is de map waar Cypress.io standaard screenshots in opslaat. Dit gaat conform de mappenstructuur in *cypress/integration*, waarbij iedere testspec een eigen map krijgt met daarin de screenshots die in die test gemaakt zijn.

cypress/screenshots_base

In deze map worden de baseline-afbeeldingen opgeslagen. Deze map zit ook in de repository van het project (wat betekent dat in de *.gitignore* is aangegeven dat deze map niet door GIT genegeerd moet worden). Hoewel er tussen verschillende developers verschillende versies van de bestanden in deze map kunnen bestaan, moet deze baseline gezien worden als een *Single Source of Truth*, omdat de baseline in de eerste plaats de developer moet helpen om veranderingen op zijn/haar branch waar te nemen. Zoals in de paragraaf Procesflow is benoemd kunnen verschillen van de baseline met de development-branch letterlijk worden rechtgetrokken, door de wijzigingen van den development-branch in de feature-branch te pullen. Hierna moeten de VRT opnieuw worden uitgevoerd om afbeeldingen te genereren die de veranderingen van beide branches weergeven. Om

het vergelijken van afbeeldingen eenvoudig te houden heeft deze map een structuur conform *cypress/screenshots*.

cypress/screenshots_diff

Pixelmatch vergelijkt twee afbeeldingen en genereert een nieuwe afbeelding waarin de gevonden verschillen gehighlight zijn. Deze afbeeldingen worden hier ook volgens de mappenstructuur van *cypress/screenshots* opgeslagen.

cypress_pm.sh

Dit script wordt gestart door *cypress.sh* wanneer de testen uit *cypress/integration/vrt* zijn uitgevoerd. Het doel van dit script is om met behulp van de afbeeldingvergelijker (i.e. Pixelmatch) alle screenshots die tijdens de VRT gemaakt zijn te vergelijken met de versie die in de baseline staat. Omdat de mappenstructuur van *cypress/screenshots*, *cypress/screenshots_base* en *cypress/screenshots_diff* hetzelfde zijn hoeft er alleen door *cypress/screenshots* gelust te worden. Wanneer een screenshot enkel in de map *cypress/screenshots* staat, verplaatst dit script de betreffende screenshot direct naar de baseline map. Wanneer een screenshot zowel in de *cypress/screenshots*- als de cypress/*screenshots_baseline* map staat worden deze met behulp van Pixelmatch met elkaar vergeleken en wordt de afbeelding met highlights naar *cypress/diff* geschreven. Pixelmatch print ook hoeveel procent van de pixels afweken. Wanneer dit 0% is wordt de gegenereerde diff-afbeelding verwijderd. Dit resulteert erin dat de developer een overzicht overhoudt van alle screenshots met een visuele verandering. Omdat deze vergelijking niet door Cypress.io wordt gedaan kon er geen gebruik gemaakt wordt van een door Cypress.io gegenereerd testrapport. Dit is opgelost door na het vergelijken van de afbeeldingen een check uit te voeren die kijkt of de map *cypress/screenshots_diff* leeg is. Wanneer deze map leeg blijkt te zijn wordt er een bestand, *vrtOK*, in de map *cypress/testresult* gemaakt die Bamboo kan laten weten dat de VRT geslaagd zijn.

cypress_pm.sh

```
#!/usr/bin/env bash
BASE=cypress/screenshots_base/vrt
RUN=cypress/screenshots/vrt
DIFF=cypress/screenshots_diff/vrt

for spec in "$RUN"/*; do
  SPECNAME=$(basename "$spec")
  for filename in "$RUN"/"$SPECNAME"/*; do
    SCRNAME=$(basename "$filename")
    if [ ! -f "$BASE"/"$SPECNAME"/"$SCRNAME" ]; then
      mkdir -p "$BASE"/"$SPECNAME"; mv "$filename" "$BASE"/"$SPECNAME"/"$SCRNAME";
      echo -e "\e[32mCreated new baseimage \e[0m" "$BASE"/"$SPECNAME"/"$SCRNAME";
    else
      mkdir -p "$DIFF"/"$SPECNAME";
      echo -e "\e[31mComparing image \e[0m" "$SPECNAME"/"$SCRNAME";
      if pixelmatch "$BASE"/"$SPECNAME"/"$SCRNAME" "$filename" "$DIFF"/"$SPECNAME"/"$SCRNAME" |
grep "error: 0%"; then
        rm "$DIFF"/"$SPECNAME"/"$SCRNAME";
      fi
    fi
  done
done

if  ! ls -R $DIFF | grep ".png"; then
    echo "No screenshots diffs found";
    mkdir -p cypress/testresults && touch cypress/testresults/vrtOK;
fi
```

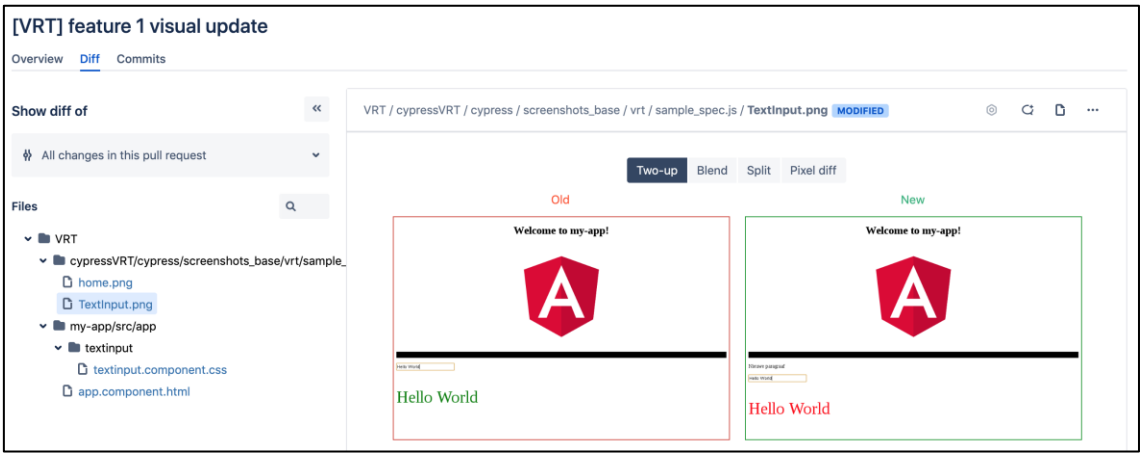
Code block 5: cypress_pm.sh

8.4.4 Integratie met Bitbucket en Bamboo

Om te verduidelijken hoe de VRT-oplossing van de PoC integreert met de CI/CD pipeline van Avisi worden de koppelingen met Bitbucket en Bamboo toegelicht.

Bitbucket

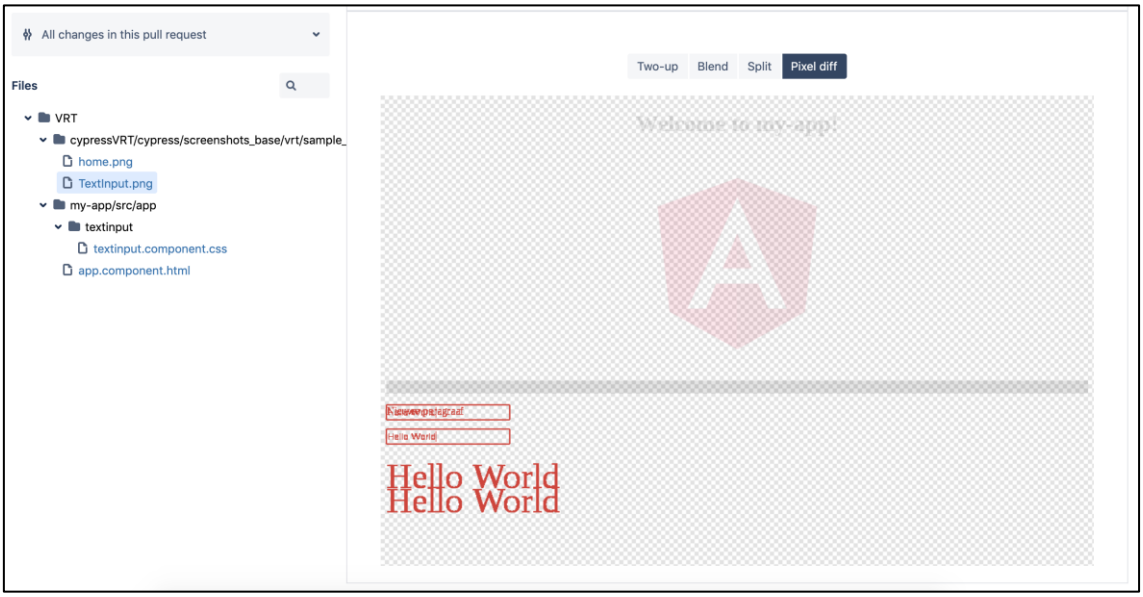
Bitbucket is de door Avisi gebruikte VCS. Door een VCS worden developers in staat gesteld om samen te werken aan code. Naast gemaakte wijzigingen in code, kan Bitbucket ook verschillen in afbeeldingen weergeven, hetgeen mooi integreert in het proces van de PoC. Zoals in de paragraaf *Procesflow* is genoemd worden gewijzigde afbeeldingen in de *screenshots_base* map opgenomen in de pull-requests. Het *Diff* tabblad van een dergelijke pull-request ziet er als volgt uit:



Afbeelding 9: Voorbeeld pull-request met visuele aanpassingen Two-up

Hier is aan de linkerkant duidelijk te zien welke bestanden zijn veranderd. Door op een afbeelding in de *screenshot_base* map te klikken kunnen reviewers aan de rechterkant eenvoudig zien wat er veranderd is.

Voor schermen waar de verschillen niet in één oogopslag te zien zijn heeft Bitbucket ook andere modi om de verschillen te highlighten, zoals de *Pixel diff* modus:

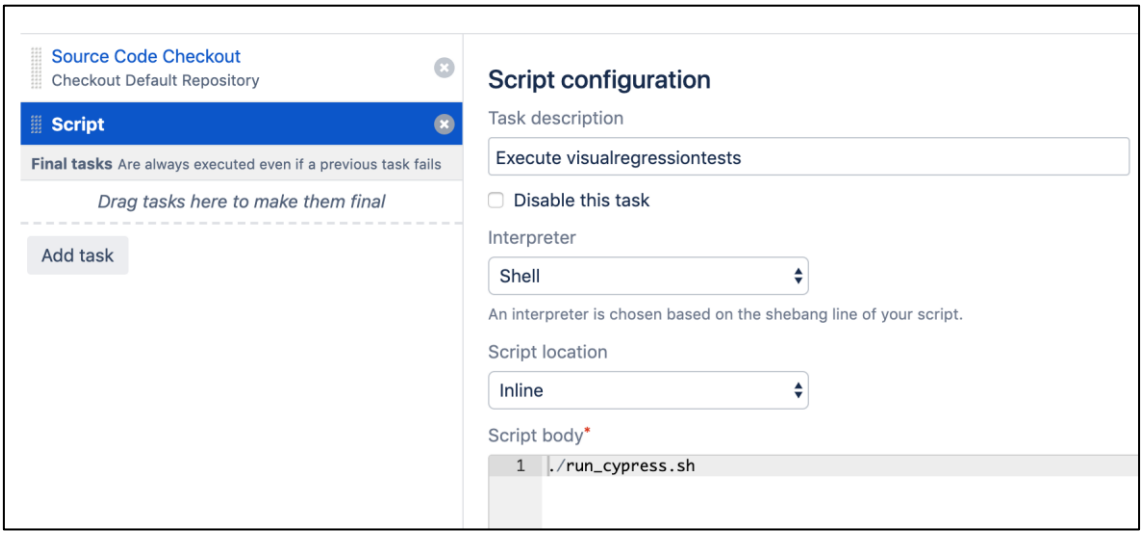


Afbeelding 10: Voorbeeld pull-request met visuele veranderingen Pixel diff

Wanneer de reviewers het eens zijn met de wijzigingen in de code en op de screenshots, kan de pull-request geaccepteerd en gemerged worden.

Bamboo

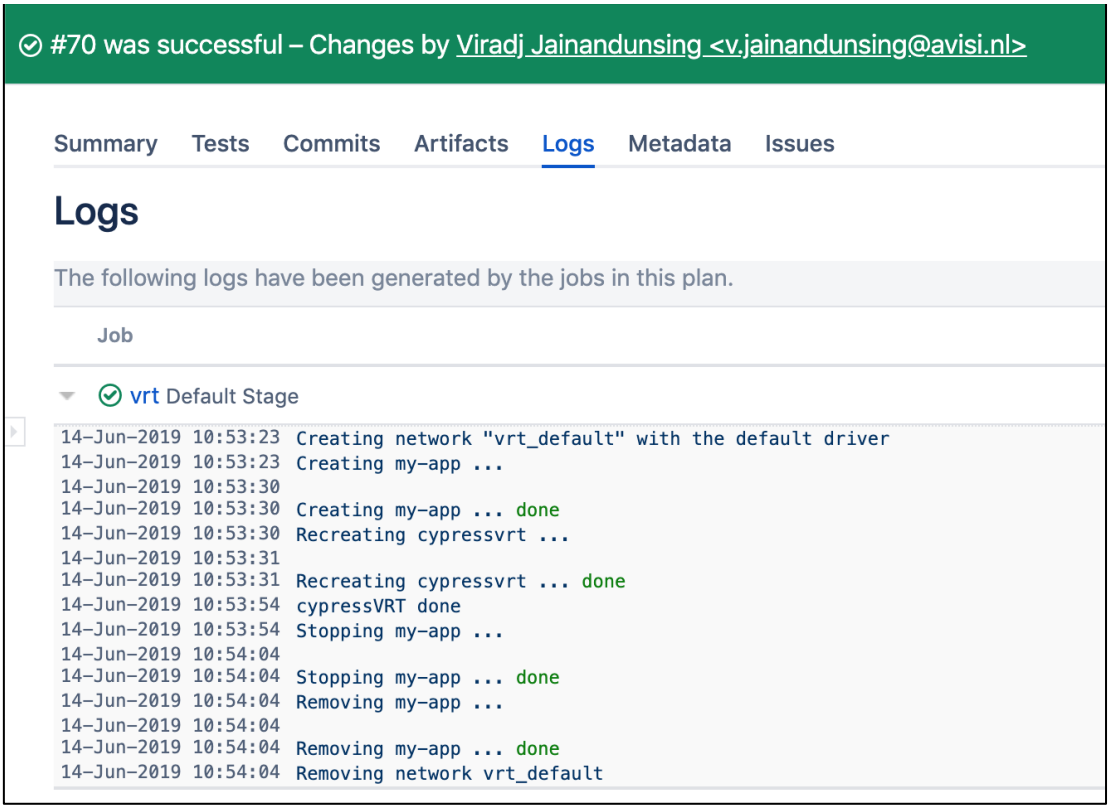
Bij het aanmaken van de pull-request start Bamboo een build waarin ook de VRT worden uitgevoerd door de *run_cypress.sh* script uit te voeren. Deze taak kan ook dusdanig worden ingesteld dat hij handmatig uitgevoerd moet worden. Hierdoor zouden de VRT niet automatisch bij iedere build gestart worden, maar pas met een klik op een knop. Dit zorgt ervoor dat er is voldaan aan eis 12: "De VRT zijn handmatig te starten in Bamboo". In het buildplan kan worden geconfigureerd welke artefacten iedere build moeten worden gepubliceerd. Dit zijn bestanden die tijdens de build worden gemaakt.



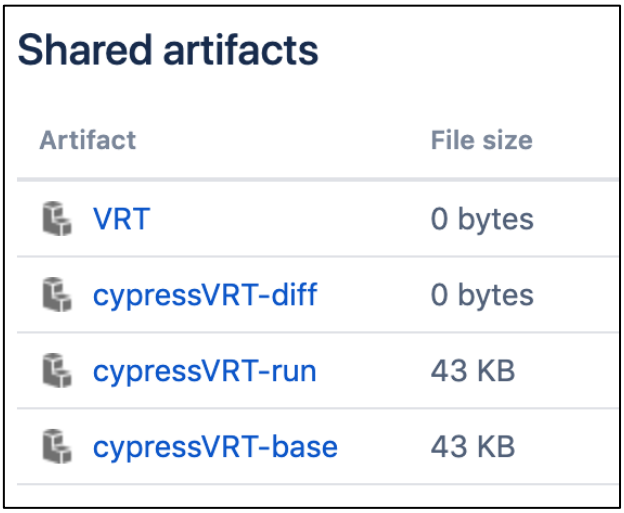
Afbeelding 11: VRT script-task op Bamboo

Geslaagde build

Zoals toegelicht in de paragraaf *Technisch ontwerp cypressVRT container* wordt er een bestand (*vrtOK*) gegenereerd wanneer er geen veranderingen zijn gevonden. Dit bestand wordt gepubliceerd als verplichte build-artefact, wat inhoudt dat de build niet slaagt bij het gebrek aan dit bestand. Hieronder staat een screenshot van de laatste regels van de logs van een geslaagde build en de bijbehorende artefacten.



Afbeelding 12: Logs geslaagde build met VRT



Afbeelding 13: Artefacten van geslaagde build met VRT

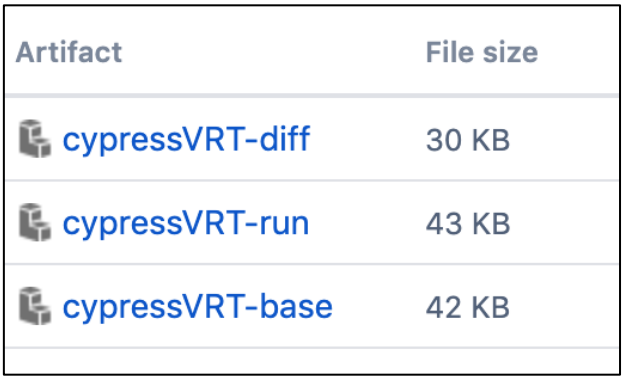
Het bovenste artefact geeft aan dat Bamboo geen visuele regressies heeft gevonden die niet door de developer is aangegeven. Daarnaast worden in drie andere artefacten de baseline-afbeeldingen, testrun-screenshots en diff-afbeeldingen gepubliceerd per build. Zoals verwacht is in dit geval de map met diff-afbeeldingen leeg.

Gefaalde build

Wanneer een developer is vergeten om (een deel van) de gewijzigde baseline-afbeeldingen te updaten, worden deze door Bamboo alsnog gevonden. Dit heeft als gevolg dat de map met diff-afbeeldingen niet leeg is en dat het *vrtOK* artefact niet wordt gemaakt. Dit resulteert in een gefaalde build, omdat dit artefact verplicht was voor de build. Dit is te zien in de laatste regels van de volgende log:

Afbeelding 14: Logs gefaalde build met VRT

Vervolgens kan met behulp van de andere artefacten (de diff-afbeeldingen in het bijzonder) achterhaald worden op welke schermen nog wijzigingen gevonden zijn die niet in de baseline zitten.



Afbeelding 15: Artefacten van gefaalde build met VRT

9. Conclusie en aanbevelingen

In dit hoofdstuk wordt teruggekoppeld naar de doelstelling van de opdracht middels het beantwoorden van de hoofdvraag van het onderzoek. Ook wordt er geëvalueerd in hoeverre de persoonlijke doelstellingen behaald zijn.

9.1 Opdracht

Deze afstudeeropdracht was bedoeld om een eerste stap te zetten richting het verbeteren van de front-end testen van Avisi. Hierbij heeft de focus gelegen op een onderzoek naar een mogelijkheid om visueleregressietesten (VRT) te integreren in het ontwikkelproces van ontwikkelteams (met name Nobilex) van Avisi. Om de hoofdvraag van het onderzoek ("Hoe kan Avisi de kwaliteit van haar webapplicaties beter waarborgen door middel van visueleregressietesten?") te beantwoorden is eerst onderzocht welke oplossingen met betrekking tot VRT reeds bestaan. Dit gaf een beeld van verschillende technieken en processen die door andere bedrijven gebruikt worden. Vervolgens is met een developer en tester van Avisi in kaart gebracht wat de wensen vanuit Avisi zijn met betrekking tot VRT. Hieruit kwamen een aantal requirements waaraan de oplossing hoe dan ook moest voldoen. Met deze *must-have* requirements kon een groot deel van de in de eerste deelvraag geïnventariseerde oplossingsmogelijkheden worden afgestreept als zijnde ongeschikt voor Avisi. Met name de requirement "De oplossing werkt niet in de cloud" was iets waar veel bestaande tools niet aan konden voldoen.

Naast de gevonden integrale oplossingen werden ook oplossingen gevonden van bedrijven die zelf een composietoplossing, bestaande uit een testrunner en een afbeeldingenvergelijker, hebben geïntegreerd in hun ontwikkelproces. Om de praktische inzetbaarheid van een composiet VRT-oplossing aan te tonen is een Proof of Concept (PoC) gebouwd die werkt in een landschap dat vergelijkbaar is met dat van het Nobilex team. Dit betekent dat er getest is tegen een Angular2 webapplicatie, waarbij Bitbucket als versiebeheersysteem (VCS) is gebruikt en Bamboo als CI/CD systeem.

Als testrunner voor de PoC is gekozen voor Cypress.io, omdat deze aan vrijwel alle requirements die gelden voor de testrunner voldoet. Als afbeeldingvergelijker is gekozen voor Pixelmatch, omdat deze als snelste uit de metingen kwam. Tijdens het testen van de PoC is er gekeken naar hoe Bitbucket omgaat met veranderingen in afbeeldingen. Hieruit bleek dat Bitbucket zelf ook verschillen in een gewijzigde afbeelding opmerkt en kan highlighten. Dit heeft tot de keuze geleid om de *baseline* in de repository van een project waarin VRT wordt toegepast te bewaren. Wanneer een scherm wordt gewijzigd door een developer, moet hij/zij ervoor zorgen dat er een geüpdate versie van een screenshot van dit scherm in de baseline terecht komt. Vervolgens kunnen andere developers/testers/UX-designers bij het reviewen van de code meteen zien welke schermen er gewijzigd zijn. Op deze manier wordt er afgedwongen dat men bewust omgaat met visuele veranderingen, maar ook dat er niet per ongeluk een onverwachte verandering wordt doorgevoerd. Ter controle voert Bamboo dezelfde VRT ook uit om te garanderen dat de developer alle gewijzigde schermen heeft geüpdatet.

Om de week is er met testers uit verschillende teams aandacht geweest voor deze opdracht. Naast de testers van Nobilex waren de testers van andere teams ook geïnteresseerd in de gebouwde oplossing. Developers van een ander team die nu bezig zijn met het bouwen van een webapplicatie noemden ook dat de VRT handig inzetbaar zouden zijn om visuele veranderingen naar de klant te communiceren.

9.2 Aanbevelingen en vervolgstappen

Omdat de composietoplossing die met de PoC gebouwd is voldoet aan alle harde eisen van Avisi, deze goed integreert in het huidige systeem en geen aanschafkosten met zich meebrengt lijkt dit een geschikte manier om VRT te introduceren bij Avisi.

Aanbevelingen

Tijdens het ontwikkelen van de PoC zijn een aantal bevindingen gedaan die worden aanbevolen om te behouden wanneer de ontwikkelde VRT-oplossing wordt gebruikt. Door de VRT eerst lokaal te draaien kunnen developers eerst zelf controleren of gemaakte wijzigingen geen onverwachte bijwerkingen hebben gehad. Door de VCS te betrekken bij het review proces kan worden gegarandeerd dat de veranderingen juist zijn geïmplementeerd. Dit kan eenvoudig worden bewerkstelligd door de baseline in de repository van een project bij te houden.

Vervolgstappen

Omdat de nadruk van het onderzoek lag bij de VRT is er gezocht naar een testrunner die voldeed aan de eisen die bij de VRT horen. Hierbij is gebruik gemaakt van Cypress.io, maar er kan een vervolgonderzoek gedaan worden naar andere testrunners die wellicht beter werken voor functionele testen. Bij dit onderzoek kan rekening worden gehouden met dezelfde eisen, om een testrunner te vinden die het best geschikt is voor beide testsoorten. De huidige manier van het updaten van de baseline gaat door afbeeldingen van de testrun map te verslepen naar de baseline. Dit kan worden verbeterd door bijvoorbeeld een UI te ontwikkelen waarin developers beide schermen naast elkaar te zien krijgen en met een muisklik de nieuwe screenshots kunnen accepteren. Ook kan er een uitbreiding gerealiseerd worden die voor de VRT een testrapport genereert waarmee Bamboo een build kan laten slagen of falen.

Tot slot leefde onder de testers de zorg dat de VRT een te grote doorlooptijd zouden hebben. De doorlooptijd hangt af van twee factoren, namelijk: de snelheid van de testrunner en de snelheid van de afbeeldingenvergelijker. Pixelmatch is met twee andere vergelijkers gemeten en doet er gemiddeld één seconde over om een vergelijking te doen. Het uitgangspunt bij deze metingen waren twee afbeeldingen van 1920 bij 1080 pixels. Cypress.io maakt echter standaard screenshots van 1000 bij 660 pixels, dus deze zullen sneller vergeleken kunnen worden. Er kan daarom worden onderzocht wat de minimale schermgrootte moet zijn voor de gemaakte screenshots. Ook is het nu zo dat Bamboo bij iedere pull-request een nieuwe build start waarbij de VRT worden uitgevoerd. Wanneer de doorlooptijd van de VRT blijken op te lopen kan ervoor worden gekozen om deze alleen in *nightly-builds* uit te voeren, zodat dit overdag niet voor extra vertraging zorgt. Het is ook verstandig om selectief te zijn in het kiezen van schermen waarop VRT moeten worden toegepast. Omdat Angular2 bijvoorbeeld met componenten werkt die je op meerdere pagina's kunt hergebruiken luidt het advies om pagina's met gedeelde componenten zeker op te nemen in de VRT. Pagina's die niet onderhevig zijn aan veranderingen, e.g. een login pagina, hoeven niet worden meegenomen in de VRT.

De PoC was zoals boven genoemd bedoeld om de praktische inzetbaarheid van VRT voor Avisi aan te tonen. Hierbij is gekozen voor Cypress.io als testrunner, Pixelmatch als afbeeldingenvergelijker, Bitbucket als VCS en Bamboo als CI/CD systeem. Deze onderdelen staan echter niet vast. Een ander team binnen Avisi noemde bijvoorbeeld dat ze binnenkort van Bitbucket willen overstappen naar Gitlab. Dit is geen probleem, omdat ook Gitlab veranderingen in afbeeldingen kan weergeven.

9.3 Persoonlijke doelen

In deze paragraaf wordt aangegeven of de persoonlijke doelen behaald zijn.

Analyseren

Er is voldoende onderzoek verricht naar de bestaande technieken en processen. Het onderzoek naar integrale oplossingen is niet uitputtend, maar de grootste spelers zijn onderzocht. Hierbij is in het begin ook contact met externe partijen geweest om mogelijkheden te onderzoeken om cloud-oplossingen *on-premise* te draaien. Er is rekening gehouden met de huidige situatie om een oplossing te vinden die daarbij past. Verder hebben de twee sessies waarin de requirements zijn opgesteld een goed beeld gegeven van de wensen van Avisi. Hiernaast heeft het consequent bijwonen van de meetings met de tester-guild een goede vorm van feedback gegeven, waarin duidelijk werd wanneer een VRT-oplossing interessant is voor testers en wanneer niet.

Ontwerpen

Het ontwerpen van de composietoplossing heeft voldoende uitdaging opgeleverd. Niet alleen was het interessant om de architectuur binnen de VRT-container uit te denken, i.e. de samenwerking tussen de verschillende onderdelen, ook moest er goed nagedacht worden over hoe de VRT in de CI/CD pipeline zou passen. Dit heeft, zoals vaker benoemd, geleid tot de conclusie dat het versiebeheersysteem een grote rol is gaan spelen bij de VRT.

Realiseren

Zoals vrij vroeg in de afstudeerperiode duidelijk werd, was er niet genoeg uitdagend programmeerwerk. Er is een PoC gerealiseerd waarbij veel kennis is opgedaan over het werken met Docker en Angular2. De technische uitdaging lag vooral in het integreren van bestaande componenten. Iets dat, ondanks de mate van complexiteit, relatief weinig code behoeft.

Adviseren

Met betrekking tot de competentie *Adviseren* staat in dit hoofdstuk het advies om VRT te introduceren in het bedrijf middels de beschreven composietoplossing. Hierbij is duidelijk aangegeven aan welke voorwaarden de verschillende onderdelen moeten voldoen om bruikbaar te zijn voor VRT.

10. Literatuurlijst

WAI-ARIA Overview. (z.d.). Geraadpleegd 3 juni 2019, van https://www.w3.org/WAI/standards-guidelines/aria/

HBO-i-domeinbeschrijving. (z.d.). Geraadpleegd 3 juni 2019, van https://hbo-i.nl/domeinbeschrijving/

Rational Unified Process - Wikipedia. (2019, 1 juni). Geraadpleegd 4 juni 2019, van https://nl.wikipedia.org/wiki/Rational_Unified_Process

ICT research methods. (z.d.). Geraadpleegd 14 februari 2019, van http://ictresearchmethods.nl/Main_Page

Docker Hub. (z.d.). Geraadpleegd 13 juni 2019, van https://hub.docker.com/u/cypress

Dockerizing an Angular App. (2019, 20 mei). Geraadpleegd 13 juni 2019, van https://mherman.org/blog/dockerizing-an-angular-app/

Visual Regression testing with Cypress.io — Xebia Blog. (2017, 9 november). Geraadpleegd 13 juni 2019, van https://xebia.com/blog/visual-regression-testing-cypress/

Regressietesten, De tien uitgangspunten. - Testersuite. (2019, 24 april). Geraadpleegd 17 juni 2019, van https://www.testersuite.nl/tien-uitgangspunten-bij-regressietesten/

Visual Regression Testing. Wat is het en wat heb je eraan? – Performance architecten. (2018, 7 november). Geraadpleegd 17 juni 2019, van https://performancearchitecten.nl/visual-regression-testing-wat-is-het-en-wat-heb-je-eraan/

Applitools Eyes. (z.d.). Geraadpleegd 17 juni 2019, van https://applitools.com/

Visual Regression Testing. (z.d.). Geraadpleegd 17 juni 2019, van https://screenster.io/

Argus Eyes. (z.d.). Geraadpleegd 17 juni 2019, van http://arguseyes.io/

JavaScript End to End Testing Framework. (z.d.). Geraadpleegd 17 juni 2019, van https://www.cypress.io/

JavaScript e2e Testing. (z.d.). Geraadpleegd 17 juni 2019, van https://www.cypress.io/pricing/

Puppeteer | Tools for Web Developers | Google Developers. (2019, 29 mei). Geraadpleegd 17 juni 2019, van https://developers.google.com/web/tools/puppeteer/

FrontPage. (z.d.). Geraadpleegd 17 juni 2019, van http://docs.fitnesse.org/FrontPage

npm: pixelmatch. (2019, 10 juni). Geraadpleegd 17 juni 2019, van https://www.npmjs.com/package/pixelmatch

ImageMagick. (z.d.). Geraadpleegd 17 juni 2019, van https://imagemagick.org/index.php

yahoo/blink-diff. (z.d.). Geraadpleegd 17 juni 2019, van https://github.com/yahoo/blink-diff

11. Bijlagen

A. Plan van aanpak

Het plan van aanpak is opgenomen als externe bijlage in het externe bestand *"Plan_van_Aanpak_Afstudeeropdracht_Viradj_Jainandunsing.pdf"*

B. Onderzoeksrapport

Het onderzoeksrapport is opgenomen als externe bijlage in het externe bestand *"Onderzoeksrapport_Afstudeeropdracht_Viradj_Jainandunsing.pdf"*

C. Reflectie

De reflectie is opgenomen als externe bijlage in het externe bestand *"Reflectieverslag_Afstudeeropdracht_Viradj_Jainandunsing.pdf"*

D. Sourcecode

De sourcecode is opgenomen als externe bijlage in het externe bestand *"PoC_Afstudeeropdracht_Viradj_Jainandunsing.zip"*