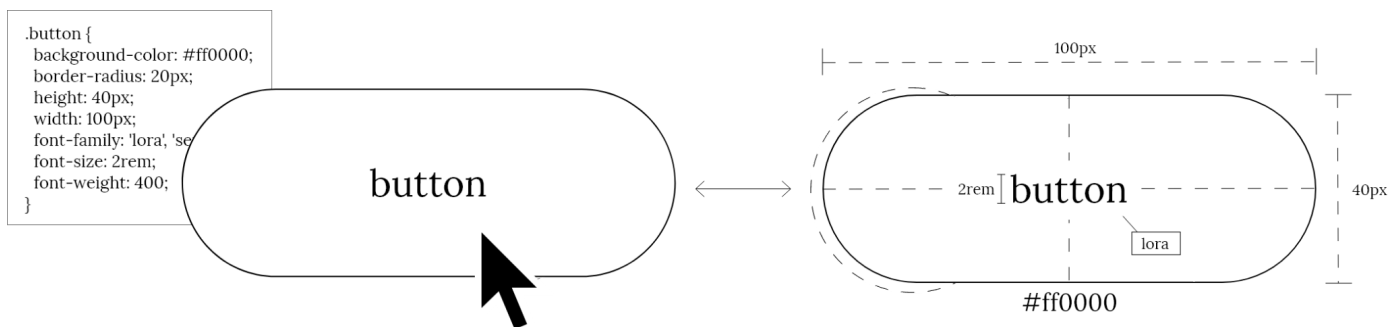


# Front-end componenten testen op basis van user interface ontwerpen

door

Jort Willemsen



1761285

Afstudeerscriptie  
HBO-ICT Software Development  
Hogeschool Utrecht

Versie: 1.0

Eerste examiner: Henk van Nimwegen  
Tweede examiner: Omar Azouguagh

Veenendaal, 29 juni 2023

# Versiehistorie

Versiehistorie		
Versienummer	Datum	Aanpassingen
0.1	31-03-23	Document opbouw gesteld, opdracht definitie toegevoegd, theoretisch kader vanuit het plan van aanpak overgenomen, grote delen deelvraag 1 en 2 beantwoord.
0.2	24-04-23	Grote delen deelvraag 1 en 2 herschreven, deelvraag 3 beantwoord, Steekwoorden conclusie opgeschreven.
0.3	08-05-23	Alle feedback verwerkt, onderzoek af.
0.4	30-05-23	Realisatie hoofdstuk toegevoegd
0.5	14-06-23	Conclusies en discussie geschreven, realisatie uitgebreid.
1.0	29-06-23	Document afgemaakt

Distributielijst			
Versienummer	Datum	Status	Aan
0.1	31-03-23	progress	Dylan Molenaar, Omar Azouguagh
0.2	24-04-23	progress	Dylan Molenaar, Nico Jansen
0.3	10-05-23	progress	Dylan Molenaar, Omar Azouguagh
0.4	30-05-23	1e concept	Dylan Molenaar, Omar Azouguagh
0.5	15-06-23	2e concept	Dylan Molenaar, Omar Azouguagh, Henk van Nimwegen
1.0	29-06-23	Definitief	Dylan Molenaar, Omar Azouguagh, Henk van Nimwegen, Kenzo Dominicus en Info Support

© Info Support B.V., Veenendaal 2023

Niets uit deze uitgave mag worden verveelvoudigd en/of openbaar gemaakt door middel van druk, fotokopie, microfilm of op welke andere wijze ook, zonder voorafgaande toestemming van Info Support BV.

No part of this publication may be reproduced in any form by print, photo print, microfilm or any other means without written permission by Info Support B.V.

Prijsopgaven en leveringen geschieden volgens de Algemene Voorwaarden van Info Support B.V. gedeponeerd bij de K.V.K. te Utrecht onder nr. 30135370. Een exemplaar zenden wij u op uw verzoek per omgaande kosteloos toe.

# Voorwoord

---

Deze afstudeerscriptie is het resultaat van alle activiteiten met betrekking tot mijn afstudeerproces. Van februari tot juli 2023 ben ik hard aan het werk geweest om een onderzoek uit te voeren en een proof of concept te realiseren. Ik wil deze gelegenheid gebruiken om een aantal mensen te bedanken die onvervangbaar waren tijdens deze periode.

Als eerst wil ik Omar Azouguagh bedanken voor het begeleiden van deze scriptie. Wanneer ik het nodig had, heeft Omar mij waardevolle feedback gegeven die mij en mijn scriptie naar een hoger niveau brachten.

Tevens wil ik mijn begeleiders vanuit Info Support, Dylan Molenaar en Kenzo Dominicus bedanken voor de ondersteuning die zij mij hebben geboden en de kennis die zij mij hebben overgebracht de afgelopen paar maanden.

Tot slot wil ik mijn diepste waardering uitspreken voor de Software Strijders. Zonder hen zou ik niet staan waar ik ben. Samen hebben we jarenlang gestreden. De vriendschappen, steun en inspiratie die ik van mijn medestrijders heb ontvangen, hebben mij gevormd tot wie ik nu ben. Dank jullie wel voor de vele discussies en projecten. Jullie hebben mijn passie voor het vak versterkt.

Ik wens u veel leesplezier toe.

Jort Willemsen

Veenendaal, 30 mei 2023

# Abstract

---

Info Support ontwikkelt voor een kredietverstrekker in Nederland, een klantportaal en een interne webapplicatie. Info Support ziet een lage testdekking voor de portalen die zij ontwikkelen voor de kredietverstrekker. Hierdoor ontstaan veel visuele bugs die vaak niet automatisch te vinden zijn.

Info Support wil minder visuele bugs en inconsistenties in de productieomgeving van de portalen vinden en tijd besparen tijdens het testen van de applicatie door geautomatiseerde tests te kunnen schrijven. Dit betekent dat er meer tijd besteed kan worden aan het toevoegen van functionaliteit en het innoveren van de software voor de kredietverstrekker.

Hiervoor is de volgende onderzoeksvraag opgesteld:

**“ Hoe kan een user interface test ontwikkeld worden voor een front-end component die eigenschappen van een user-interface design vergelijkt met de geïmplementeerde code, in een nieuwe duurzame javascript omgeving, zodat de samenhang tussen het user-interface design en het component gewaarborgd kan worden? “**

Om de test te ontwikkelen, zijn verschillende factoren geïdentificeerd die de softwarekwaliteit kunnen beoordelen, zoals correctheid, integriteit, testbaarheid en structuur. Door het vergelijken van de eigenschappen van het design met die van de geïmplementeerde code, kunnen deze factoren worden getest. Om dit te realiseren, moet er een speciale test runner worden ontwikkeld, aangezien bestaande test runners voor Javascript deze functionaliteit niet bieden.

Voor het renderen van het component wordt gebruikgemaakt van Storybook, gezien het ondersteuning biedt voor de meeste frameworks. Een webdriver wordt gebruikt om de HTML-pagina via code te inspecteren en eigenschappen van het component te extraheren op basis van CSS-selectors. Het design platform Figma wordt gebruikt omdat het goede integratiemogelijkheden biedt via een API. Met behulp van de node-structuur die Figma levert, kunnen de eigenschappen van het design worden opgehaald en vergeleken met de eigenschappen van het component om een test te genereren.

Standaard testing frameworks zijn niet geschikt voor deze vergelijkingstaak, dus er moet een nieuw testing framework worden ontwikkeld om een testspecificatie te genereren, de test uit te voeren en het testrapport te genereren.

Het resultaat van de testrun is een testrapport dat inzicht geeft in de kwaliteitsfactoren van het component, zoals het aantal gegenereerde tests, het aantal geslaagde of gefaalde tests en de complexiteit van de componenten. Dit biedt een indicatie voor de softwarekwaliteit van het front-end component.

Dit onderzoek biedt een aanpak voor het ontwikkelen van een user interface test die eigenschappen van een user-interface design vergelijkt met de geïmplementeerde code. Het maakt gebruik van specifieke tools en frameworks, zoals React, Storybook, Figma en een op maat gemaakt testing framework, om de gewenste functionaliteit te realiseren en inzicht te bieden in de kwaliteit van de software.

# Inhoudsopgave

---

<b>Versiehistorie</b>	<b>1</b>
<b>Voorwoord</b>	<b>2</b>
<b>Abstract</b>	<b>3</b>
<b>Inhoudsopgave</b>	<b>4</b>
<b>1. Inleiding</b>	<b>5</b>
<b>2. Opdrachtdefinitie</b>	<b>6</b>
2.1. Kwestie	7
2.2. Doelstelling	7
2.3. Onderzoeksvragen	8
<b>3. Theoretisch kader</b>	<b>9</b>
<b>4. Onderzoek</b>	<b>13</b>
4.1. Beantwoording deelvraag 1	13
4.2. Beantwoording deelvraag 2	21
4.3. Beantwoording deelvraag 3	30
4.4. Onderzoeksvraag	34
<b>5. Realisatie</b>	<b>35</b>
5.1. Architectuur	35
5.2. Way of working	39
5.3. Softwarekwaliteit	41
5.4. Functionaliteit	44
5.5. Advies	48
<b>6. Conclusie &amp; aanbevelingen</b>	<b>49</b>
6.1. Kwaliteit	50
6.1.1. Leesbare code	50
6.1.2. Testen	50
6.1.3. Uitwisselbaarheid	50
6.1.4. Logging	50
<b>7. Discussie</b>	<b>51</b>
7.1. Validiteit	51
7.2. Procesgang	53
7.3. Aanbevelingen	54
<b>8. Literatuur</b>	<b>55</b>

# 1. Inleiding

---

Als afstudeerstage voor de HBO opleiding software ontwikkeling van de Hogeschool Utrecht, heeft Info Support gevraagd de student onderzoek te doen naar het ontwikkelen van user-interface testen om visuele bugs te verminderen in de web-portalen die Info Support bouwt voor een kredietverstrekker in Nederland. In dit document worden de resultaten van het onderzoek beschreven en hoe de implementatie van het onderzoek is verlopen.

Hoofdstuk 2 beschrijft de opdrachtdefinitie. Hier worden de specifieke doelstellingen en uitdagingen van het project in detail besproken. In hoofdstuk 3, het theoretisch kader, worden relevante concepten en methoden geïntroduceerd die van toepassing zijn op het onderzoek. Hoofdstuk 4, is gewijd aan het onderzoek zelf. Hier wordt het onderzoek en de resultaten gepresenteerd. Hier zijn de beantwoordingen van de deelvragen en de hoofdvraag te vinden.

Hoofdstuk 5 behandelt de realisatie van de oplossing. Het gaat in op de architectuur en de werkwijze van de ontwikkelde applicatie. Na de realisatie volgen de conclusies in hoofdstuk 6. Hier worden de belangrijkste bevindingen en resultaten samengevat. Dit hoofdstuk geeft een helder overzicht van de behaalde doelstellingen. Ten slotte wordt in hoofdstuk 7 de discussie gevoerd. Er wordt gereflecteerd op de uitgevoerde werkzaamheden en resultaten, en beperkingen en suggesties voor de toekomst worden besproken.

## 2. Opdrachtdefinitie

---

De student werkt aan een oplossing voor Info Support (opdrachtgever), een groot ICT consultancy bedrijf uit Veenendaal. Info Support staat bekend als een innovator op gebied van softwareontwikkeling en consultancy.

Info Support maakt voor een kredietverstrekker in Nederland een klantportaal om klanten inzicht te geven in alle zaken die te maken hebben met zijn / haar krediet. Daarnaast bouwen zij een interne applicatie waar kredietacceptatie en contract intakes worden uitgevoerd door medewerkers van de kredietverstrekker.

De student ontwikkelt een oplossing om visuele bugs in de portalen op te lossen door onderzoek te doen naar hoe front-end component testen geïntroduceerd kunnen worden in de portalen zodat er fouten uit de code gehaald kan worden zonder deze handmatig te laten testen door ontwikkelaars of quality assurance testers. De student heeft een specifieke focus op het testen of de front-end componenten voldoen aan user-interface (UI) eigenschappen van een vooraf gedefinieerd UI design. Er moet ook worden onderzocht of het mogelijk is om deze front-end component test automatisch te laten genereren op basis van de eigenschappen uit dit UI design.

Een uitdaging is dat de portalen geschreven zijn met verouderde technieken die niet meer ondersteund worden door ontwikkelaars en daardoor niet meer geüpdatet worden voor nieuwe functionaliteit of beveiligingsopties (*Knockout: Home, n.d.*). Er is een grote kans dat deze verouderde technieken de nieuwe functionaliteit niet ondersteunen. Daarom heeft Info Support samen met de student besloten dat er tijd besteed wordt in het onderzoek om een nieuwe technologie te vinden die (i) past bij Info Support, (ii) toekomstbestendig en onderhoudbaar is en (iii) de gewenste functionaliteit voor de afstudeeropdracht van de student ondersteund.

Na de uitvoering van het onderzoek, bouwt de student een proof of concept om de bevindingen die in het onderzoek naar voren zijn gekomen te testen in een realistische omgeving.

De opdracht van Info Support voor de student luidt:

**“ Onderzoek hoe een user interface test ontwikkeld kan worden voor een front-end component die eigenschappen van een user-interface design vergelijkt met de geïmplementeerde code, in een nieuwe duurzame javascript omgeving, zodat de samenhang tussen het user-interface design en het component gewaarborgd kan worden “.**

## 2.1. Kwestie

Info Support ziet een lage testdekking voor de portalen die zij ontwikkelen voor een kredietverstrekker. Hierdoor ontstaan veel visuele bugs die vaak niet automatisch te vinden zijn. Dit betekent dat bij iedere verandering aan de code iemand van Info Support de complete applicatie handmatig moet testen om te kijken of deze applicatie nog werkt naar behoren. Om dit zorgvuldig te doen kost het heel veel tijd die Info Support en de klant liever besteden aan het verbeteren van hun portalen of nieuwe functionaliteit introduceren die businesswaarde biedt voor de klant.

Om deze visuele bugs te verminderen wil Info Support graag gaan werken met user-interface designs. Deze kunnen helpen een ontwikkelaar een beter beeld te geven tijdens het ontwikkelen van de front-end componenten. Door deze designs naast het component te houden kan er dus gekeken worden of deze overeenkomen en wordt de kans op bugs en visueel inconsistente componenten kleiner.

Ook geeft Info Support aan dat de portalen op dit moment met verouderde technologieën ontwikkeld zijn die geen toekomstbestendigheid bieden. De technologie: Knockout.js, waarmee de front-end componenten op dit moment gebouwd zijn, wordt niet meer ondersteund door de ontwikkelaars en mist daardoor belangrijke updates en quality of life verbeteringen. Hierdoor is het programmeren van nieuwe functionaliteit heel uitdagend of in sommige gevallen zelfs onmogelijk.

## 2.2. Doelstelling

Info Support wil minder visuele bugs en inconsistenties in de productieomgeving van de portalen van de klant vinden en tijd besparen tijdens het testen van de applicatie door geautomatiseerde tests te kunnen schrijven. Dit betekent dat er meer tijd besteed kan worden aan het toevoegen van functionaliteit en het innoveren van de software voor de klant.

Om dit te doen wilt Info Support ook graag een modernisering in de gebruikte technologieën voor het ontwikkelen van de portalen zodat nieuwe functionaliteit makkelijker geïmplementeerd kan worden en de software toekomstbestendiger wordt doordat deze nieuwe technologie ondersteund en geüpdatet wordt.

Het hoofddoel voor de student is om af te studeren. Dit houdt in dat de student moet voldoen aan de afstudeereisen gesteld door de Hogeschool Utrecht. Als de student voldoet aan deze eisen is de student afgestudeerd en heeft hij zijn opleiding voltooid.



## 2.3. Onderzoeksvragen

De te testen user interface componenten van de portalen van de kredietverstrekker zijn gemaakt in een javascript framework: knockout.js (Knockout: Home, n.d.). Deze user interface componenten worden geïnjecteerd in een ASP.NET web applicatie (ASP.NET | Open-source Web Framework For .NET, n.d.). Een uitgangspunt van de opdrachtgever is dat het huidige javascript framework niet geschikt is voor geautomatiseerde user interface testen. De student gaat onderzoek doen naar een nieuw duurzaam framework waarin geautomatiseerde user interface testen ontwikkeld kunnen worden en bij voorkeur naast het verouderde Knockout.js framework gebruikt kan worden zodat langzaam naar het duurzame framework gemigreerd kan worden.

De student gaat onderzoeken welke eigenschappen een user interface design bevat en welke belangrijk zijn om te testen. Hierna gaat de student een manier zoeken om deze eigenschappen te testen in het duurzame javascript framework. Waarna er onderzocht wordt op welke manier en in welke mate deze tests automatisch de eigenschappen kunnen testen die in het user interface design van het component beschreven zijn.

De hoofdvraag van het onderzoek is dus:

**“ Hoe kan een user interface test ontwikkeld worden voor een front-end component die eigenschappen van een user-interface design vergelijkt met de geïmplementeerde code, in een nieuwe duurzame javascript omgeving, zodat de samenhang tussen het user-interface design en het component gewaarborgd kan worden? “**

### 2.3.1. Deelvragen

De hoofdvraag valt onder te delen in verschillende deelvragen die helpen om de hoofdvraag zo duidelijk mogelijk te beantwoorden. Hieronder staan de verschillende deelvragen beschreven:

1. Welk duurzaam javascript framework dat geschikt is voor geautomatiseerd user interface testen, kan gebruikt worden ter vervanging van het huidige Knockout.js framework?
2. Hoe kan binnen het duurzame javascript framework testen geschreven worden voor een front-end component die user interface design eigenschappen valideren?
3. In welke mate kunnen de component testen automatisch de eigenschappen van het user experience design gebruiken voor de specificatie van de test?

Iedere deelvraag wordt los beantwoord in het onderzoek en bevat een eigen conclusie. In §4.5. wordt op basis van de conclusies van de deelvragen de hoofdvraag beantwoord. De conclusie van het onderzoek wordt beschreven in hoofdstuk 5.

Deelvraag	Onderzoek methoden
1	Literatuuronderzoek, interviews, available product analysis
2	Literatuuronderzoek, available product analysis
3	Literatuuronderzoek, available product analysis

## 3. Theoretisch kader

---

### 3.1. Javascript framework

Li et al. (2020) Ziet dat sinds de introductie van micro-service architectuur, het frontend landschap aan het veranderen is. Oude MVC (Model, View, Controller) architectuur wordt ingewisseld voor een gescheiden landschap waar backend en frontend van elkaar losgetrokken worden.

MVC biedt een scheiding tussen business logic en de grafische gebruikersinterface voor de gebruiker maar houdt dit binnen dezelfde technologie. Sinds de opkomst van verschillende “op component gebaseerde” architecturen zijn frameworks als React en Angular belangrijker geworden. Zij bieden namelijk manieren om componenten te customizen en te encapsuleren (Li et al., 2020).

In de context van dit project wordt een javascript framework gedefinieerd als een “op componenten gebaseerd” framework dat is geschreven in Javascript.

## 3.2. User interface design

International Standards Organisation [ISO]. (2019) beschrijft user experience als:

*“Person’s perceptions and responses resulting from the use and/or anticipated use of a product, system or service”*

- International Standards Organisation [ISO]. (2019)

ISO geeft aan dat user experience probeert systemen bruikbaar te maken door de focus te leggen op de gebruiker, hun behoeften en eisen. Dit verhoogt de gebruikerstevredenheid, toegankelijkheid en duurzaamheid.

Department of Health and Human Services, (z.d.) Ziet belangrijke doelen van het gebruik van user interface designs:

- Het strategisch gebruik maken van kleur, en textuur zodat focus voor de gebruiker geleid kan worden naar belangrijke delen of juist weggeleid kan worden van andere delen van de software.
- Het correct gebruik van typografie om juiste hiërarchie en helderheid in de software aan te brengen
- Consistentie creëren door het hergebruiken van elementen zodat de gebruiker zich comfortabeler voelt en sneller de software kan gebruiken zoals de gebruiker dat wil.

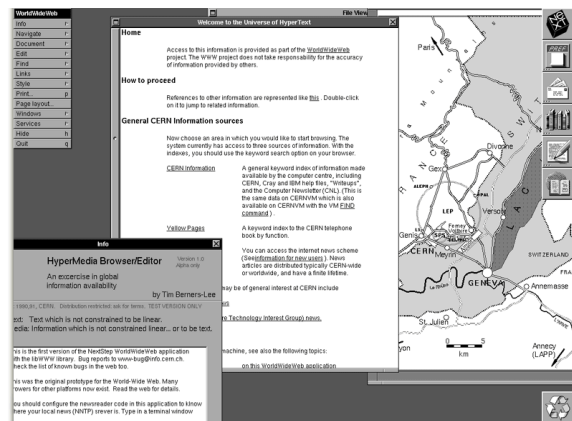
Er zijn verschillende eigenschappen van een design die belangrijk zijn voor de interactie van de gebruiker met de software. Figma. (z.d.) geeft aan dat onder andere attributen zoals, kleur, afmetingen en verschillende teksteigenschappen een aandeel hebben in de gebruiksvriendelijkheid van het eindresultaat.

De student wil de bovengenoemde doelen makkelijker maken om te testen op geïmplementeerde user interface designs in front-end componenten van Info Support. Door de eigenschappen van een user interface design te gebruiken kan een strategie opgezet worden om de eigenschappen die deze doelen realiseren, geautomatiseerd te testen.

### 3.3. Front-end web software

Het wereldwijde web (WWW) is uitgevonden in 1989 door Tim Berners-Lee, werkend bij CERN om wetenschappers over de hele wereld met elkaar te verbinden (A short history of the Web, 2023). Het WWW model liet snelle en makkelijke toegang tot belangrijke informatie toe aan CERN wetenschappers.

Om de wereld toegang te geven tot het WWW heeft Berners-Lee de eerste webbrowser ontwikkeld waarin visueel geïnteracteed kon worden met informatie op het internet door middel van Hyper Text Markup Language (HTML). Door de jaren heen zijn zowel de webbrowser als de webpagina's door ontwikkeld tot een onmiskenbaar deel van het leven van 66% van de wereldbevolking, ofwel 5.3 miljard mensen. (Facts and Figures 2022, z.d.).



Figuur 1 - Tim Berners-Lee's original WorldWideWeb browser (cern.info.ch, z.d.)

Front-end web software is een grafische user-interface van een website door gebruik te maken van HTML waardoor gebruikers kunnen interacteren met de website en bouwt voort op de principes en technologieën bedacht door Berners-Lee in de vroege jaren 90.

#### 3.3.1. Moderne front-ends

Tegenwoordig is het wereldwijde web verdeeld en moet een website kunnen draaien op veel verschillende browsers en apparaten. Hiervoor zijn moderne technologieën bedacht zoals de front-end frameworks die het makkelijker maken websites en webapplicaties te ontwikkelen. Pekarsky (2020) ziet dat frameworks de volgende oplossingen kunnen bieden:

- Beter onderhoudbare code
- Makkelijke UI manipulatie op basis van data
- Hergebruik van code door implementatie componenten
- Gebruik van boilerplate code om veelvoorkomende uitdagingen op te lossen.

De jaarlijkse enquête van Stackoverflow (2022), die door meer dan 70.000 ontwikkelaars wordt ingevuld geeft aan dat bij de vraag:

*" Which web frameworks and web technologies have you done extensive development work in over the past year, and which do you want to work in over the next year? (If you both worked with the framework and want to continue to do so, please check both boxes in that row.) "*

60% van de ontwikkelaars React.js gebruikt, 20% Angular.js en 19% Vue.js en daardoor onmisbaar zijn geworden voor moderne front-end web ontwikkeling.

De student doet onderzoek hoe testen geschreven kunnen worden voor webapplicaties die werken met moderne technologieën zoals React.js, Angular.js en Vue.js.

## 3.4. Front-end component

Moderne front-ends zijn vaak gebaseerd op een op componenten gebaseerde architectuur. Dit houdt in dat de software opgebouwd is uit verschillende geïsoleerde stukken software (componenten) die los van elkaar kunnen functioneren. Voordelen (Frontend Component-based architecture, z.d.) hiervan zijn:

- **Geen grote scope**  
Reduceert complexiteit met een kleine component scope. Hierdoor hoeft de ontwikkelaar niet het gehele project in de gaten te houden.
- **Geen afhankelijkheden naar andere componenten**  
Componenten kunnen veranderd of vervangen worden zonder impact te hebben op de functionering van het hele systeem.
- **Testen**  
Het ontwikkelen van testen is makkelijker omdat het component geïsoleerd is tegenover andere componenten waardoor deze getest kan worden zonder rekening te hoeven houden met de rest van het systeem.

### 3.4.1. Front-end component- & user-interface testen

Het is belangrijk dat componenten getest worden. Dit kan volgens Boog (2023) voordelen bieden zoals de kwaliteit van de code verifiëren, het versimpelen van het refactoren van code, vereenvoudigen van debuggen en het risico verminderen voor het veranderen van oudere source code.

Front-end componenten worden gebruikt om user-interfaces te ontwikkelen. Hierdoor testen we ook de user-interface van de applicatie. Fay (2023) van Poll the People, een marktonderzoek organisatie zegt het volgende over waarom user interface testing belangrijk is:

*"User interface testing is important because it helps ensure that your app or website is easy to use and navigate for your users. By testing the UI, you can catch any potential issues that might make using your app or website difficult or confusing for your users."*

- (Fay, 2022)

Er zijn verschillende tools beschikbaar om front-end componenten te testen. De student gaat uitzoeken welke tools hierbij kunnen helpen en hoe component testen gerealiseerd kunnen worden binnen een duurzaam Javascript framework voor Info Support.

## 4. Onderzoek

---

### 4.1. Beantwoording deelvraag 1

*“Welke duurzame javascript frameworks die geschikt zijn voor geautomatiseerd user interface testen, kunnen gebruikt worden ter vervanging van het huidige Knockout.js framework?”*

De opdracht vraagt naar een oplossing om front-end componenten te testen op basis van een user-interface design. Hiervoor moet onderzocht worden welke front-end javascript framework of frameworks ondersteund moeten worden door deze oplossing. Gezien de opdracht uitgevoerd wordt op vraag van een opdrachtgever is er gekozen om interviews af te nemen met de opdrachtgever en een aantal ontwikkelaars bij Info Support die werken met front-end software om er achter te komen welke frameworks zij gebruiken en of belangrijk vinden om te ondersteunen.

#### 4.1.1. Knockout.js

Momenteel wordt gewerkt met Knockout.js. De ontwikkelaars van Info Support geven aan dat het een verouderd javascript framework dat niet meer in onderhoud is waardoor het gebruik hiervan snel achteruit loopt en belangrijke (security) updates niet meer ontwikkeld worden. Ook geven ze aan dat zij willen migreren naar een nieuw framework dat ondersteund wordt door de oplossing die de student gaat realiseren. De opdrachtgever geeft aan dat het nieuwe framework duurzaam moet zijn. Duurzaamheid is een begrip met een losse betekenis. Om dit duidelijk te krijgen is er gevraagd aan de ontwikkelaars wat zij vinden dat duurzaamheid betekent in de context van een front-end framework die gebruikt kan worden als vervanging voor het huidige Knockout.js framework.

#### 4.1.2. Duurzaamheid

Info Support beschrijft duurzaamheid als een toekomstbestendig framework. Dit houdt in dat er een, bij voorkeur open source, community van ontwikkelaars en onderhouders achter zit die actief bezig zijn met het framework om deze zo goed mogelijk up-to-date te houden met de nieuwste technieken en (security) updates. Bij voorkeur wordt het framework onderhouden of gesteund door een groot techbedrijf. Nick de Beer, een ontwikkelaar bij Info Support die werkt aan de portalen geeft aan dat grote tech bedrijven actief bezig zijn met het ontwikkelen van front-end frameworks en dat er enorm veel geld aan gespendeerd wordt waardoor de kwaliteit hiervan meestal hoger ligt en sneller (security) updates geïntroduceerd kunnen worden wanneer dit nodig is.

Volgens de ontwikkelaars van Info Support is het gewenst dat een framework zo lang mogelijk ondersteund wordt. Hier komt het open-source gedeelte van het framework aan bod: Als een bedrijf zich terugtrekt uit een framework kan de community ervoor kiezen deze nog steeds te blijven ondersteunen voor feature en security updates en blijft de source code beschikbaar om over te nemen of te gebruiken om zelf mee door te werken.

Info Support en de student definiëren duurzaamheid als: Toekomstbestendig (support & populariteit), bruikbaar en testbaar. Op basis van deze factoren wordt een framework uitgekozen. Voor iedere factor krijgt een framework een score van 1 tot 5. Waarbij 1 een lage score is en 5 een goede score is.

### 4.1.3. Aansluiting bij Info Support

Het is belangrijk dat de oplossing die uiteindelijk gekozen wordt om het doel van de opdracht te bereiken zo goed mogelijk bij Info Support past. Dit betekent dat het bruikbaar moet zijn binnen de portalen die zij bouwt voor de kredietverstrekker maar het liefst ook voor andere klanten die Info Support van front-end applicaties verzorgd. Om dit uitgangspunt in acht te nemen is hier expliciet bij stil gestaan tijdens de interviews die de student heeft gehouden met ontwikkelaars van Info Support. Een samenvatting van deze interviews is te vinden in Bijlage 1.1.

De ontwikkelaars hebben aangegeven welke frameworks zij ervaring mee hebben, welke op dit moment gebruikt worden binnen projecten en welke zij denken dat in de toekomst gebruikt gaan worden. Hier is een lijst van drie frameworks uitgekomen: (I) Angular, (II) React.js, (III) Vue.js.

#### Angular

Angular wordt vooral gebruikt binnen de projecten voor de kredietverstrekker en wordt daarom benoemd door alle geïnterviewden. Het framework wordt onderhouden door Google en wordt gebruikt door miljoenen ontwikkelaars over de gehele wereld (Angular. z.d.). Ook geeft Angular aan op hun website dat het een platform is dat gebouwd wordt voor de toekomst en de intentie heeft om nog vele jaren door te gaan met het ontwikkelen van het framework.

#### React.js

Een andere technologie die door iedere ontwikkelaar benoemd werd is: React.js. Over de afgelopen 10 jaar, sinds release op 30 mei 2013, is React.js doorgroeid tot de meest gebruikte en populairste web technologie. React.js wordt ontwikkeld en onderhouden door Meta (voorheen Facebook).

Een groot voordeel van React.js is dat het geen framework is maar een library. Dit houdt in dat het framework abstract is en daardoor op veel verschillende manieren en voor veel doeleinden gebruikt kan worden. Denk hierbij aan het React Native framework dat de React.js library gebruikt om user-interfaces te realiseren voor mobiele applicaties.

#### Vue.js

De laatste, en kleinste van de zogenoemde “grote drie” web frameworks is Vue.js. Een zelf genoemd progressief framework houdt in dat het makkelijker en aangenamer is om te ontwikkelen in Vue. Het is bedacht en wordt onderhouden door Evan You en een core team van open source ontwikkelaars (*Meet the Team | Vue.js, z.d.*). Het wordt gesponsord door veel grote bedrijven en organisaties zoals: the open source collective, story block en Sentry (*Vue.js - the Progressive JavaScript Framework | Vue.js, z.d.*).

#### Overig

Om zeker te weten of de drie bovengenoemde frameworks voldoen, is er gekeken of er geen frameworks zijn waar Info Support geen of weinig ervaring mee heeft, maar wel aansluiting bieden bij de wensen van Info Support. Hiervoor wordt gekeken naar de jaarlijkse

#### 4.1.4. Testbaarheid

Om te weten op welke frameworks de oplossing van de student zich moet richten, is er gekeken welke frameworks gebouwd zijn, of makkelijk werken met user-interface testen. In deelvraag 2 wordt onderzocht hoe een user-interface test ontwikkeld kan worden. Het is belangrijk dat die manier ondersteund wordt door het javascript framework dat gekozen wordt voor Info Support er haar klant.

In deelvraag 2 is onderzocht of Storybook gebruikt moet worden om de componenten mee te renderen. Dit bleek de beste optie te zijn. Het is daarom van belang dat Storybook ondersteuning biedt voor het javascript framework.

	React	Angular	Vue	Solid	Svelte
<b>Storybook</b>	yes	yes	yes	~	~

Tabel 1: Framework ondersteuning voor React  
(Feature Support for Frameworks, z.d.)

In Tabel 1 is te zien dat Storybook op dit moment goede ondersteuning biedt voor React, Angular en Vue. Voor deze frameworks heeft Storybook teams die de integratie hiermee onderhouden.

Voor Solid en Svelte is er geen “first-party” ondersteuning, maar wordt volgens Storybook wel door de community aan gewerkt door middel van open-source contributies. Storybook kan een blijvende integratie voor deze platforms niet garanderen, maar de meeste features van Storybook worden op dit moment wel ondersteund (Feature Support for Frameworks, z.d.).

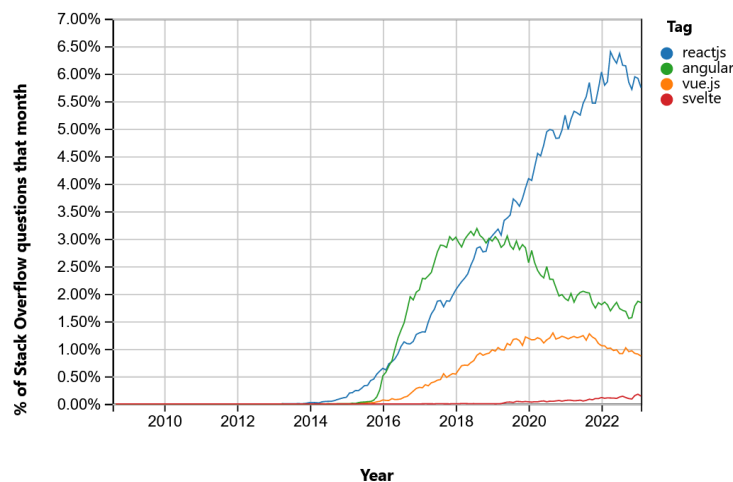
	React	Vue	Angular	Solid	Svelte
<b>Score</b>	5	5	5	3	3



### 4.1.5. Populariteit & Support

Info Support geeft aan bij het definiëren van het begrip duurzaamheid dat toekomstbestendigheid, populariteit en de community die actief bezig is met het onderhouden en verbeteren van het framework een grote rol spelen.

Stack Overflow is een platform waarop vragen gesteld kunnen worden door developers, aan andere developers. Stack Overflow geeft zelf aan dat 81% van alle ontwikkelaars minimaal 1 keer per week de website bezoeken om een antwoord op een vraag te bekijken of er zelf één te stellen. Door het aantal gestelde vragen op het platform af te zetten tegen hoe vaak de tag van een van deze frameworks is gebruikt kan er een diagram geplotted worden die inzicht kan geven in de populariteit en gebruik van een framework onder ontwikkelaars.



Figuur 1: Aantal Stack Overflow vragen m.b.t. frameworks (Stack Overflow, z.d.)

Het aantal vragen in % die de tags: "reactjs", "angular", "vue.js" en "svelte" bevatten op Stack Overflow per maand.

Om een getal te binden aan hoe goed de support is voor een framework wordt er gekeken naar hoe vaak de package van deze frameworks geüpdatet wordt. Als een package frequenter geüpdatet wordt, betekent dit dat de support hoger is.

Framework (npm package naam)	NPM downloads (x per week)	NPM update frequentie (x per week)
react	17.572.850x	14x
vue	3.470.000x	2x
@angular/core	2.923.000x	7x
solid-js	88.000x	1,9x
Svelte	470.000x	0,5x

Tabel 2: Aantal updates per week per framework op npm (node package manager [npm], n.d)

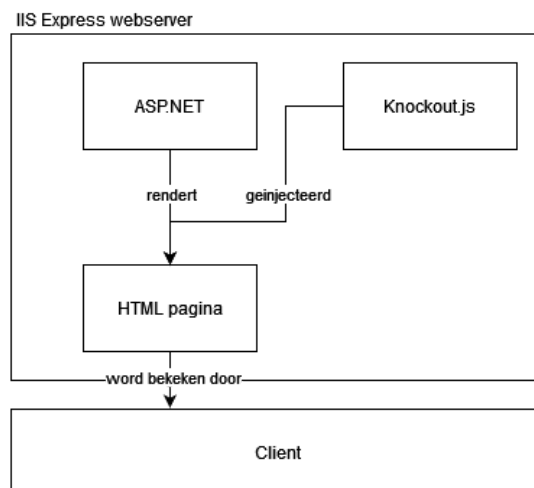
Support	React	Vue	Angular	Solid	Svelte
Score	5	3	4	2	1

Populariteit	React	Vue	Angular	Solid	Svelte
Score	5	3	4	1	2

#### 4.1.6. Bruikbaarheid

Om te onderzoeken of frameworks te gebruiken zijn als vervanging voor het huidige Knockout.js framework, zijn er prototypes gebouwd die dat kunnen verifiëren.

De huidige implementatie bestaat uit een ASP.NET server side rendered applicatie waar Knockout.js componenten geïnjecteerd worden. Deze wordt geserveerd via IIS Express, een webserver host. Dat ziet er als volgt uit:



Figuur 2: Huidige front-end architectuur bij Info Support

De ontwikkelde prototypes imiteren de APS.NET en IIS express omgeving door deze op een simpele manier na te bouwen waarna vervolgens gekeken wordt of hierin een nieuw framework geïmplementeerd kan worden. Er zijn prototypes gebouwd voor React, Angular en Vue.

## React

```
<div id="content">  
  @Html.React("Components.Button", new  
  {  
    text = "Hi React in ASP.NET!"  
  }, renderFunctions: new ChainedRenderFunctions(reactJssFunctions))  
</div>
```

Figuur 3: React component in CSHTML

Door gebruik te maken van de Nuget package Reactjs.NET (React Integration for ASP.NET MVC | ReactJS.NET, z.d.) kan op een intuïtieve manier React code geïnjecteerd worden in CSHTML. In Figuur 3 wordt een Button component in de inhoud van de pagina geplaatst.

## Vue

Rios (2021) heeft een stappenplan om Vue in een ASP.NET MVC applicatie te gebruiken. Het proces is een stuk complexer omdat er geen specifieke componenten geschreven worden in javascript. Er wordt direct in de CSHTML geschreven waardoor het niet mogelijk is voor Storybook om het component te renderen.

## Angular

```
@section scripts{  
  <script src="~/Scripts/angular.js">script<  
}</script>
```

Next, apply the ng-app directive and any other required directives on the HTML element as shown below:

```
<div ng-app="" class="row">  
  <input type="text" ng-model="name" />  
  {{name}}  
</div>
```

Figuur 4: Voorbeeld van Angular in CSHTML

Ferreira (2018) geeft een manier om Angular te injecteren in ASP.NET. Het kost veel tijd om te implementeren gezien de setup complex is. De bundeling van de Angular Javascript code in de ASP.NET omgeving is niet makkelijk en vereist een diepe kennis van beide talen en technieken.

## Solid & Svelte

Voor Solid en Svelte is er gekeken of het theoretisch mogelijk is om deze te ondersteunen binnen de ASP.NET applicatie. Er is op Google en Nuget gezocht op de termen:

- SolidJS in ASP.NET MVC,
- Solidjs for .NET,
- Use SolidJS in ASP.NET

Google en Nuget gaven “Fable:Core” aan als een tool die misschien kan helpen. Fable is een manier om Javascript en F# tegelijk te gebruiken door F# te compileren naar Javascript (Fable.Core 4.0.0, z.d.). Het biedt ondersteuning voor SolidJS door gebruik te maken van de Fable:Solid package Fable-Compiler (z.d.). Dit is alleen geen oplossing om SolidJS te gebruiken in ASP.NET. Er zijn op google geen andere manieren of hulpmiddelen gevonden.

```
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}

<svelte-app id="khalid" name="Khalid"></svelte-app>
<svelte-app id="paul" name="Paul"></svelte-app>
```

Figuur 5: Svelte in CSHTML volgens Abuhakmeh (2021)

Voor Svelte is er op dezelfde manier gezocht naar tutorials, tools of andere hulpmiddelen die kunnen helpen om Svelte te implementeren in ASP.NET. Abuhakmeh (2021) heeft een manier gevonden om dit te doen. Door Svelte bestanden via Vite mee te bundelen in het ASP.NET project kan er in runtime gebruik gemaakt worden van de Svelte componenten die gedefinieerd zijn. Dit is precies wat nodig is voor dit project.

Populariteit	React	Vue	Angular	Solid	Svelte
Score	5	3	4	0	5

### 4.1.7. Conclusie

Om een conclusie te kunnen trekken over welk framework het best bij Info Support, en dus gebruikt wordt in dit project om mee te testen, wordt er een beslissingsmatrix gemaakt waarin alle frameworks een score krijgen. Deze wordt op basis van de volgende formule berekend.

$$score = s * p * t * b$$

s = support; p = populariteit; t = testbaarheid; b = bruikbaarheid

Hier komen de volgende resultaten uit.

	React	Angular	Vue	Solid	Svelte
Support	5	3	4	2	1
Populariteit	5	3	4	1	2
Testbaarheid	5	5	5	3	3
Bruikbaarheid	5	3	4	0	5
Score	<b>20</b>	<b>15</b>	<b>17</b>	<b>6</b>	<b>10</b>

Tabel 3: Totale score van de Javascript frameworks

React heeft de beste score in iedere categorie en maakt daarom een goed passend framework bij Info Support. Om deze reden wordt React gebruikt tijdens het realiseren van de oplossing als “test-case”.

Gezien Angular en Vue ook goed gescoord hebben, is het de bedoeling dat deze wel theoretisch ondersteund worden door de oplossing die gerealiseerd wordt. Hierdoor worden de meeste frameworks waar Info Support ervaring mee heeft en gebruikt in projecten voor klanten ondersteund.

## 4.2. Beantwoording deelvraag 2

“Hoe kan binnen duurzame javascript frameworks testen geschreven worden voor een front-end component die user interface design eigenschappen valideren?”

Het testen van front-end componenten is een concept waar developers al jaren mee bezig zijn. Zeker nu accessibility steeds belangrijker wordt, is de samenhang tussen ontwerp en uitvoering essentieel geworden.

### 4.2.1. Wat is een test?

Een software test is volgens IBM (z.d.) een proces om te evalueren en te verifiëren dat een software product of applicatie doet wat het moet doen. Het helpt een programmeur om moeilijk voorspelbare situaties te ontdekken en te bewaken die kunnen leiden tot softwarefouten.

Functionaliteit (Uiterlijke kwaliteit)	Ontwikkeling (innerlijke kwaliteit)	Aanpasbaarheid (toekomstbestendigheid)
Correctheid	Efficiëntie	Flexibiliteit
betrouwbaarheid	Testbaarheid	Herbruikbaarheid
Bruikbaarheid	Documentatie	Onderhoudbaarheid
Integriteit	Structuur	-

Tabel 4: Testbare kwaliteitsfactoren volgens Pan (1999)

Pan (1999) ziet dat softwarekwaliteit niet direct getest kan worden. Wel kunnen er een aantal factoren getest worden die kwaliteit zichtbaar maken zoals in Tabel 4 is weergegeven.

Voor de toepassing van deze opdracht wordt er gericht op bepaalde factoren uit de kwaliteitscategorieën: Correctheid, Testbaarheid en Structuur. Deze factoren worden binnen de context van de opdracht vertaald naar:

- **Correctheid**, In hoeverre zijn de eigenschappen van het user-interface design en het geïmplementeerde component gelijk aan elkaar?
- **Integriteit**, Correctheid in verschillende browsers
- **Testbaarheid**, Hoeveel testen kunnen gegenereerd worden op basis van de nodes van het design en het geïmplementeerde component?
- **Structuur**, Hoe complex is de geïmplementeerde structuur van het component vergeleken met de structuur van het design?

Om deze factoren te kunnen testen is er een testing framework nodig. De gekozen factoren in de context van de opdracht zijn echter zo specifiek dat er onderzocht moet worden of bestaande testing frameworks aangepast kunnen worden om de gewenste factoren mee te nemen in de test of dat er een eigen test framework geschreven moet worden.

## 4.2.2. Testing framework

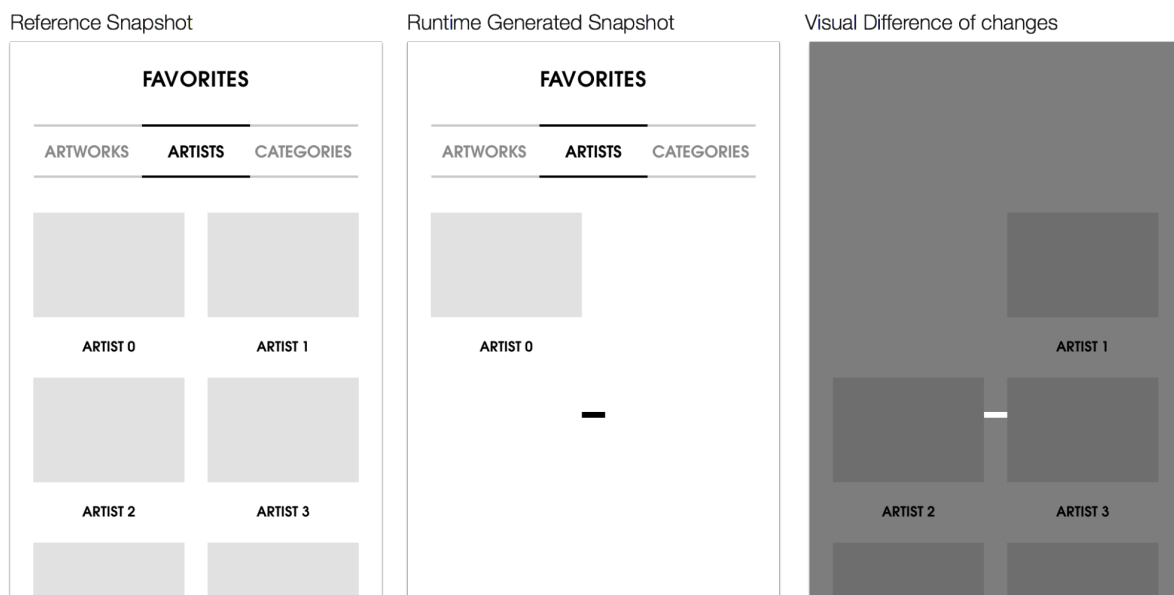
Unadkat (2022) van BrowserStack vindt Jest, Mocha of Jasmine de beste opties om javascript frontend code te testen. Het nadeel hiervan is dat deze frameworks niet goed aanpasbaar zijn voor de doeleinden van deze opdracht.

### Snapshot testing

Testing frameworks zoals Jest, Mocha en Jasmine gebruiken: snapshot testen, een techniek om de visuele staat van een component te testen (Trinh, 2020). Om een snapshot te maken wordt het component gerenderd waarna er een foto van gemaakt wordt. Deze wordt naast het component bewaard als een “correcte staat”. Wanneer er iets in het component verandert, wordt er een nieuwe snapshot gemaakt die vergeleken wordt met de “correcte staat”. Als deze niet overeenkomen gaan er alarmbellen af en ziet de ontwikkelaar dat er iets niet klopt (Snapshot Testing · Jest, 2023).

Trinh (2020) ziet dat Snapshot testing de meest voorkomende manier is om de staat van een front-end component te testen. Het vergelijkt de nieuwe staat met een vorige staat waarvan we weten dat deze klopt. Het maakt een foto en kijkt niet naar de daadwerkelijke waarden van het component (Jest, z.d.).

Deze manier van testen wordt compleet geautomatiseerd gedaan. Hierdoor heeft de gebruiker / ontwikkelaar geen invloed op de eigenschappen die getest worden. Een doel van de opdracht is om duidelijke logging te hebben tijdens het testen over wat er fout is gegaan en waarom. Snapshot testen geeft niet aan wat de waarden waren van de gefaalde tests, alleen dat er een visueel verschil geconstateerd is.



Figuur 6: Snapshot testen (Therox, z.d.)

De runtime generated snapshot wordt vergeleken met de reference snapshot. Rechts wordt het verschil laten zien.

## Nadelen

- Ieder framework verwacht dat er speciale testfiles gemaakt worden waarin de specificatie van de test in geschreven wordt.
- Frameworks gebruiken snapshot testing om visuele bugs te constateren, wat weinig semantische waarde biedt.

Deze nadelen zorgen ervoor dat de standaard test frameworks niet aanpasbaar genoeg zijn voor dit project. Een mogelijkheid is om een eigen test framework te ontwikkelen. Om te onderzoeken wat hiervoor nodig is, zijn de meest gebruikte front-end testing frameworks naast elkaar gelegd om te kijken wat alle overeenkomsten zijn. De meest gebruikte front-end testing frameworks zijn: Jest, Mocha en Jasmine (Unadkat, 2022b; Singh, z.d.; “JavaScript Unit Testing Frameworks in 2022: A Comparison,” 2022). Alle testing frameworks bevatten de volgende elementen in hun architectuur om om een test te draaien:

(I) de test specificatie, (II) het test assertion framework, (III) de test runner, en (IV) het testrapport (Architecture · Jest, 2023; Mocha - the Fun, Simple, Flexible JavaScript Test Framework, 2023; Karma - Spectacular Test Runner for Javascript, z.d.).

In het volgende diagram staan de implementaties van deze elementen beschreven per testing framework.

Framework	Specificatie	Assertion	Runner	Rapport
<b>Jest</b>	file.test.js ( Jest syntax)	Expect*	Jest worker *	Aggregated Test Result *
<b>Mocha</b>	file.test.js (mocha syntax)	Chai, assert	Mocha runner *	Spec cli *
<b>Jasmine</b>	fileSpec.js (Jasmine syntax)	Chai, assert	Jasmine runner *	Jasmine cli report *

*Tabel 5: Test-framework architectures*

*(Architecture · Jest, 2023; Mocha - the Fun, Simple, Flexible JavaScript Test Framework, 2023; Karma - Spectacular Test Runner for Javascript, z.d.)*

\* = eigen gebouwd;



### 4.2.3. Test specificatie

Voor deze opdracht is de bedoeling dat de testen automatisch gegenereerd worden op basis van het user-interface design. Dit betekent dat er in de oplossing die gerealiseerd wordt, een aantal test-specificaties gemaakt worden die op basis van de eigenschappen die het design bevat aangepast kunnen worden om de correcte eigenschappen te testen. In deelvraag 3 wordt er ingegaan op welke soorten test-specificaties gemaakt kunnen worden.

Tijdens het genereren van de specificatie moet door het test framework voor ieder te testen component het design opgehaald worden. Op basis van het design genereert het test framework de specificaties. De bedoeling is dat deze specificaties meegegeven worden aan de test runner om ze uit te voeren.

### 4.2.4. Test assertion framework

Durand & Curran (2012) geven aan een assertion een testbare of meetbare expressie is om een implementatie te vergelijken met een statement die daarover gemaakt wordt. Assertions geven vertrouwen in de test en kunnen een basis bieden voor code coverage analyse.

Er zijn libraries die deze functionaliteit bieden. Volgens Kandi, een SaaS tech bedrijf die miljoenen open source code accelerators bouwt, zijn Chai, (*11 Best JavaScript Assertion Libraries in 2023 | Kandi, z.d.*)

Dit framework heeft functies om waarden te vergelijken, exceptions op te vangen of situaties te verwachten.

### 4.2.5. Test runner

Iedere test moet uitgevoerd worden. Om dit te doen is een test runner nodig. Deze kan de test vinden, de assertion library aanroepen en het resultaat van de assertie opslaan en doorgeven aan het testrapport. In de context van deze opdracht is de runner verantwoordelijk voor de volgende functionaliteit:

- Het beschikbaar stellen van het te testen component
- Het uitlezen van het component
- Het verwerken van de test specificatie
- Het aanroepen van de assertion
- Het verwerken van het resultaat van de test

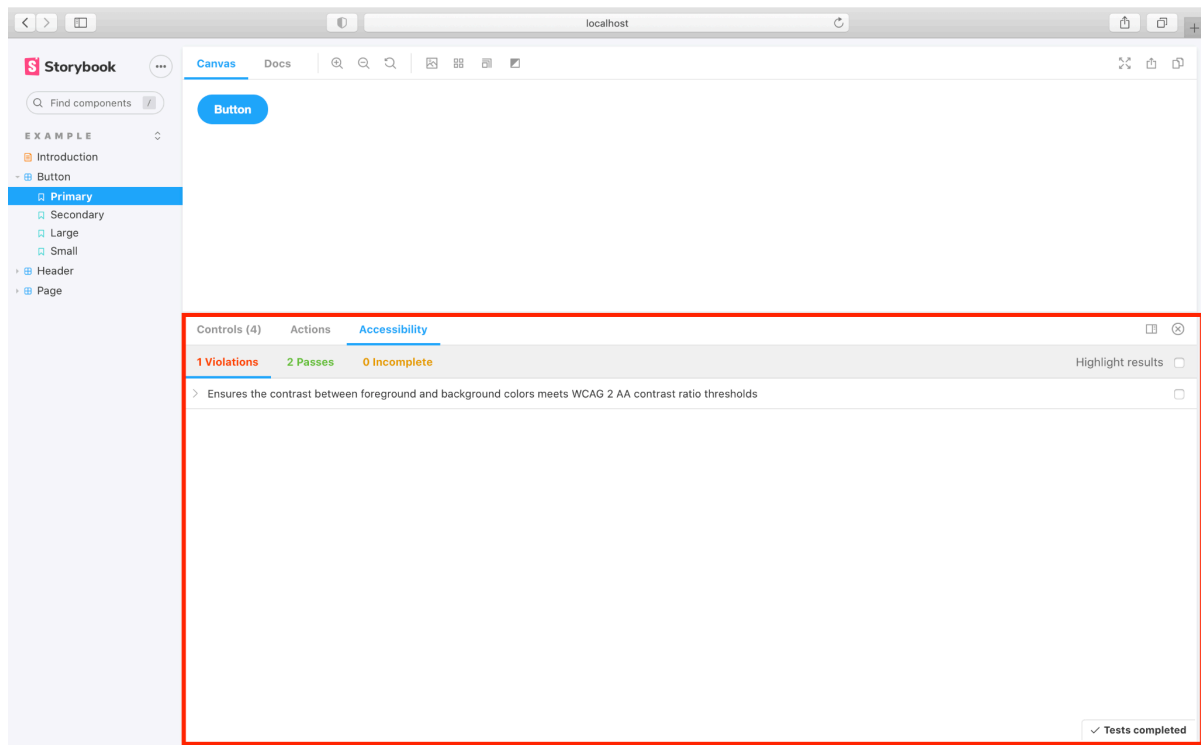
### Renderen

Om de user-interface eigenschappen van een component te testen moet het component gerenderd worden. Op die manier wordt er door de testrunner naar het component gekeken en kunnen eigenschappen van het component vergeleken worden met eigenschappen uit het user-interface design. Het liefst wordt dit niet gedaan door de applicatie te runnen want dan zijn er te veel elementen te zien op de pagina die niet interessant zijn voor het testen van het component en het duurt langer dan nodig om de applicatie op te starten.

## Storybook

Een mogelijke oplossing om component los te renderen is Storybook. Storybook is een virtuele werkplaats om user-interface componenten te ontwikkelen in isolatie (Storybook: Frontend Workshop for UI Development, z.d.). Het kan helpen om moeilijke randgevallen te vinden zonder de complete applicatie te draaien. Het biedt veel integraties met third party tools om de functionaliteit van Storybook uit te breiden. Dit noemt Storybook: “addons”.

## Addons



*Figuur 7: Voorbeeld van een Storybook addon (Install Addons, z.d.)*

*Met rood omlijnnde Storybook addon die accessibility eisen test voor een button component.*

Om de functionaliteit van Storybook uit te breiden hebben de ontwikkelaars addons bedacht. Addons kunnen ontwikkeld worden door iedereen en zijn heel belangrijk geweest voor het succes van Storybook. Er zijn onder andere addons om accessibility te testen, designs toe te voegen aan je component of om bepaalde eigenschappen van het component te highlighten (Integrations | Storybook: Frontend Workshop for UI Development, z.d.).

Er kan een addon geschreven worden voor Storybook om een user-interface design te vergelijken met het geïmplementeerde component gezien Storybook de verantwoordelijkheid neemt om de componenten te renderen en te kunnen benaderen vanuit de addon. Hiervoor kan het panel pattern (*Types of Addons*, n.d.) gebruikt worden om per component een extra paneel toe te voegen aan de UI van Storybook, die de resultaten van de gegenereerde testen laat zien aan de ontwikkelaar.

### **Voordelen**

Het grootste voordeel van Storybook is (I) dat het veel zorgen van je af neemt. Het is een productie complete omgeving die veel features en functionaliteit ondersteunt. (II) Addons zijn makkelijk te installeren in ieder project en Storybook is voor een groot deel framework abstract. Het ondersteunt: React.js, Angular, Vue.js en web-componenten (Storybook, Made for, z.d.). (III) Dit betekent dat het voor het grootste gedeelte een plug-and-play situatie is die gebruikt kan worden in veel Info Support projecten.

### **Nadelen**

Er zijn wel wat nadelen aan het gebruik van Storybook: (I) Het introduceert extra features en functionaliteiten die niet altijd nodig zijn in een project. Een subset van deze features wordt maar gebruikt voor het doel van deze opdracht. (II) Storybook is framework abstract maar niet voor alle frameworks. Het ondersteunt frameworks zoals Svelte en SolidJS niet waardoor het misschien niet toekomstbestendig genoeg is. (III) Het opzetten van Storybook vereist een complete herordening van een project als deze al bestaat. Het is een enorme ingreep om Storybook te introduceren. (IV) Addons bieden alleen ondersteuning per component. Dit houdt in dat alleen als een specifiek component bekeken wordt, hiervoor de addon tests gedraaid kunnen worden. Storybook kan niet een addon draaien die in één keer alle componenten langs kan gaan om een test rapport te maken. Dit zou handmatig moeten gebeuren.

### **Webdrivers**

Storybook heeft zijn eigen manier om gerenderde componenten te bekijken en mee te interacteren, maar als er meer precisie en controle verwacht wordt, worden web drivers gebruikt om deze functionaliteit mogelijk te maken. Een web driver is een op afstand bestuurbare interface waarmee een gebruiker op een implementatie abstracte wijze een browser kan instrueren. De bekendste voorbeelden van webdrivers zijn Selenium, WebdriverIO en ChromeDriver. Ook frameworks zoals Cypress en Playwright hebben ingebouwde webdrivers om hun testen goed uit te kunnen voeren. Onderwater wordt bij bijna alle implementaties Selenium gebruikt om de webpagina te renderen.

### **Standalone**

Het belangrijkste voor een webdriver is dat deze in een standalone versie kan draaien. Dat betekent dat deze niet verbonden is aan een testing framework. Dit testing framework vraagt om test bestanden die voldoen aan een specifieke syntaxis die niet de eigenschappen testen die belangrijk zijn voor dit project en geautomatiseerd werken waardoor er geen functionaliteit aan toegevoegd kan worden.

Een standalone webdriver houdt in dat er vanuit code een browser instantie aangemaakt kan worden die aanstuurt op de manier die de gebruiker wilt. Hiermee kunnen DOM elementen (*Introduction to the DOM - Web APIs* | MDN, 2023) en styling properties opgehaald worden maar ook acties uitgevoerd worden die een echte gebruiker simuleren zoals een muisklik of een pagina scroll.

WebdriverIO werkt in een standalone modus terwijl cypress dit niet ondersteunt. Storybook werkt heel goed op een “per component basis”, maar kan niet zo goed omgaan met een test runner die meerdere componenten tegelijk wilt testen. Storybook is ook heel groot en verwacht een herstructurering van code en een andere denkwijze bij het ontwikkelen van front-end applicaties.

## Combinatie van Storybook en Webdrivers

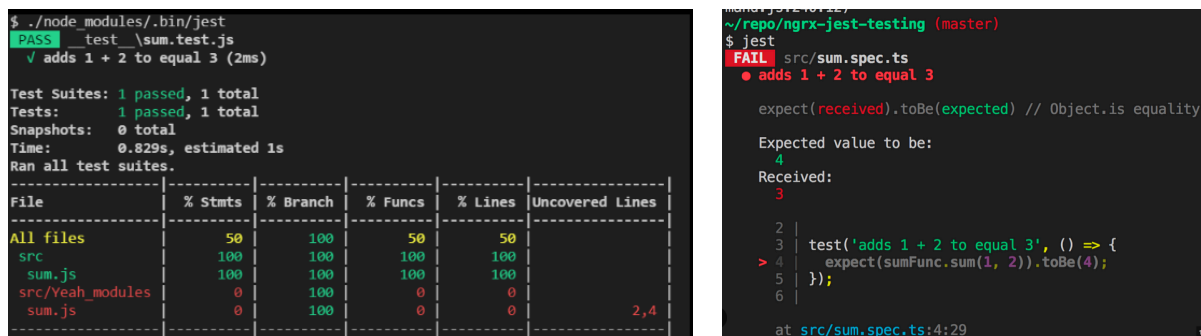
Om gebruik te kunnen maken van de flexibiliteit die WebDriverIO biedt moet het component wel gerenderd worden, zodat de webdriver er bij kan om zijn testen uit te voeren.

Een probleem is dat er weinig manieren zijn om componenten te renderen zonder een eigen oplossing voor dat specifieke framework te bouwen. Dit is geen goede manier omdat dit heel veel overhead genereert, zeker omdat er alleen maar getest hoeft te worden. Voor ieder framework is de manier om te renderen compleet anders.

Gelukkig biedt Storybook hier wel een oplossing voor. Storybooks framework abstracte rendering kan gebruikt worden om de componenten beschikbaar te stellen voor WebDriverIO. Storybook heeft nog steeds de nadelen zoals eerder beschreven, maar deze zijn altijd beter dan de extra complexiteit en tijd die het bouwen van een eigen oplossing met zich mee brengt. Het renderen van het component is op dit moment niet de hoofdfocus van dit project.

### 4.2.6. Testrapport

Het is belangrijk dat het resultaat van een test goed opgeslagen wordt, zodat aan het einde van de testrun het resultaat weergegeven kan worden en de gebruiker een duidelijk overzicht krijgt wat er goed of fout gaat.



```
./node_modules/.bin/jest
PASS  _test_\sum.test.js
  ✓ adds 1 + 2 to equal 3 (2ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:  0 total
Time:        0.829s, estimated 1s
Ran all test suites.
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
All files	50	100	50	50	
src	100	100	100	100	
sum.js	100	100	100	100	
src/Yeah_modules	0	100	0	0	
sum.js	0	100	0	0	2,4

```
~/repo/ngrx-jest-testing (master)
$ jest
FAIL  src/sum.spec.ts
  ● adds 1 + 2 to equal 3

    expect(received).toBe(expected) // Object.is equality

    Expected value to be:
      4
    Received:
      3

       2 |
       3 |   test('adds 1 + 2 to equal 3', () => {
       4 |     expect(sumFunc.sum(1, 2)).toBe(4);
       5 |   });
       6 |
     at src/sum.spec.ts:4:29
```

Figuur 8: Jest testrapport

Informatie	Kwaliteitsfactor	Toelichting
Naam van de test	-	Geeft aan welke test er gerund is. Ook staat hier bij of deze geslaagd of gefaald is.
Totaal aantal gerunde testen	Testbaarheid	Geeft aan hoeveel testen er zijn gegenereerd en gerund zijn tijdens deze test run.
Aantal geslaagde testen	Testbaarheid	Geeft aan hoeveel testen er geslaagd zijn. Als alle testen geslaagd zijn, is de testrun geslaagd.
Aantal gefaalde testen	Testbaarheid	Geeft aan of er testen zijn die niet slagen. Als dit zo is, dan is de testrun gefaald.
Waarom een test gefaald is	Correctheid	Als een test gefaald is, wordt er aangegeven waar waarom deze test gefaald is, zodat deze zo snel mogelijk opgelost kan worden.
Browser waarin de test gerund is.	Integriteit	Welke browser is gebruikt het component te testen, zodat de gebruiker weet of een bepaalde browser het verkeerd rendert.
Run duur van één test	Testbaarheid	In milliseconden. Geeft aan hoe lang een test heeft geduurd.
Totale duur van de testrun	Testbaarheid	In seconden. Geeft aan hoelang een testrun heeft geduurd.
Verskil in complexiteit	Structuur	Hoe veel meer of minder nodes heeft de DOM vergeleken met het design.

Tabel 5: Informatie weergegeven in een testrapport

In Figuur 8 is te zien welke informatie Jest weergeeft aan het einde van een testrun. De naam van iedere test wordt weergegeven samen met of deze test geslaagd of gefaald is. Als deze gefaald is wordt er ook aangegeven waar de test gefaald is in de code. Het is belangrijk dat de informatie in Tabel 5 verwerkt wordt in een testrapport (Geek, 2022). Deze informatie geeft een inzicht in de geteste kwaliteitsfactoren die belangrijk zijn voor dit project.

### 4.2.7. Conclusie

Software kwaliteitsfactoren zoals: correctheid, integriteit, testbaarheid en structuur kunnen vertaald worden binnen de context van de opdracht naar:

- **Correctheid**, In hoeverre zijn de eigenschappen van het user-interface design en het geïmplementeerde component gelijk aan elkaar?
- **Integriteit**, Correctheid in verschillende browsers
- **Testbaarheid**, Hoeveel testen kunnen gegenereerd worden op basis van de nodes van het design en het geïmplementeerde component?
- **Structuur**, Hoe complex is de geïmplementeerde structuur van het component vergeleken met de structuur van het design?

De bestaande testing frameworks bieden geen ondersteuning om deze kwaliteitsfactoren te testen en er moet dus een nieuw framework bedacht worden die wel in staat is om die eisen te testen. Door de bestaande testing frameworks te vergelijken (Jest, Mocha en Jasmine) zijn er parallellen te zien binnen de architectuur en procesgang die gebruikt kunnen worden om een eigen test runner te schrijven. De testing frameworks bevatten namelijk allemaal een testspecificatie, assertion framework, test runner en een test rapport.

Voor het runnen van de test wordt het front-end component gerenderd middels Storybook en kan dit component vervolgens uitgelezen worden middels een webdriver. In dit geval WebDriverIO. Nadat de testrunner de test heeft uitgevoerd, wordt er een testrapport opgesteld dat informatie biedt om inzicht te geven in de geteste kwaliteitsfactoren.

## 4.3. Beantwoording deelvraag 3

“In welke mate kunnen de component testen automatisch de eigenschappen van het user interface design gebruiken voor de specificatie van de test?”

Om een test te kunnen automatiseren moeten alle parameters voor de test te verkrijgen zijn via code. Omdat eigenschappen van het design vergeleken worden met de eigenschappen van het component zullen beiden dus bekend moeten zijn voordat de test gegenereerd kan worden.

### 4.3.1. Design platform

Zowel Stevens (2023) als Cahill and May (2023) en Maze (2023) geven recentelijk aan dat er meerdere design platforms beschikbaar zijn om designs mee te ontwikkelen. Gezien het niet mogelijk is om te betalen voor een design platform voor dit project is ervoor gekozen om alleen platforms weer te geven die een gratis versie beschikbaar hebben.

Voor dit project wordt er ook verwacht dat het platform ondersteuning biedt voor het uitlezen van de design file op aanvraag.

Platform	Key features	Design file op aanvraag
Adobe XD	<ul style="list-style-type: none"><li>- Workflow integration</li><li>- Element creation (interactable &amp; reusable)</li></ul>	Adobe XD biedt een scriptable API om plugins mee te ontwikkelen. Adobe geeft aan dat plugins ondersteuning bieden voor het creëren van designs, het samenwerken aan designs en het delen van designs met andere mensen. Het biedt dus geen uitkomst voor dit project. (Adobe XD APIs for Developers and Scripters, z.d.).
Invision	<ul style="list-style-type: none"><li>- Outline user journeys</li><li>- Create design handoff features</li></ul>	Invision biedt veel integraties met andere tools waaronder Google Docs en Jira. maar heeft geen manier om handmatig of geautomatiseerd design files op te vragen (Invision, 2023).
Framer	<ul style="list-style-type: none"><li>- Close to production prototyping</li><li>- First party HTML &amp; CSS integration</li></ul>	Framer biedt “Custom page code” en “code overrides”. Manieren om de user interface binnen Framer aan te passen en functionaliteit toe te voegen. Het heeft geen manier om designs te exporteren (Framer: Developers, z.d.).
Figma	<ul style="list-style-type: none"><li>- Cloud based design</li><li>- Collaboration tools</li><li>- Browser based interface</li></ul>	Figma biedt een REST API waarmee een JSON representatie van het design waar specifieke eigenschappen uitgehaald kunnen worden (Figma, z.d.).

Tabel 6: Voorbeeld van een Storybook addon (Install Addons, z.d.)

Met rood omliggende Storybook addon die accessibility eisen test voor een button component.

In Tabel 6 is te zien dat er meerdere design platforms beschikbaar zijn die allemaal hun eigen redenen hebben om te gebruiken. Er is maar één platform dat de ondersteuning biedt die nodig is: Figma. Door middel van de REST API kan er een JSON representatie van het design uitgelezen worden waarin de eigenschappen staan die nodig zijn om te vergelijken.

### 4.3.2. Design eigenschappen

Type	Overeenkomende HTML tag	Functie	Onderscheidende eigenschappen
Text	<p>, <h1>, <h2>, <h3>	Weergeven van tekst	font-family, font-size, text-alignment, characters.
Rectangle	<div>, <span>	Weergeven van een rechthoek als achtergrond of decoratie	border-radius
Table	<table>	Weergeven van tabellen	number-of-rows, number-of-columns,
Vector	<svg>	Bevat basiseigenschappen van veel andere nodes zoals: rectangle en andere vorm gerelateerde nodes.	fills, strokes, stroke-weight
Line	<hr>	Weergeven van lijnen	width, stroke-width

Tabel 7: Verschillende Figma design nodes (Node Types | Plugin API, z.d.)

Een user-interface design voor een component bestaat uit verschillende elementen genaamd nodes. Een node bevat informatie over dat element van het ontwerp. Design platform Figma onderscheidt verschillende soorten nodes (Node Types | Plugin API, z.d.). In Tabel 7 staan een aantal van die nodes die gebruikt kunnen worden om mee te vergelijken omdat ze lijken op eigenschappen van een geïmplementeerd component.

Figma (z.d.) biedt een Restful web API aan waarmee deze nodes, inclusief eigenschappen, opgehaald en ingezien kunnen worden. Deze nodes en eigenschappen kunnen dan vergeleken worden met de eigenschappen van het geïmplementeerde component.

### 4.3.3. Component eigenschappen

Componenten geschreven in Javascript frameworks worden door Storybook omgezet naar renderbare HTML (Story Rendering, z.d.). Deze HTML code bestaat uit twee onderdelen: een Document Object Model (DOM) en een Cascading Style Sheet (CSS). De DOM is een boomstructuur bestaande uit HTML tags die aangeven welke content er op de pagina weergegeven moet worden (Introduction to the DOM - Web APIs | MDN, 2023). De CSS is een bestand waarin voor iedere node op de boomstructuur van de DOM aangegeven kan worden hoe deze gepresenteerd wordt (Keller & Nussbaumer, 2009).

Een webdriver wordt gebruikt om door middel van een query, CSS waarden op te halen die te vinden zijn op een HTML tag. De webdriver kiest de juiste HTML tag op basis van een selector. Dit kan een CSS class, id of andere property (Selectors | WebdriverIO, z.d.).



#### 4.3.4. Vergelijken

Eenheid	Afkorting	Staat gelijk aan
Centimeters	cm	1cm = 37.8px = 25.2/64in
Millimeters	mm	1mm = 1/10th of 1cm
Quarter-millimeters	Q	1Q = 1/40th of 1cm
Inches	in	1in = 2.54cm = 96px
Picas	pc	1pc = 1/6th of 1in
Points	pt	1pt = 1/72nd of 1in
Pixels	px	1px = 1/96th of 1in
Hexadecimale	#	-
RGB	rgba	-
Percentage	%	-

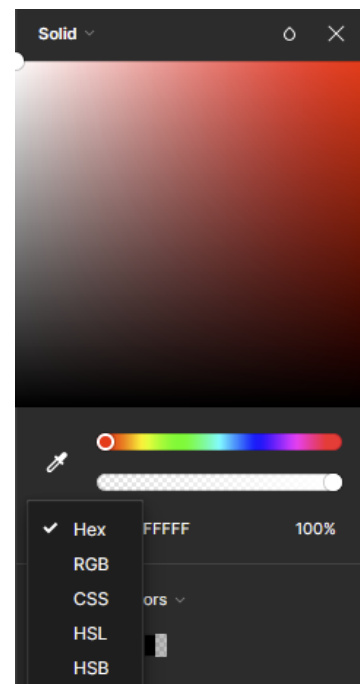
Tabel 8: Absolute CSS eenheden (CSS Values and Units - Learn Web Development | MDN, 2023)

Het is belangrijk dat Figma en CSS gebruik maken van dezelfde eenheden, zodat er zo accuraat en makkelijk mogelijk vergeleken kan worden of de waarden gelijk zijn aan elkaar. In Tabel 8 staan volgens MDN de absolute eenheden die CSS kent (CSS Values and Units - Learn Web Development | MDN, 2023).

Figma is een stuk onduidelijker over het gebruik van eenheden en geeft in de API documentatie alleen aan of de waarde in de JSON representatie in de code een number, string of andere eenheid uitgedrukt kan worden. Bash (2023) geeft als antwoord op de vraag: "What Is the Measuring Unit in Figma?", gesteld op 8 maart 2023, dat Figma gebruik maakt van pixels voor de dimensionale waarden zoals lengte, breedte en afstand.

Voor kleur geeft Figma in de kleur picker (Figuur 9) aan dat het zowel hexadecimale als RGB-eenheden ondersteunt. Voor waarden waar percentages verwacht worden, kan Figma hier ook mee omgaan gezien er voortdurend percentages gebruikt worden om doorzichtigheid waarden te erkennen.

Door gebruik te maken van de pixel, hex, rgb en percentage eenheden kan er goed vertaald worden tussen CSS en Figma. Er moet alleen wel goed gekeken worden naar welke eenheid Figma gebruikt voor waarden waarvan het niet gelijk duidelijk is.



Figuur 9: Eenheden gebruikt voor de kleur picker in Figma (2023)

### 4.3.5. Conclusie

Om een test te kunnen automatiseren is moeten alle parameters voor de test te verkrijgen zijn via code. Omdat eigenschappen van het design vergeleken worden met de eigenschappen van het component zullen beiden dus bekend moeten zijn voordat de test gegenereerd kan worden.

Het design kan middels de web API van Figma verkregen worden in JSON formaat. De JSON bevat een boomstructuur waarin alle elementen van het design beschreven staan, inclusief eigenschappen.

De webdriver kan middels selectors zoals een CSS class, id of andere property een HTML document opvragen. Dit HTML document bevat een DOM boom waarin de structuur van het component beschreven wordt en een zogeheten stylesheet geschreven in CSS waarin de bijbehorende visuele eigenschappen vermeld staan.

Doordat Figma en de HTML/CSS gebruik maken van andere soorten eenheden moet er voor sommige eigenschappen afgesproken worden welke eenheid gebruikt wordt. Door alleen gebruik te maken van de pixel, hex, rgb en percentage eenheden kunnen beide waarden met elkaar vergeleken worden.

## 4.4. Onderzoeksvraag

*“Hoe kan een user interface test ontwikkeld worden voor een front-end component die eigenschappen van een user-interface design vergelijkt met de geïmplementeerde code, in een nieuwe duurzame javascript omgeving?”*

Door te kijken naar welke eigenschappen Info Support belangrijk vindt in een Javascript framework: (I) support, (II) populariteit, (III) bruikbaarheid en (IV) testbaarheid, is er onderzocht welk framework er gebruikt wordt om het project mee te ontwikkelen. React heeft uit de vergelijking tussen React, Angular, Vue, Solid en Svelte de hoogste score gekregen op de voornoemde eigenschappen en wordt gebruikt in dit project.

Door te onderzoeken hoe een test gemaakt kan worden zijn er meerdere factoren ontdekt die software kwaliteit kunnen testen: correctheid, integriteit, testbaarheid en structuur. Door eigenschappen van het design te vergelijken met de eigenschappen van het component kunnen deze factoren getest worden. Om dit te doen moet er een test runner geschreven worden gezien de populaire bestaande test runners voor Javascript deze functionaliteit niet aan kunnen.

Om het component te renderen is Storybook een vereiste. Het biedt voor dit project geen toegevoegde waarde om zelf een front-end render pipeline op te zetten. Door gebruik te maken van Storybook worden de meeste frameworks ondersteund, en scheelt het in deze fase van dit project veel tijd. Door een webdriver te gebruiken kan er gekeken worden op een HTML pagina via code naar de eigenschappen die het component bevat. Dat houdt in dat de DOM structuur gescand wordt en eigenschappen van het component eruit gehaald worden op basis van CSS selectors.

Omdat het goede integratiemogelijkheden biedt via een API, wordt Figma gebruikt als design platform. Figma levert het document op in een node structuur waarmee de eigenschappen van het design opgehaald kunnen worden. Deze worden dan vergeleken met de eigenschappen van het component om een test te genereren.

Om de vergelijking te maken kan geen gebruik gemaakt worden van een standaard testing framework. Die hebben niet de juiste focus om de gewenste functionaliteit te ondersteunen. Hiervoor moet dus een nieuw testing framework ontwikkeld worden om een test specificatie te kunnen genereren, de test te runnen, de assertion uit te voeren en het testrapport te genereren.

Als de testrun klaar is wordt het testrapport getoond en kan er op basis van informatie zoals: het aantal gegenereerde tests, het aantal geslaagde of gefaalde tests en de complexiteit van de componenten, inzicht gegeven worden in de kwaliteitsfactoren die een indicatie kunnen geven voor een hoge softwarekwaliteit.

## 5. Realisatie

---

Het beroepsproduct heeft het doel om de consistentie tussen het user-interface design en het geïmplementeerde component te verbeteren. Om dit te doen is er onderzocht dat er een tool gebouwd moet worden die (I) kan communiceren met het design platform Figma, (II) kan integreren met een te testen project, (III) kan communiceren met componenten om eigenschappen te vergaren en (IV) een test kan starten waarna een rapport wordt laten zien met de resultaten. Dit product moet gebouwd worden en daarvoor zijn een aantal zaken belangrijk die in dit hoofdstuk besproken worden zoals hoe het product gebouwd is, welke tools er gebruikt zijn en wat er uiteindelijk gerealiseerd is.

### 5.1. Architectuur

Er is een duidelijke architectuur gerealiseerd die de procesgang van een test runner nabootst. In het onderzoek is er waargenomen dat een test runner de volgende onderdelen bevat: (I) de test specificatie, (II) het test assertion framework, (III) de test runner en (IV) het testrapport.

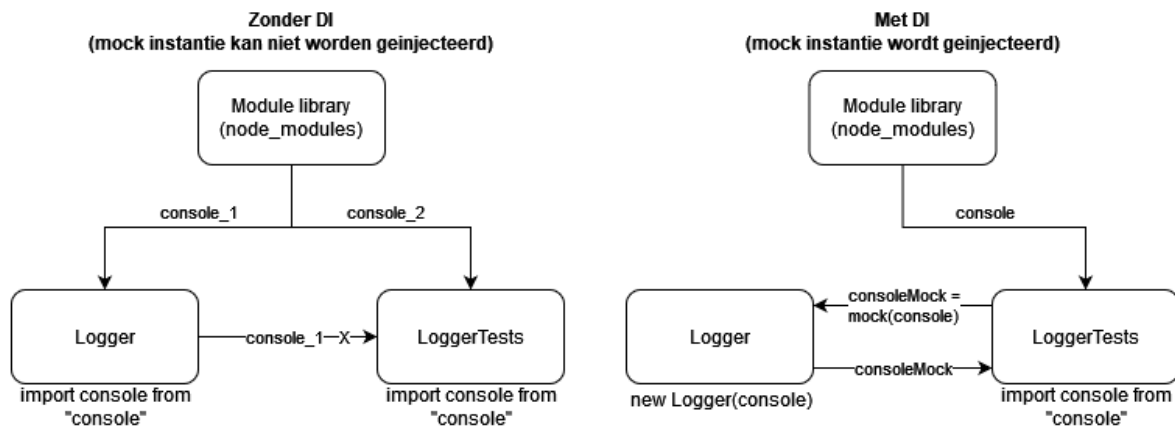
Er is gekozen om het project “Mystique” te noemen, gezien Info Support andere projecten bouwt die ook vernoemd worden naar X-men characters. Een voorbeeld hiervan is “Stryker”.

#### 5.1.1. Node.JS CLI application

Mystique gaat integreren met bestaande front-end projecten die voornamelijk geschreven zijn in Javascript of Typescript. Om deze samenwerking zo vloeiend mogelijk te laten verlopen is er gekozen, boven andere talen en technieken zoals Python, c# of Java, om een node.js applicatie te maken. Decan et al. (2018) geeft aan dat voor javascript projecten Node.JS en npm meer gebruikt worden dan andere package managers. Als het publiek toegang wil krijgen tot Mystique zal deze beschikbaar moeten zijn binnen package managers. De ondersteuning van npm is daarbij heel belangrijk. Npm ondersteunt namelijk alleen maar Node.js packages (Npm About, z.d.).

Om zo min mogelijk overhead te creëren is ervoor gekozen om geen gebruik te maken van een javascript bundler. Bundlers kunnen javascript tot 1 file samen stoppen zodat het over het internet zo makkelijk mogelijk bij de gebruiker komt. Voor een npm package is dit niet nodig en kan voor Typescript projecten net zo goed de TSC compiler gebruikt worden om de applicatie om te zetten naar een javascript distribution folder die geupload kan worden op NPM. (*You May Not Need a Bundler for Your NPM Library*, 2022).

## 5.1.2. Dependency injection



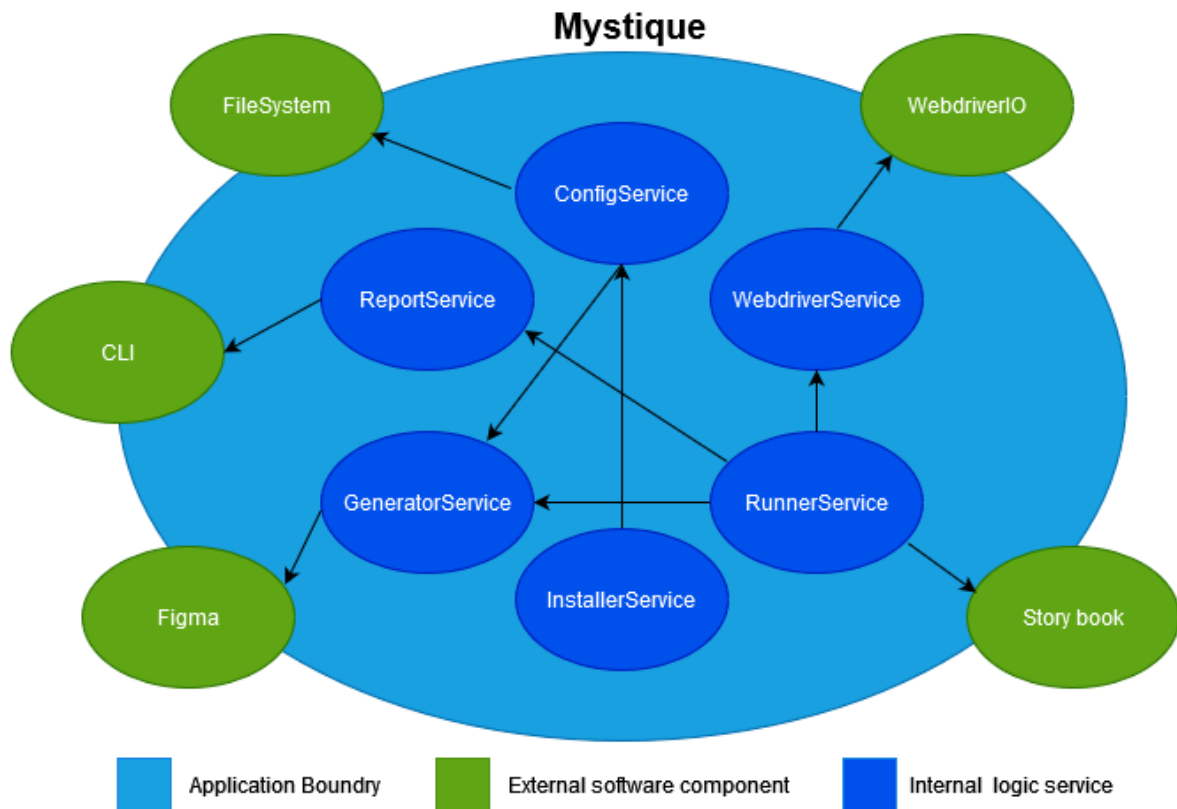
Figuur 10: Het gebruik van dependency injection om ESM te mocken

Gezien het product gebruik maakt van Typescript waardoor ECMAScript modules (ESM) (JavaScript Modules - JavaScript | MDN, 2023) een gegeven zijn, is het belangrijk dat deze modules gemockt kunnen worden wanneer dat nodig is in een unit test (Zhu et al., 2020). Als een ESM geïmporteerd wordt in een class dan kan deze niet gemockt worden door de test class omdat deze allebei een andere instantie van de module importeren. Hierdoor kan de mocking library die module niet aanpassen of bespioneren, zoals te zien in Figuur 10. De Logger class uit het figuur importeert zelf een instantie van console en de LoggerTests class kan niet bij deze instantie komen om deze te mocken.

Door gebruik te maken van dependency injection (DI) heeft de ontwikkelaar zelf de controle over de instantie van de ESM die meegegeven wordt aan de class (Smith et al., 2007). In figuur 10 is te zien dat de LoggerTests class zelf een mock kan maken van zijn geïmporteerde instantie en geeft deze door middel van de constructor aan Logger mee.

Voor dit project is ervoor gekozen om gebruik te maken van TypedInject (Nicojs, z.d.) als dependency injection library omdat het is ontwikkeld door Nico Jansen, een medewerker van Info Support en daardoor Info Support veel ervaring heeft met TypedInject. Een ander voordeel is dat het een 100% type safe injection library is, waardoor het heel fijn samenwerkt met Typescript.

### 5.1.3. Landschap



Figuur 11: Architectuur diagram

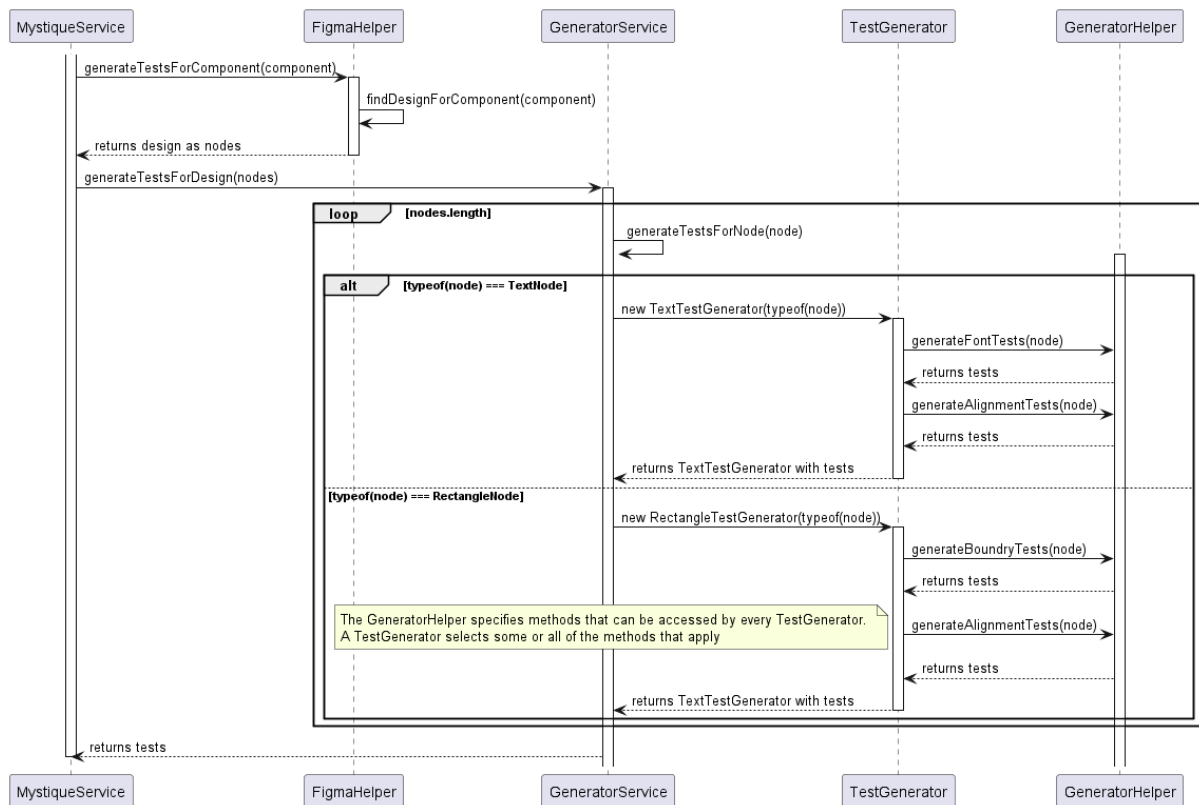
In Figuur 11 is te zien dat er zes services gemaakt zijn die de kern van de applicatie bevatten. Hierin worden de testen gegenereerd en gerund, de applicatie wordt geïnstalleerd en het configuratiebestand wordt uitgelezen. Deze services zijn in Figuur 11 als blauw aangegeven.

Er wordt gecommuniceerd met softwarecomponenten buiten de applicatie. Dit zijn externe applicaties, services of tools die nodig zijn om de applicatie correct te laten werken. Deze worden in het groen weergegeven en staan buiten de applicatie. Er is altijd een service verantwoordelijk voor de communicatie met een van deze externe softwarecomponenten.

De gehele lichtblauwe rand om de zes services is de grens van de applicatie. Alles binnen deze rand wordt meegeleverd in de package dat uiteindelijk geleverd gaat worden aan de gebruiker.

In Bijlage 3, een flowchart, is te zien hoe deze services met elkaar communiceren in de context van een test run die uitgevoerd wordt.

### 5.1.4. Test generatie



Figuur 12: Test generation sequence diagram

Een belangrijk en complex onderdeel van de applicatie is het genereren van de test. Een software kwaliteitseis opgesteld in het plan van aanpak verwacht dat er zo min mogelijk geduplicateerde code nodig is voor het genereren en uitvoeren van testen voor verschillende componenten. In Figuur 12 staat beschreven hoe het genereren op dit moment werkt. De generator service kijkt welk type de node is en voert op basis daarvan een generatie strategie uit die de correcte testen genereert. Alle mogelijke testen die gegenereerd kunnen worden staan beschreven in de GeneratorHelper. Iedere TestGenerator heeft hier toegang toe en selecteert de juiste voor de node-type.

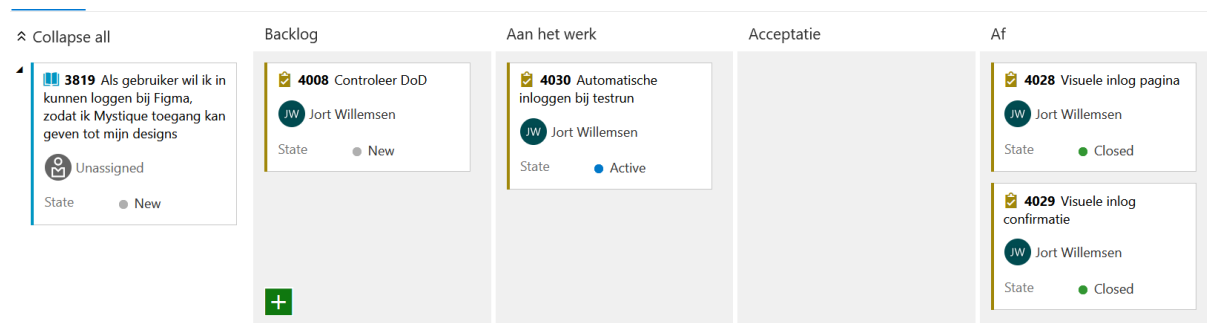
Dit zorgt ervoor dat iedere mogelijke node gebruik kan maken van dezelfde tests en de test generatie niet afhankelijk is van het type van de node. Hierdoor hoeven alle soorten testen maar één keer beschreven te worden en kan iedere keer dat er een nieuwe node testbaar gemaakt moet worden een nieuwe TestGenerator gemaakt worden die de juiste tests selecteert voor dat specifieke node-type.

## 5.2. Way of working

Er is gekozen om te werken met de werkmethode: Kanban. Kanban is een methode om kenniswerk-diensten (processen) te definiëren, managen en verbeteren. De kracht van Kanban zit vooral in de kunst om werk inzichtelijk te krijgen en een goede prioriteit te kunnen stellen aan dit werk (Agile Scrum Group, 2022).

Scrum kent duidelijke rollen met verantwoordelijkheden, zoals de product owner, scrum master en developer, die goed te vervullen zijn in een team van meer dan drie personen. In een team van één of twee personen zoals bij dit project is dit niet gewenst omdat één iemand dan meerdere rollen moet vervullen. Scrum verwacht ook een duidelijke feedback loop die een vooraf bepaalde frequentie bijgehouden wordt. Voor een project zoals dit kan dat onnodige tijd kosten die de tijd wegneemt van de kern van het project (West, z.d.).

### 5.2.1. Kanban



Figuur 13: Voorbeeld van taken in het kanbanbord

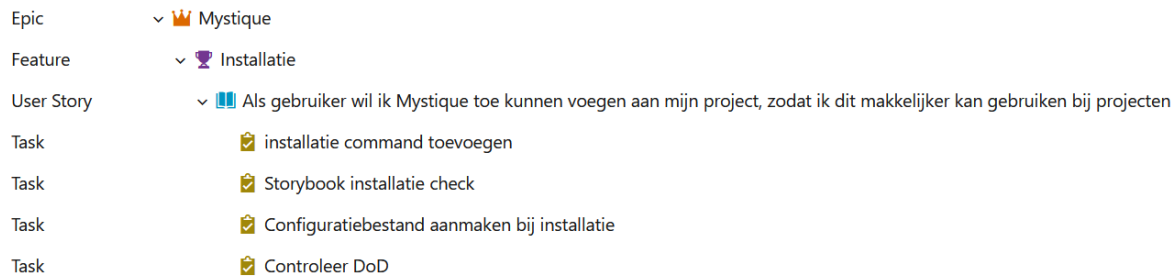
Er is een kanbanbord opgezet met de processtappen: Backlog, Aan het werk, Acceptatie en Af. Iedere taak doorloopt deze stappen. Op basis van deze stappen wordt de situatie van een taak zichtbaar voor iedere stakeholder die dit belangrijk vindt. In Figuur 13 is te zien hoe de taken verdeeld zijn. Alle taken die op de backlog staan, moeten nog gedaan worden, alle taken onder “aan het werk”, worden op dit moment actief ontwikkeld. Dit is er meestal maar één gezien de student maar aan één taak tegelijkertijd kan werken. In uitzonderlijke situaties kan het voorkomen dat het uitvoeren van de taak onvoorziene problemen met zich meebrengt. Als dit zo is, kan er voor gekozen worden deze taak halverwege de uitvoering uit te stellen. In dat geval blijft de taak onder de “aan het werk” kolom staan, zodat het inzichtelijk blijft welke taken al iets aan gedaan is.

De acceptatie kolom bevat taken die volgens de student af zijn volgens de gestelde acceptatiecriteria en de algemene definition of done, maar nog niet goedgekeurd zijn door de opdrachtgever omdat deze het nog niet gezien heeft of expliciet heeft afgekeurd waardoor er aanpassingen gedaan moeten worden aan de taak.

Als een taak af is wordt deze onder de kolom gezet met dezelfde naam. Als een taak hier staat betekent het dat deze is goedgekeurd door de opdrachtgever en student omdat deze voldoet aan de gestelde acceptatiecriteria en algemene definition of done. Er hoeft dan verder niks meer gedaan te worden met de taak.



## 5.2.2. Taakstructuur



Figuur 14: Structuur van taken op het kanbanbord

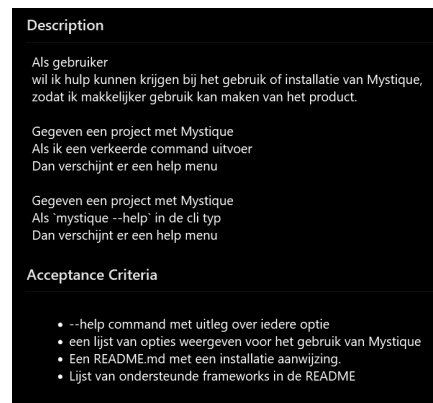
Om alle taken overzichtelijk te houden is er een structuur aangebracht in de uit te voeren taken. In Figuur 14 staat weergegeven hoe deze structuur werkt. De taken zijn ingedeeld in epics, features, user stories en tasks. Deze structuur kan gebruikt worden bij veel soorten agile werkmethoden en is de standaard binnen Azure devops, waarin dit project beheerd wordt.

Een epic is de hoogste laag en geeft weer bij welk product de taak hoort. In dit geval is dat Mystique, de naam van de testtool. In dit project zijn er drie epics: Mystique, demo project en scriptie. Onder de epics zijn features gedefinieerd. Deze features zijn een verzameling van user stories die samen een productie klaar stuk software zijn (Chcomley, 2023).

## 5.2.3. Definition of done & acceptatie criteria

Er is besloten om tijdens dit project alle voor alle user stories een algemene definition of done op te stellen. Deze wordt gebruikt om standaard criteria te stellen aan alle user stories die opgepakt worden. De definition of done voor dit project luidt als volgt:

- De software die gebouwd wordt of aangepast wordt, moet een test coverage van minimaal 85% bedragen op statement, function, branch en file niveau.
- Alle code die veranderd wordt moet via git naar de master branch op Azure Devops gepusht zijn.
- De commits moeten door de CI pipeline geverifieerd worden.
- De code moet voldoen aan de Prettier guidelines zoals opgesteld zijn voor het project.
- De opgestelde handmatige acceptatietesten moeten voldaan zijn.



Figuur 15: Voorbeeld van beschrijving en acceptatiecriteria voor een Story

De definition of done voor dit project waarborgt de softwarekwaliteit van het beroepsproduct door de styling, en testen nauwlettend in de gaten te houden. De acceptatiecriteria worden per story opgesteld en zijn gericht op de businesswaarde van een story. In Figuur 15 is hier een voorbeeld van te zien.

### 5.2.4. Afwijkingen planning

Tijdens de onderzoeksfase is de planning aangepast. Gezien het onderzoek meer tijd kostte dan dat de student van tevoren gedacht had, mede door het bouwen van de prototypen, is ervoor gekozen om de planning meer ruimte te bieden voor de onderzoeksfase. Gelukkig geeft het bouwen van prototypen een voorsprong op de realisatiefase en kon die code hergebruikt worden voor het eindproduct waardoor de herziene planning correct gevolgd kon worden. De opdrachtgever en de student zijn zowel tevreden met de initiële planning in Bijlage 4.1 als de herziene planning in Bijlage 4.2.

## 5.3. Softwarekwaliteit

Voor een product dat andere ontwikkelaars gaan gebruiken is het belangrijk dat de kwaliteit van de software die opgeleverd wordt door de student nauwlettend in de gaten en bijgehouden wordt. Om dit voor elkaar te krijgen zijn er een aantal technieken die ingezet kunnen worden zoals een CI pipeline en linting. Deze technieken staan vermeld in de definition of done zodat deze niet vergeten worden tijdens het ontwikkelproces.

### 5.3.1. Unit testen

Khorikov (2020) geeft in zijn boek: *Unit Testing Principles, Practices, and Patterns: Effective testing styles, patterns, and reliable automation for unit testing, mocking, and integration testing with examples in C#* aan wat het doel is van unit testen. Als code moeilijk te testen is, dan is dat een sterke indicatie dat de code verbeterd moet worden en het promoot een lage koppeling tussen softwarecomponenten waardoor code schaalbaarder wordt. Ook zorgt unit testen volgens Khorikov dat unit testen de duurzaamheid van een project verhoogt. Hoe meer code er geschreven wordt hoe moeilijk het wordt om de impact van een verandering in te schatten. Unit testen kan hierbij helpen.

In dit project worden unit testen geschreven op het Node.js platform door middel van Jest. Jest (z.d.) is een JavaScript testing framework ontworpen door Facebook om Node.JS omgevingen te testen. Het heeft goede ondersteuning voor Typescript. De belangrijkste reden om Jest te kiezen is de ondersteuning voor ECMAScript Modules (*ECMAScript Modules · Jest*, 2023). Het is een experimentele implementatie maar werkt wel goed om alle modules te importeren die nodig zijn tijdens de unit tests.

#### Mocking

Voor unit testen is het belangrijk om modules die je niet wilt testen weg te mocken zodat deze de test niet beïnvloeden en je geïsoleerde code aan het testen bent. Er is gekozen voor Sinon als mocking library omdat Info Support hier ervaring mee heeft in de Stryker projecten die zij ontwikkelen en onderhouden. Sinon werkt met ieder unit testing framework en is een drop-in oplossing voor mocking, spying en stubbing (*Sinon.JS - Standalone Test Fakes, Spies, Stubs and Mocks for JavaScript. Works With Any Unit Testing Framework.*, z.d.).

## Resultaten

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	97.82	86.66	95.23	97.76	
helpers	100	100	100	100	
AxiosHelper.ts	100	100	100	100	
FigmaFileHelper.ts	100	100	100	100	
GeneratorHelper.ts	100	100	100	100	
GenericHelper.ts	100	100	100	100	
WebdriverHelper.ts	100	100	100	100	
services	96.29	75	90.47	96.25	
ConfigService.ts	94.44	100	83.33	94.44	72
FigmaApiService.ts	100	100	100	100	
InstallService.ts	92	60	80	92	66-67
RunnerService.ts	100	100	100	100	
Test Suites: 8 passed, 8 total					
Tests: 57 passed, 57 total					
Snapshots: 0 total					
Time: 7.644 s, estimated 10 s					
Ran all test suites.					

Figuur 16: Unit test resultaat voor Mystique

In Figuur 16 is te zien dat er in totaal 57 unit testen zijn voor de applicatie waarvoor de minimale coverage van 85% behaald is.

Unit testen bieden een goed inzicht in de werking van de applicatie maar kunnen niet alles testen. Gezien de applicatie nauw integreert met bestaande projecten is het moeilijk om unit testen te schrijven die deze situaties nauwkeurig kunnen nabootsen. Om toch zeker te zijn dat de applicatie goed werkt staan in de definition of done een aantal handmatige testen beschreven die uitgevoerd moeten worden na het ontwikkelen van een user story.

### 5.3.2. Formatting

Dit project wordt alleen ontwikkeld door de student maar moet na de stageperiode overgedragen worden aan Info Support. De opdrachtgever heeft met deze gedachte een requirement opgesteld om code formatting te introduceren. De opdrachtgever geeft aan te werken in Visual Studio Code en verwacht een oplossing voor deze IDE.

Er zijn verschillende opties voor code formatting: Visual Studio Code heeft een ingebouwde code formatter die Javascript, Typescript, HTML, CSS en JSON ondersteunt (*Basic Editing in Visual Studio Code*, 2021). In Bijlage 2 staan de meest gebruikte talen die in Mystique gebruikt worden, waarin aangegeven wordt dat dit TypeScript, JavaScript, CSS, HTML en C# zijn. De ingebouwde formatter van Visual Studio Code ondersteunt geen C# en kan daarom niet zonder modificaties gebruikt worden.

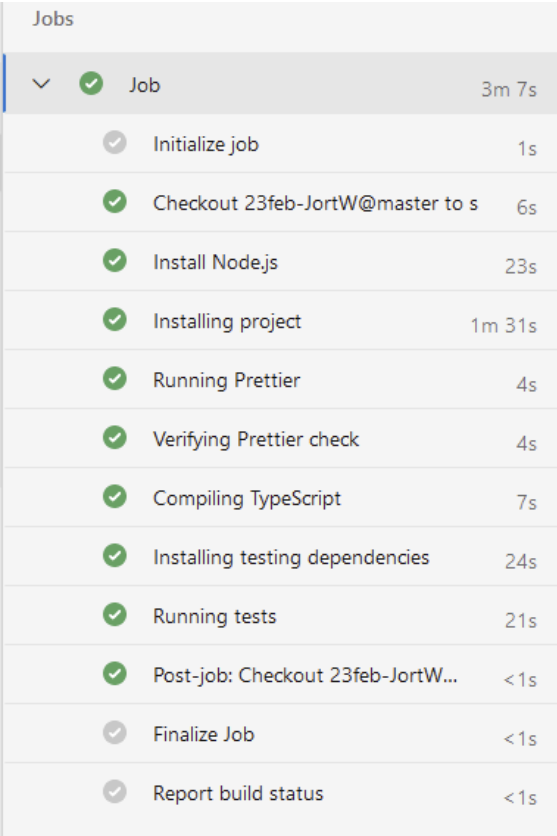
Een optie om wel C# formatting te gebruiken in Visual Studio Code: Prettier. Prettier is een formatter die gebruikt kan worden in veel IDE's, waaronder Visual Studio, Visual Studio Code, Rider, Webstorm en IntelliJ. Een ander voordeel van Prettier is dat deze een configuratie ondersteund die per project in te stellen is door middel van een configuratie file die bij het klonen van het project automatisch gedetecteerd wordt door de IDE. Dit doet Prettier door gebruik te maken van extensies geïntegreerd in IDEs. Hierdoor hoeft een ontwikkelaar lokaal niks zelf te configureren om te voldoen aan de format regels van het project (*Why Prettier?* · Prettier, z.d.). Info Support heeft ervaring met Prettier en vind het uitstekend werken bij andere projecten.

### 5.3.3. CI Pipeline

Om het volledige project en de definition of done te verifiëren wordt er gebruikgemaakt van een Continuous Integration (CI) pipeline die bij iedere push op de master branch uitgevoerd wordt. De CI pipeline is verantwoordelijk voor het checken van het formatteren door prettier uit te voeren, het compileren van het Typescript project en het runnen van alle unit testen die geschreven zijn voor dit project. Bij het runnen van de testen wordt er gekeken naar een minimum coverage van 85%.

Omdat de repository op Azure Devops staat wordt de CI pipeline gerund op basis van Azure Pipelines. De repository bevat een `azure-pipelines.yml` waarin de pipeline geconfigureerd staat. In Figuur 17 staan alle stappen van de pipeline aangegeven.

Als de pipeline faalt, wordt een mail gestuurd naar de student, zodat ook als de pipeline wat langer duurt, de student zo snel mogelijk inzicht krijgt in of de kwaliteit van de software correct is.



Jobs	
✓ Job	3m 7s
Initialize job	1s
Checkout 23feb-JortW@master to s	6s
Install Node.js	23s
Installing project	1m 31s
Running Prettier	4s
Verifying Prettier check	4s
Compiling TypeScript	7s
Installing testing dependencies	24s
Running tests	21s
Post-job: Checkout 23feb-JortW...	<1s
Finalize Job	<1s
Report build status	<1s

Figuur 17: Azure Continuous Integration Pipeline

## 5.4. Functionaliteit

De applicatie heeft verschillende functionaliteiten. De belangrijkste functies zijn: het installeren van Mystique in het te testen project, het genereren van de testen en het uitvoeren van deze testen.

### 5.4.1. Installatie

Mystique moet verbonden worden met het te testen project. Dit wordt gedaan door middel van een *installatie* commando. Dit commando wordt meegeleverd met de installatie van Mystique op het lokale systeem en kan vanaf daar aan ieder project verbonden worden.

Als het commando `mystique --install` uitgevoerd wordt in de terminal, gaat Mystique verschillende checks uitvoeren om er achter te komen of het mogelijk is om mystique te verbinden aan het project. De eerste check die uitgevoerd wordt is het zoeken van Storybook gezien Mystique niet kan werken zonder Storybook en dus verwacht dat deze aanwezig is. Is dat niet zo, dan geeft het een fout die aangeeft dat Storybook niet gevonden kan worden en daarom de installatie afgebroken wordt. Als tweede wordt gecheckt of Mystique toegang heeft tot de bestanden van de computer omdat er een configuratiebestand geplakt moet worden. Als Mystique geen toegang heeft, wordt hier ook een foutmelding voor gegeven.

Wanneer de checks voorbij zijn, kan Mystique verbonden worden aan het project. Dit wordt gedaan door een configuratiebestand te plakken in de hoofdmap van het project. Dit bestand heet: *mystique.config.json*. En is een JSON bestand waarin de Storybook url staat die gevonden is en een lege lijst met componenten die gevuld kunnen worden met vooraf gedefinieerde testen.

Als de installatie van Mystique afgerond is, wordt dit aangegeven in de terminal, waarna mogelijke commando's worden beschreven die uitgevoerd kunnen worden door de gebruiker.

### 5.4.2. Testen genereren

Het belangrijkste onderdeel van de applicatie is het genereren van de testen. Dit kan op twee verschillende manieren. De eerste en makkelijkste manier is het uitlezen van het configuratiebestand. De tweede manier vraagt het Figma design op en probeert testen af te leiden aan wat Mystique herkent.

#### Uit configuratiebestand

De opdrachtgever vindt het belangrijk dat ook zonder gebruik te maken van Figma, de applicatie nog steeds toegevoegde waarde heeft. Daarom is er besloten om ook de mogelijkheid te geven vooraf zelf testen te specificeren die meegenomen worden in de test run. In Figuur 18 is een voorbeeld opgenomen van een test die verwacht dat de achtergrondkleur van een knop rood is. De waarden van deze test in de JSON worden direct overgenomen en in een test geplaatst waardoor de gebruiker volledige controle heeft over de testen.

```
"components": [
  {
    "name": "Button",
    "tests": [
      {
        "name": "Background color should be red",
        "selector": ".background",
        "property": "background-color",
        "value": "rgb(255,0,0)"
      }
    ]
  }
]
```

Figuur 18: Voorbeeld van een test geschreven in het configuratiebestand.

De testen worden gestructureerd op basis van de componenten waar ze onder vallen. Hierdoor hoeft er per test zo min mogelijk geschreven te worden en is het duidelijk voor Mystique welke componenten er getest moeten worden. Een nadeel van het handmatig schrijven van testen is dat er geen spelfouten gemaakt mogen worden, gezien deze direct overgenomen worden in de test.

### **Uit Figma design**

Als de testen uit het configuratiebestand opgenomen zijn wordt er gekeken of er een Figma ID is meegegeven aan het uitvoer commando in de terminal. Als dit zo is, wordt er aan de gebruiker gevraagd om in te loggen bij Figma. Dit wordt gedaan om toegang te krijgen tot de designs die de gebruiker heeft gemaakt onder zijn / haar account. Bij het inloggen wordt er een webpagina geopend waar de gebruikers inloggegevens ingevoerd kunnen worden. Figma authenticceert de gebruiker en stuurt door middel van een HTTP request de OAuth gegevens van de gebruiker door naar de applicatie (Figma, z.d.-b). Vanaf dat moment kunnen deze OAuth gegevens gebruikt worden om requests te sturen naar de web API van Figma om inzicht te krijgen in de user-interface designs van de gebruiker. Door de Figma API het ID mee te geven, vragen we het goede design document op. Het document is in JSON formaat geschreven en kan in één keer omgezet worden naar een Typescript object. Nu het design bekend is kunnen er testen gegenereerd worden. Het design bevat nodes die samen een document opmaken. Iedere node kan van een ander type zijn. De meest voorkomende types zijn: Rectangle, vector en text. De applicatie bekijkt iedere node om uit te zoeken of er testen voor gegenereerd kunnen worden. Wanneer er een node is gevonden die getest kan worden, worden hiervoor testen gegenereerd die vooraf opgestelde richtlijnen volgen. Zo wordt voor een text node altijd een test gegenereerd die checkt of de font-size klopt.

### 5.4.3. Testen uitvoeren

Als alle testen gegenereerd zijn, worden ze meegegeven aan de runner die ze uit gaat voeren en het resultaat laat zien aan de gebruiker. Voordat de testen uitgevoerd worden wordt er in de CLI weergegeven hoeveel testen er uitgevoerd moeten worden. Als dit gebeurt worden de testen één voor één uitgevoerd. De webdriver checkt of het component dat getest wordt daadwerkelijk bestaat door naar de pagina te navigeren waar deze te vinden zou moeten zijn. Als deze pagina niet gevonden wordt of Storybook niet reageert, wordt er een foutmelding weergegeven.

Als de webdriver contact heeft met het component worden de waardes uit de test specificatie gebruikt om de juiste HTML tag te vinden in het component waarna de juiste eigenschap en waarde daarvan opgehaald wordt.

De waarde die opgehaald is door de webdriver wordt vergeleken met de verwachte waarde die de specificatie bevat. Als de twee waarden gelijk zijn wordt er een positief testresultaat opgeslagen. Mochten de waarden niet gelijk zijn wordt er een negatief testresultaat opgeslagen met een foutmelding die aangeeft waarom de test gefaald is. Deze resultaten worden na het uitvoeren van alle testen overhandigd aan het rapport om de resultaten via de CLI weer te geven aan de gebruiker.

### 5.4.4. Rapport



```

Found 4 tests for 2 components.
- Button
  PASS Background color should be red (1498ms) - chrome: ✓, edge: ✓, firefox: ✓
  FAIL Font family should be Impact (1195ms) - chrome: ✓, edge: ✓, firefox: X
  - firefox: Actual: undefined, is not equal to expected: Impact.
  FAIL Ruby align should be center (1140ms) - chrome: X, edge: X, firefox: ✓
  - chrome: Actual: undefined, is not equal to expected: center.
  - edge: Actual: undefined, is not equal to expected: center.
- Application-title
  PASS Font family should be Impact (1186ms) - chrome: ✓, edge: ✓, firefox: ✓
Total of 4 tests run; 2 passed; 2 failed; (5.02s)
  
```

Figuur 19: Mystique's rapport na een test run

In §4.2.6 van het onderzoek worden de software kwaliteitseigenschappen beschreven die getest kunnen worden om de kwaliteit van front-end componenten te waarborgen. Tabel 5 geeft weer hoe deze eigenschappen verwerkt kunnen worden in het testrapport om dit inzichtelijk te maken voor de gebruiker. In Figuur 19 is het gerealiseerde testrapport te zien dat een groot deel van de elementen in Tabel 5 verwerkt zijn in het testrapport.

Nr.	Informatie	Kwaliteitsfactor	Toelichting
1	Naam van de test	-	Geeft aan welke test er gerund is. Ook staat hier bij of deze geslaagd of gefaald is.
2	Totaal aantal gerunde testen	Testbaarheid	Geeft aan hoeveel testen er zijn gegenereerd en gerund zijn tijdens deze test run.
3	Aantal geslaagde testen	Testbaarheid	Geeft aan hoeveel testen er geslaagd zijn. Als alle testen geslaagd zijn, is de testrun geslaagd.
4	Aantal gefaalde testen	Testbaarheid	Geeft aan of er testen zijn die niet slagen. Als dit zo is, dan is de testrun gefaald.
5	Waarom een test gefaald is	Correctheid	Als een test gefaald is, wordt er aangegeven waarom deze test gefaald is, zodat deze zo snel mogelijk opgelost kan worden.
6	Browser waarin de test gerund is.	Integriteit	Welke browser is gebruikt het component te testen, zodat de gebruiker weet of een bepaalde browser het verkeerd rendert.
7	Run duur van één test	Testbaarheid	In milliseconden. Geeft aan hoe lang een test heeft geduurd.
8	Totale duur van de testrun	Testbaarheid	In seconden. Geeft aan hoelang een testrun heeft geduurd.

Tabel 9: Elementen verwerkt in het testrapport

In Tabel 9 staan de elementen die beschreven die in het rapport verwerkt zijn. In Figuur 19 staan de cijfers die corresponderen met die elementen. De structuurvergelijking is niet geïmplementeerd omdat het geen prioriteit is voor de opdrachtgever en daardoor meer tijd besteed kan worden aan andere kwaliteitsfactoren die meer waarde hebben voor de opdrachtgever.



## 5.5. Advies

Het beroepsproduct is niet alleen een manier om de onderzoeksresultaten te verifiëren. Het is ook een bruikbaar product dat Info Support in gebruik kan nemen en door kan ontwikkelen. Wanneer de student klaar is met het afstuderen zijn er een aantal zaken die Info Support kan doen met betrekking tot het beroepsproduct.

### 5.5.1. Gebruik van het beroepsproduct

Heel hoofdstuk 5 beschrijft wat het beroepsproduct kan en waar het goed in is. Voor Info Support is er naast de code van het product ook een handleiding opgeleverd die aangeeft hoe Mystique geïnstalleerd kan worden binnen een front-end project en hoe Mystique gebruikt kan worden om testen te schrijven, genereren en uit te voeren.

Mystique kan gebruikt worden in ieder front-end project dat Storybook ondersteunt en biedt toegevoegde waarde voor deze projecten. Het kan gebruikt worden naast een traditionele unit-testing framework die door de meeste projecten al gebruikt wordt. De student heeft al van meerdere collega's gehoord binnen Info Support dat Mystique iets is wat hen enorm kan helpen en zij enorm geïnteresseerd zijn om ermee aan de slag te gaan.

### 5.5.2. Verdere ontwikkeling

Mystique is in een gebruiksklare staat maar het blijft een proof of concept. Tijdens het interne gebruik zal er aan doorontwikkeld moeten blijven worden. De student ziet graag een open source project dat ondersteund wordt door Info Support, net als Stryker. Verder kunnen de aanbevelingen die gegeven worden in §7.3 uitgevoerd worden om tot een beter product te komen dat gebruikt kan worden door veel meer projecten binnen en buiten Info Support.

De student ziet ook graag dat er meer stageopdrachten gecreëerd worden voor dit onderwerp omdat er nog veel gedaan kan worden in een ruimte waar nog niet veel onderzoek naar is gedaan.

## 6. Conclusie & aanbevelingen

---

De opdracht zoals opgesteld door de student en opdrachtgever luidt: “ Onderzoek hoe een user interface test ontwikkeld kan worden voor een front-end component die eigenschappen van een user-interface design vergelijkt met de geïmplementeerde code, in een nieuwe duurzame javascript omgeving, zodat de samenhang tussen het user-interface design en het component gewaarborgd kan worden ”. Uit het onderzoek is gebleken dat door te kijken naar de software kwaliteitseigenschappen: correctheid, integriteit, testbaarheid en structuur een frontend component getest kan worden.

Testen om deze eigenschappen te waarborgen kunnen niet geautomatiseerd geschreven worden door gebruik te maken van gevestigde test runners binnen het Javascript landschap. Hiervoor is een nieuwe test runner geschreven die gemaakt is om specifiek deze software kwaliteitseigenschappen te testen en om te voldoen aan de wensen van de opdrachtgever. In het onderzoek wordt beschreven dat de gevestigde test runners parallellen bevatten in hun software architectuur. Deze parallellen zijn: (I) de test specificatie, (II) het test assertion framework, (III) de test runner, en (IV) het testrapport. Deze zijn gebruikt om de architectuur voor de zelfgeschreven testrunner te definiëren.

De test runner maakt gebruik van een webdriver om via Storybook gerenderde componenten te benaderen, zodat variabelen uit deze componenten gebruikt kunnen worden om te testen. De webdriver is, uit onderzoek gebleken, de beste optie om een webbrowser te manipuleren en Storybook de beste manier om front-end componenten te renderen in een browser, gezien er geen tijd en expertise is om een eigen renderer te maken.

De opdracht beschrijft dat er geautomatiseerd testen gemaakt moeten worden op basis van een user-interface design. Dit is gerealiseerd door gebruik te maken van de openbare API die Figma beschikbaar stelt. Figma is na onderzoek de beste en enige optie gebleken die de functionaliteit ondersteunt die nodig is voor dit project.

De geschreven test runner werkt out of the box met drie van de vijf onderzochte javascript frameworks die onderzocht zijn in deelvraag 1. Daarom is gekozen voor een Node.JS CLI applicatie (*Command-line API | Node.js v20.3.1 Documentation*, n.d.) die als NPM package geïnstalleerd kan worden in alle onderzochte frameworks. Voor deze frameworks zijn testgevallen beschreven die geautomatiseerd gegenereerd worden voor de componenten die de gebruiker wil testen. Het doel van de opdracht was om te onderzoeken of het mogelijk was om deze testen te genereren. Daarom is er nog geen tijd geweest om veel test cases te beschrijven.

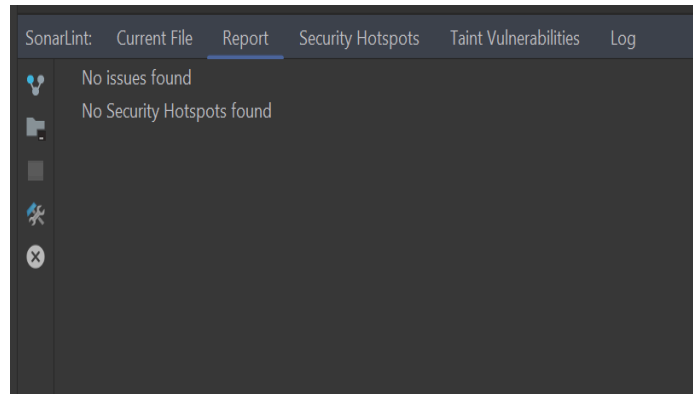
Door de test runner te gebruiken op een demo project met een user-interface design uit Figma is te zien dat testen gegenereerd worden en de software kwaliteitseigenschappen weergegeven worden in het rapport dat aan het einde van de test run teruggegeven wordt. Hierdoor kan de consistentie tussen user-interface design en geïmplementeerde code gewaarborgd worden.

## 6.1. Kwaliteit

In het plan van aanpak zijn een aantal kwaliteitseisen opgesteld door de student en opdrachtgever die de kwaliteit waarborgen van de code en het product. De volgende eisen zijn opgesteld: (I) leesbare code, (II) testen, (III) uitwisselbaarheid en (IV) logging.

### 6.1.1. Leesbare code

Gezien het project gebruikt gaat worden door andere ontwikkelaars en gebruikers, is het belangrijk dat de code leesbaar en correct is. Hiervoor is gebruikgemaakt van statische codeanalyse en formatting. Deze worden ook uitgevoerd in de CI pipeline die is opgesteld. Hierdoor kan deze kwaliteitseis gewaarborgd worden.



Figuur 20: Statische code analyse rapport van SonarLint

### 6.1.2. Testen

De gehele applicatie wordt getest door unit testen en acceptatie. De eis zoals beschreven in het plan van aanpak verwacht een minimale test coverage van 80% en een mutation coverage van 60%. De test coverage eis is gehaald maar de mutation coverage is niet getest gezien dit niet werkend is gekregen omdat de omgeving waarin gewerkt wordt een niet standaard javascript omgeving is en de mutation coverage tools (Stryker Mutator, z.d.) hier niet mee kunnen werken. De unit testen worden uitgevoerd in de CI pipeline waardoor deze altijd gewaarborgd worden.

### 6.1.3. Uitwisselbaarheid

De eis zoals beschreven in het plan van aanpak beschrijft uitwisselbaarheid als code dat met zo min mogelijk aanpassingen gebruikt kan worden om andere componenten te testen. In §5.1.3 wordt uitgelegd hoe een testrunner gebouwd kan worden die verschillende typen componenten kan testen zonder hiervoor verschillende code te schrijven. Het enige wat niet herbruikbaar is, is de code om de testen te genereren voor verschillende nodes uit het figma bestand. Dit staat los van de componenten en is geen duplicaat vergeleken met andere code in de repository.

### 6.1.4. Logging

Op dit moment logt de applicatie goed bij welke test de test run faalt en geeft ook aan welke waarde verwacht wordt en welke waarde er verkregen is. Hierdoor krijgt de gebruiker een duidelijk inzicht in waar de test gefaald is. Gezien er geen code geschreven wordt voor de test is het onmogelijk om aan te geven op welke lijn in de code de test faalt. De opdrachtgever vindt dat deze kwaliteitseis correct behaald is.

## 7. Discussie

---

### 7.1. Validiteit

Voor deze opdracht is gekozen om een literatuuronderzoek uit te voeren. Het doel was om te onderzoeken of het mogelijk is om een geautomatiseerde test te schrijven die een front-end component op basis van een user interface design.

#### 7.1.1. Onderzoek

De eerste deelvraag gaat in op de aansluiting van een front-end framework voor Info Support. Deze deelvraag is niet direct van toepassing op de opdracht maar geeft inzicht in het landschap waarin het beroepsproduct gaat functioneren. Er is gebruikgemaakt van meerdere onderbouwde eigenschappen die belangrijk zijn voor Info Support, die als eerste het product in gebruik neemt, en de gehele front-end community.

Deelvraag 2 gaat dieper in op hoe een front-end test geschreven kan worden en wat er dan getest kan worden. Door te kijken naar reeds gevestigde en veelgebruikte test runners in §4.2.2, zijn er parallellen getrokken die inzicht geven in een mogelijke architectuur die geïmplementeerd kan worden in het beroepsproduct. De functionaliteit die de test runners bedragen en welk onderdeel daar verantwoordelijk voor is, is daarbij meegenomen. Deze architectuur is daarna ook daadwerkelijk uitgewerkt en wordt nu gebruikt om het beroepsproduct te realiseren. In deelvraag 1 is onderzocht dat

Ook is er in deelvraag 2 onderzocht, op basis van literatuuronderzoek, hoe een software test werkt en waarom deze inzicht geeft in bepaalde eigenschappen van software kwaliteit. Een groot deel van deze eisen zijn geïmplementeerd in het beroepsproduct om testen te genereren die gebruikt worden om de software kwaliteit van de componenten te testen. Het is nog moeilijk te zeggen of deze eigenschappen met een 100% zekerheid getest worden door de gegenereerde testen. Hiervoor zouden er meer verschillende testcases geïmplementeerd moeten worden. Dit was namelijk niet de kern van de opdracht en er was geen tijd voor over.

Het huidige onderzoek is een aanvulling op de bestaande literatuur over front-end testing omdat er nog weinig bekend is over het verbinden van geïmplementeerde code en software componenten. Op basis van dit onderzoek zou er gekeken kunnen worden welke testcases er verder gedefinieerd moeten worden om een adequate testdekking te krijgen. Zodat de teststrategie goed genoeg is om met zekerheid te kunnen bepalen of het component en het user-interface design consistent zijn met elkaar.

## 7.1.2. Beroepsproduct

De student heeft een beroepsproduct gerealiseerd dat bruikbaar is voor Info Support. Info Support heeft de student gevraagd om een manier waarop een user-interface design vergeleken kan worden met een geïmplementeerd component en het gerealiseerde beroepsproduct is een losse node.js applicatie die dit succesvol kan uitvoeren.

Een limitatie van het product is de testdekking. Het doel van het project was om uit te zoeken of het mogelijk was om minimaal één test te genereren. In de praktijk zijn er vijf verschillende testen die gegenereerd kunnen worden:

- Font-size checken voor tekst.
- Font-family checken voor tekst.
- Kleur voor tekst
- Achtergrond kleur voor een rectangle (div element)
- Border van een rectangle (div element)

Om echt een geïmplementeerd component te kunnen valideren zullen er veel meer testen geschreven moeten worden zodat meer eigenschappen gevalideerd kunnen worden. Om het product toch bruikbaar te maken voor Info Support hebben de opdrachtgever en de student bedacht om zelf geschreven specificaties toe te voegen aan het configuratiebestand. Als de gebruiker wil, kan deze zelf een test definiëren die nog niet automatisch gegenereerd kan worden. Hierdoor kan Info Support het product wel, gelimiteerd, gebruiken binnen projecten.

## 7.2. Procesgang

De procesgang is over het algemeen goed verlopen. De student heeft nooit het gevoel gehad dat hij te weinig tijd zou hebben gehad om de opdracht te voltooien binnen de gestelde tijd om af te studeren. Wel zijn er een aantal uitdagingen geweest die de student heeft moeten aangaan.

### 7.2.1. Opdrachtdefinitie

De originele opdracht, zoals Info Support had opgesteld, voldeed niet aan de eisen van de Hogeschool Utrecht. Met name de diepgang en mogelijkheid om theoretisch onderzoek uit te voeren was niet voldoende. De student is gelijk om de tafel gaan zitten met de Hogeschool Utrecht en Info Support om de kern van de opdracht zo op te stellen dat alle partijen tevreden waren. Hierdoor heeft de student een goede opdrachtdefinitie opgeschreven die de diepgang heeft om een correct en interessant theoretisch onderzoek mee uit te voeren.

### 7.2.2. Planning

In het plan van aanpak is een planning gemaakt waarin een initialisatie-, onderzoek-, realisatie- en scriptie fase zijn opgesteld. Deze planning is te vinden in Bijlage [num]. De student heeft een herhalende afspraak (2x per week een half uur) ingepland om de planning bij te houden, de voortgang met elkaar te delen en te discussiëren.

De planning is goed bijgehouden tot de overgang tussen de onderzoeksfase en de realisatiefase, waar bleek dat het onderzoek uitgebreider was dan van tevoren gedacht. De student heeft met de opdrachtgever en procesbegeleider overlegt en een herziene planning ingediend die de noodzaak voor extra onderzoek in acht nam. Deze herziene planning is te vinden in Bijlage [num]. De herziene planning is genoeg geweest om het resterende gedeelte van het project uit te voeren.

### 7.2.3. Onderzoek

De opdracht, die Info Support de student gevraagd heeft uit te voeren, is innovatief en nieuw. Hierdoor heeft de student veel tijd besteed aan het onderzoek dat complex was. De originele deelvragen zoals beschreven in het plan van aanpak tijdens de initialisatiefase waren bij nader inzien niet geschikt voor het onderzoek. Daarom is er tijdens de onderzoeksfase besloten, na overleg met de opdrachtgever en docent, om een deelvraag te schrappen en te verwerken in een andere deelvraag. Hierdoor loopt het verhaal dat verteld wordt in het onderzoek beter en is het proces overzichtelijker.

## 7.3. Aanbevelingen

Het onderzoek dat is uitgevoerd door de student is bedoeld om te bewijzen of een oplossing voor het gestelde probleem mogelijk is. De student heeft dit bewezen en daarvoor een proof of concept gerealiseerd. Dit betekent ook dat het proof of concept nog aangepast en verbeterd kan worden tot een verfijnd product. De volgende aanbevelingen kunnen verwerkt worden tot één of meerdere stageopdrachten. De volgende aanbevelingen zijn hier een

### 7.3.1. Hoger test gehalte

Zoals aangegeven in §7.1.2 is het doel van het beroepsproduct om te onderzoeken of er testen gegenereerd kunnen worden voor front-end componenten. Dat is gelukt en om het te bewijzen zijn er een aantal voorbeeld testen gedefinieerd die gebruikt kunnen worden in het beroepsproduct. Om echt componenten te kunnen valideren zullen er veel meer soorten testen gedefinieerd moeten worden. Hiervoor kan gekeken worden naar de documentatie van Figma om daaruit zo veel mogelijk eigenschappen te trekken waarmee een test gemaakt zou kunnen worden.

### 7.3.2. Custom rendering

Er is een keuze gemaakt om Storybook te gebruiken voor het renderen van componenten. Een uitleg waarom hiervoor gekozen is, is te vinden in §4.2.5. Het komt erop neer dat er geen tijd en expertise in huis is om een complex vraagstuk als rendering te behandelen in het tijdsbestek van deze opdracht.

Het toevoegen van een eigen gebouwde rendering framework zet de deuren open voor de ondersteuning van andere frameworks en een geoptimaliseerde render pipeline die test snelheid kan verbeteren en de afhankelijkheid van externe tools verlaagd.

### 7.3.3. Realtime reporting

Het uitvoeren van een test kan lang duren en op dit moment krijgt de gebruiker pas een resultaat als alle testen zijn uitgevoerd. Het kan daarom lang duren om een resultaat te krijgen die eerder al bekend was. Een mogelijke oplossing hiervoor kan realtime reporting zijn. Dit houdt in dat het rapport automatisch update iedere keer als er nieuwe informatie beschikbaar is. Hierdoor heeft de gebruiker meer en sneller inzicht.

### 7.3.4. Performance verbeteren

Op dit moment is de applicatie goed bruikbaar voor een klein project omdat deze projecten meestal minder complexe en aantallen componenten bevatten. Dit komt door de tijd dat het kost om een test uit te voeren. Het kost nu ~500ms om een test uit te voeren op drie browsers, waardoor een test suite met honderden componenten lang kan duren voordat er resultaten zijn.

## 8. Literatuur

11 best JavaScript Assertion libraries in 2023 | kandi. (2023, March 13).

<https://kandi.openweaver.com/collections/assertion/javascript-assertion>

A short history of the Web. (2023, March 3). CERN.

<https://home.cern/science/computing/birth-web/short-history-web>

Abuhakmeh, K. (2021, April 22). Add Svelte To ASP.NET Core Projects. Khalid Abuhakmeh's Blog.

<https://khalidabuhakmeh.com/add-svelte-to-aspnet-core-projects>

Adobe XD APIs for developers and scripters. (n.d.). <https://developer.adobe.com/xd/>

Agile Scrum Group. (2022). Wat is Kanban? Een volledige uitleg van de Kanban Methode + template. Agile Scrum Group.

<https://agilescrumgroup.nl/wat-is-kanban-methode/>

Angular. (n.d.). <https://angular.io/>

Architecture · Jest. (2023, March 6).

<https://jestjs.io/docs/architecture>

ASP.NET | Open-source web framework for .NET. (n.d.). Microsoft.

<https://dotnet.microsoft.com/en-us/apps/aspnet>

Bash, M. (2023, March 8). What Is the Measuring Unit in Figma?

WebsiteBuilderInsider.com.

<https://www.websitebuilderinsider.com/what-is-the-measuring-unit-in-figma/>

Basic Editing in Visual Studio Code. (2021, November 3).

[https://code.visualstudio.com/docs/editor/codebasics#\\_formatting](https://code.visualstudio.com/docs/editor/codebasics#_formatting)

Cahill, C., & May, T. (2023). The best UI design tools in 2023. Creative Bloq.

<https://www.creativebloq.com/how-to/20-best-ui-design-tools>

Chcomley. (2023, February 24). Define features and epics, organize backlog items - Azure Boards. Microsoft Learn.

<https://learn.microsoft.com/en-us/azure/devops/boards/backlogs/define-features-epics?view=azure-devops&tabs=agile-process>

Command-line API | Node.js v20.3.1 Documentation. (n.d.).

<https://nodejs.org/api/cli.html>

CSS values and units - Learn web development | MDN. (2023, February 23).

[https://developer.mozilla.org/en-US/docs/Learn/CSS/Building\\_blocks/Values\\_and\\_units](https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Values_and_units)

Cypress Component Testing | Cypress Documentation. (n.d.).

<https://docs.cypress.io/guides/component-testing/overview>

Decan, A., Mens, T., & Constantinou, E. (2018). On the Evolution of Technical Lag in the npm Package Dependency Network.

<https://doi.org/10.1109/icsme.2018.00050>

Durand, J., & Curran, P. (2012, October 15).

Test assertions model version 1.0. Oasis Open. Retrieved May 2, 2023, from <http://docs.oasis-open.org/tag/model/v1.0/os/testassertionsmodel-1.0-os.pdf>

ECMAScript Modules · Jest. (2023, March 12).

<https://jestjs.io/docs/ecmascript-modules>

Fable-Compiler. (n.d.). GitHub - fable-compiler/Fable.Solid: Fable bindings for SolidJS. GitHub.

<https://github.com/fable-compiler/Fable.Solid>



Fable.Core 4.0.0. (n.d.).

<https://www.nuget.org/packages/Fable.Core>

Facts and Figures 2022. (n.d.).

<https://www.itu.int/itu-d/reports/statistics/facts-figures-2022/#:~:text=Share%20ths%3A,over%20own%20a%20mobile%20phone>

Fay, O. (2023, March 8). User interface testing: Why it's important and how to do it right. Poll the People. Retrieved March 10, 2023, from <https://pollthepeople.app/user-interface-testing/>

Feature support for frameworks. (n.d.). <https://storybook.js.org/docs/react/api/frameworks-feature-support>

Ferreira, C. A. (2018, April 25). How to use AngularJS in ASP.NET MVC and Entity Framework. Medium. <https://medium.com/@ciceroaferreira/how-to-use-angularjs-in-asp-net-mvc-and-entity-framework-319ef2dd01c3>

Figma. (n.d.-a). Figma. <https://www.figma.com/developers/api>

Figma. (n.d.-b). Figma. <https://www.figma.com/developers/api#oauth2>

Framer: Developers. (n.d.). <https://www.framer.com/developers/>

Geek, N. (2022). How to write a good Test Summary Report? | BrowserStack. BrowserStack. <https://www.browserstack.com/guide/how-to-write-test-summary-report>

getCSSProperty | WebdriverIO. (n.d.). <https://webdriver.io/docs/api/element/getCSSProperty/>

Godbolt, M. (2016). Frontend Architecture for Design Systems: A Modern Blueprint for

Scalable and Sustainable Websites. "O'Reilly Media, Inc."

Install addons. (n.d.). <https://storybook.js.org/docs/react/addons/install-addons>

Integrations | Storybook: Frontend workshop for UI development. (n.d.). <https://storybook.js.org/integrations>

Introduction to the DOM - Web APIs | MDN. (2023, April 15). [https://developer.mozilla.org/en-US/docs/Web/API/Document\\_Object\\_Model/Introduction](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction)

Invision. (2023). Integrations. Invision App. Retrieved May 3, 2023, from <https://www.invisionapp.com/integrations>

JavaScript modules - JavaScript | MDN. (2023, June 18). <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules>

JavaScript unit testing frameworks in 2022: A comparison. (2022, October 6). Raygun Blog. <https://raygun.com/blog/javascript-unit-testing-frameworks/>

Jest. (n.d.). <https://jestjs.io/>

Karma - Spectacular Test Runner for Javascript. (n.d.). <https://karma-runner.github.io/6.4/index.html>

Keller, M., & Nussbaumer, M. (2009). Cascading style sheets. <https://doi.org/10.1145/1526709.1526907>

Khorikov, V. (2020). Unit Testing Principles, Practices, and Patterns: Effective testing styles, patterns, and reliable automation for unit testing, mocking, and integration testing with examples in C#. Manning.

Knockout : Home. (n.d.). <https://knockoutjs.com/>

Li, K., Ding, Y., Shen, D., Li, Q. X., & Zhen, Z. (2020). The Design and Research of Front-End Framework for Microservice Environment. In 2020 International Conference on Computer Information and Big Data Applications (CIBDA).  
<https://doi.org/10.1109/cibda50819.2020.00036>

Maze. (2023, April 26). 19 Best UI/UX Design Tools for 2023 (Ultimate List) | Maze. Maze.  
<https://maze.co/collections/ux-ui-design/tools/>

Meet the Team | Vue.js. (n.d.).  
<https://vuejs.org/about/team.html>

Mocha - the fun, simple, flexible JavaScript test framework. (2023, April 12).  
<https://mochajs.org/>

Mulder, P. (2022, December 2). MoSCoW Methode, een praktische uitleg met voorbeelden - Toolshero. Toolshero.  
<https://www.toolshero.nl/project-management/moscow-methode/>

Nicojs. (n.d.). GitHub - nicojs/typed-inject: Type safe dependency injection for TypeScript. GitHub.  
<https://github.com/nicojs/typed-inject>

node package manager [npm]. (n.d.-a). npm: @angular/core. Npm.  
<https://www.npmjs.com/package/@angular/core?activeTab=versions>

node package manager [npm]. (n.d.-b). npm: svelte. Npm.  
<https://www.npmjs.com/package/svelte?activeTab=versions>

node package manager [npm]. (n.d.-c). npm: vue. Npm.  
<https://www.npmjs.com/package/vue?activeTab=versions>

node package manager [npm]. (n.d.-d). React npm. Npm. Retrieved April 14, 2023, from

<https://www.npmjs.com/package/react?activeTab=versions>

Node Types | Plugin API. (n.d.).  
<https://www.figma.com/plugin-docs/api/nodes/>

npm About. (n.d.).  
<https://www.npmjs.com/about>

Our Audience | Stack Overflow Advertising - Stack Overflow. (n.d.-a).  
<https://stackoverflow.co/advertising/audience/>

Our Audience | Stack Overflow Advertising - Stack Overflow. (n.d.-b).  
<https://stackoverflow.co/advertising/audience/>

Pan, J. (1999). Software Testing [Pdf]. Carnegie Mellon University.  
<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=28abbfdcd695f6ffc18c5041f8208dcfc8810aaf>

Pekarsky, M. (2020, February 6). Does your web app need a front-end framework? - Stack Overflow Blog. Stack Overflow Blog.  
<https://stackoverflow.blog/2020/02/03/is-it-time-for-a-front-end-framework/>

React integration for ASP.NET MVC | ReactJS.NET. (n.d.). <https://reactjs.net/>

Rios, G. R. (2021, February 18). Using Vuejs with ASP.NET Core MVC. Genesisrrios.  
<https://www.genesisrrios.com/en/blog/using-asp-net-core-with-vue-js/>

Selectors | WebdriverIO. (n.d.).  
<https://webdriver.io/docs/selectors/>

Singh, T. (n.d.). Top 8 JavaScript Testing Frameworks: Everything You Need to Know. <https://www.trio.dev/blog/top-8-javascript-testing-frameworks>

Sinon.JS - Standalone test fakes, spies, stubs and mocks for JavaScript. Works with any

unit testing framework. (n.d.).

<https://sinonjs.org/>

Smith, J., Development, I. a. O. S. a. T. F., & Engineering, I. a. O. S. a. T. F. D. T. C. O. S. (2007). Proceedings of the 11th IASTED International Conference on Software Engineering and Applications: November 19 - 21, 2007, Cambridge, Massachusetts, USA.

Snapshot Testing · Jest. (2023, March 6).

<https://jestjs.io/docs/snapshot-testing>

Stack Overflow. (n.d.).

<https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangular%2Csvelte>

Stackoverflow. (2022). 2022 developer survey. Retrieved March 15, 2023, from <https://survey.stackoverflow.co/2022/#section-most-popular-technologies-web-frameworks-and-technologies>

Stevens, E. (2023, March 2). The 10 Best UI Design Tools to Try in 2023. UX Design Institute.

<https://www.uxdesigninstitute.com/blog/user-interface-ui-design-tools/>

Story rendering. (n.d.).

<https://storybook.js.org/docs/react/configure/story-rendering>

Storybook: Frontend workshop for UI development. (n.d.).

<https://storybook.js.org/>

Stryker Mutator. (n.d.).

<https://stryker-mutator.io/>

Świstak, T. (2021, January 25). The rise of the Svelte JavaScript framework. ValueLogic | Blog.

<https://valuelogic.one/blog/the-rise-of-the-svelte-javascript-framework/>

Therox, O. (n.d.). Snapshot Testing · objc.io.

<https://www.objc.io/issues/15-testing/snapshot-testing/>

Trinh, H. (2020). A practical approach to JavaScript testing [Pdf]. Metropolia.

[https://www.theseus.fi/bitstream/handle/10024/349418/huytrinh\\_thesis.pdf?sequence=2&isAllowed=y](https://www.theseus.fi/bitstream/handle/10024/349418/huytrinh_thesis.pdf?sequence=2&isAllowed=y)

Types of addons. (n.d.).

<https://storybook.js.org/docs/react/addons/addon-types#panels>

Unadkat, J. (2022, May 30). Top Javascript Testing Frameworks | BrowserStack. BrowserStack.

<https://www.browserstack.com/guide/top-javascript-testing-frameworks>

Vue.js - The Progressive JavaScript Framework | Vue.js. (n.d.). <https://vuejs.org/>

WebDriver. (2023, March 21).

<https://www.w3.org/TR/webdriver/>

West, D. D. (n.d.). Agile scrumrollen | Atlassian. Atlassian.

<https://www.atlassian.com/nl/agile/scrum/roles>

What is Software Testing and How Does it Work? | IBM. (n.d.).

<https://www.ibm.com/topics/software-testing>

Why Prettier? · Prettier. (n.d.).

<https://prettier.io/docs/en/why-prettier.html>

You may not need a bundler for your NPM library. (2022, May 27).

<https://cmdcolin.github.io/posts/2022-05-27-youmaynotneedabundler>

Zhu, H., Wei, L., Wen, M., Liu, Y., Cheung, S., Sheng, Q., & Zhou, C. (2020). MockSniffer.

<https://doi.org/10.1145/3324884.3416539>