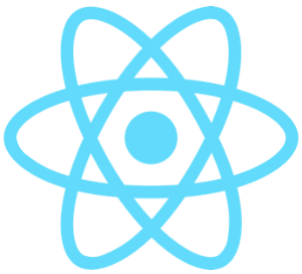
Codedelen tussen React en SpringBoot met Kotlin

Kevin Scholten

Verslag Afstudeerstage Topicus Healthcare - Screening

Kotlin



React

Spring Boot

Naam	Kevin Scholten
Student e-mail	446792@student.saxion.nl
Zakelijke e-mail	Kevin.Scholten@topicus.nl
Bedrijfsbegeleider	Robert Spee
Stagebegeleider	Dick Heijink
Organisatie	Topicus Healthcare B.V.
Locatie	Deventer
Versie	2.0
Datum	03-04-2022

Voorwoord

U leest het afstudeerverslag genaamd ‘het toepassen van Kotlin in modules van het cliëntportaal om uniformiteit te creëren’. Dit project is uitgevoerd binnen de applicatie ‘Cliëntportaal’ van het product ‘ScreenIT’ dat de Nederlandse bevolkingsonderzoeken naar 3 soorten kanker dient. De opdracht speelt zich af binnen de divisie Healthcare van het bedrijf Topicus. Dit verslag is geschreven naar aanleiding van het afstuderen van Kevin Scholten aan de opleiding HBO-ICT aan Saxion Hogeschool te Enschede. Binnen deze opleiding is de leerrichting Software Engineering gevolgd. De periode van het afstudeerproject heeft gelopen van *15 november 2021 tot 15 april 2022* (5 kalendermaanden).

Vanuit Topicus is mij begeleiding geboden door Robert Spee. Vanuit Saxion Hogeschool is mij begeleiding geboden door Dick Heijink. Bij deze wil ik beiden bedanken voor de begeleiding gedurende het project.

Veel leesplezier gewenst!

Met vriendelijke groet,
Kevin Scholten

Inhoudsopgave

Samenvatting

Het doel van dit project en onderzoek is het onderzoeken van de mogelijkheden die KotlinJS te bieden heeft met betrekking tot het delen van code met een KotlinJVM module. Is het mogelijk om bepaalde delen niet meer dubbel te hebben in beide modules maar deze op een gezamenlijke locatie te hebben? Kan het complete systeem functioneren met 1 programmeertaal aan de front- en back-end?

Kotlin is oorspronkelijk gemaakt als high-end programmeertaal die, net als Java, word gecompileerd naar Java bytecode (JVM). Deze taal wordt ook wel KotlinJVM genoemd. Later is Kotlin door Jetbrains ook ingezet om te compileren naar JavaScript (als KotlinJS). Topicus wil Kotlin inzetten in verschillende informatiesystemen om een aantal voordelen te creëren. Deze voordelen houden in: Het dichter bij elkaar brengen van front- en back-end door dezelfde programmeertaal (Kotlin) in te zetten binnen de frameworks (React en Spring Boot). Hierdoor ontstaat er voor de ontwikkelaar het gemak van makkelijk kunnen springen van front- naar back-end, en het kunnen wijzigen van code op 1 plek voor beide modules doordat code gedeeld is.

Om de onderzoeksvragen te beantwoorden zijn er verschillende onderzoeksmethoden gebruikt. Zo is er veel kennis opgedaan door middel van het onderzoeken van de werking van Kotlin met verschillende frameworks en is er onderzoek gedaan naar de sociale aspecten die komen kijken door het veranderen van een programmeertaal. Hoe denken bijvoorbeeld de ontwikkelaars die ermee verder moeten werken hierover?

Om erachter te komen hoe Kotlin het best op het cliëntportaal kan worden toegepast, moet er eerst een goed beeld zijn over hoe het cliëntportaal in elkaar zit en hoe dit in elkaar steekt ten opzichte van de andere applicaties binnen het product. Hiervoor zijn schetsen gemaakt van huidige- en gewenste situaties van het cliëntportaal en het product als geheel.

Kotlin in combinatie met front-end frameworks is vrij nieuw. Qua literatuur is er wel wat beschikbaar, maar in de praktijk wordt het nog niet veel gebruikt. Hierdoor is het lastig om informatie voor een specifiek probleem te vinden. Uit een enquête onder de ontwikkelaars is gebleken dat zij de overstap naar Kotlin wel zien zitten, mits er niet te veel tijd omgaat in het migreren hierheen en de compatibiliteit met andere frameworks goed verloopt.

De conclusie van de onderzoeken is dat codedeling tussen een front- en back-end mogelijk is om duplicate code te elimineren en het gemak van het gebruik van dezelfde programmeertaal toe te passen. In Proof of Concept 3 is aangetoond dat het mogelijk is om code te delen tussen een KotlinJS front-end (React) en een KotlinJVM back-end (Spring Boot).

Het algemene advies luidt: Op dit moment lijkt het nog geen goed idee om een front-end in KotlinJS te schrijven, door het klein aantal *kotlin wrappers* die op dit moment beschikbaar zijn. Op internet is ook nog niet veel ondersteuning te vinden voor KotlinJS in combinatie met 3rd party frameworks. Voor elk framework waar nog geen KotlinJS *wrapper* voor is, zou deze zelf moeten worden geschreven. Voor TypeScript of JavaScript is dit niet nodig, wat dus enorm veel tijd scheelt. De conclusie: het is wellicht te vroeg om Kotlin in te zetten binnen een front-end framework.

1. Inleiding

Hoe zou het zijn om aan de front-end en aan de back-end dezelfde moderne programmeertaal te gebruiken? Welke voordelen zijn hieraan verbonden en zijn er nadelen? Wegen de voordelen op tegen deze nadelen? Op welke manier wordt er code gedeeld tussen de front-end en back-end? Op deze vragen hoop ik in mijn project antwoord te geven door middel van het onderzoeksrapport en het Proof of Concept. Het project speelt zich af bij het bedrijf Topicus Healthcare.

Mijn interesse voor deze opdracht werd gewekt doordat ik op een eerdere stage ook al heb gewerkt met Kotlin en dit als een fijne taal beschouw. Tijdens deze periode heb ik de voordelen van Kotlin ten opzichte van Java mogen ontdekken in combinatie met Android. Ik ben benieuwd welke toepassingen er kunnen worden geleverd met deze moderne programmeertaal en of het op dit moment een groot voordeel biedt voor Topicus. Welke voordelen kan Kotlin bieden voor de systemen van de bevolkingsonderzoeken naar kanker? Kunnen de React front-end en de Spring Boot back-end beiden Kotlin ondersteunen en kan dit codedeling mogelijk maken?

Samen met mijn bedrijfsbegeleider van Topicus, Robert Spee, heb ik de opdracht specifieker gevormd tot wat het nu is en is de omvang van de opdracht vastgesteld in het plan van aanpak. Mijn doel is de lezer te adviseren waarom wel, of juist niet Kotlin te gebruiken als programmeertaal in het Cliëntportaal-systeem en andere front-end applicaties.

Eerst wordt de organisatie van het bedrijf besproken. In hoofdstuk 3 de opdracht, wat houdt de opdracht precies in en in welke omgeving vindt deze plaats? In hoofdstuk 4 wordt de aanpak van het project besproken. In hoofdstuk 5 wordt er ingegaan op bepaalde onderzoeksmethodes en de opzet van het onderzoek, en in hoofdstuk 6 worden de resultaten besproken. Uit deze resultaten wordt vervolgens een conclusie gevormd waaruit weer een advies luidt.

1.1 Uitleg Kotlin

Kotlin is een programmeertaal gemaakt door het bedrijf Jetbrains die in 2016 officieel uit is gekomen. Het is een high-end taal die als vervanger voor Java zou gaan dienen. Echter is Kotlin uitgegroeid tot veel meer dan dat. Kotlin kent verschillende soorten met verschillende compilers. De twee soorten die tijdens dit project van belang zijn, zijn **KotlinJVM** en **KotlinJS.** De eerste, KotlinJVM is de eerste “originele” Kotlin. Deze geschreven code compileert naar Java Byte Code, net zoals Java. Deze bytecode draait op een Java Virtual Machine (JVM). De geschreven code van de tweede, KotlinJS, compileert naar JavaScript. De JavaScript draait in de browser.

In de React front-end wordt KotlinJS ingezet en in de Spring Boot back-end KotlinJVM. Kotlin vervangt hier dus niet het framework, maar wordt ingezet in combinatie met de frameworks.

1.2. Relevante afkortingen en terminologie

In het Plan van Aanpak en het afstudeerdossier zullen ongetwijfeld bepaalde afkortingen en terminologie worden gebruikt. In onderstaande tabel is dit beschreven.

Afkorting/term	**Betekenis**
PvA	Plan van Aanpak
PoC	Proof of Concept
BMHK	Baarmoederhalskanker
BK	Borstkanker
DK	Darmkanker
KotlinJVM	Kotlin gecompiled naar Java Virtual Machine
KotlinJS	Kotlin gecompiled naar JavaScript
Requirements	Geprioriteerde eisen die terugkomen in het eindproduct.
DoD	Definition of Done
CORS	Cross-Origin Resource Sharing
DTO	Data Transfer Object
UI	User Interface
JVM	Java Virtual Machine

2. Organisatie

2.1 Omschrijving

Topicus is in 2001 opgericht met als kernactiviteit IT/Software. Het motto van Topicus is "*Impact met IT"*. Hiermee laten ze zien dat de IT-oplossingen van Topicus echt impact heeft op het dagelijks leven van gebruikers. De kernactiviteit van Topicus is dan ook het maken van hoge kwaliteit software.

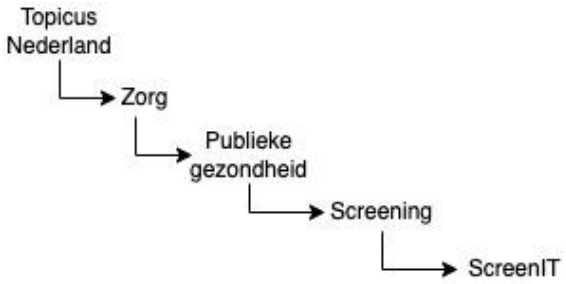
Topicus is een besloten vennootschap en heeft verschillende vestigingen in diverse nederlandse steden met het hoofdkantoor in Deventer. Topicus heeft in totaal om en nabij de 1000 medewerkers.

Topicus heeft 5 verschillende divisies met elk verschillende producten en opdrachtgevers. Deze divisies zijn: Finance, Zorg, Onderwijs, Legal en Overheid. Deze opdracht houdt zich bezig in de divisie Zorg (ca 475 mensen), met als business line Publieke gezondheid (ca 75 mensen) onder het team *Screening* (ca 20 mensen). Zie figuur 1 voor globaal overzicht.

2.2 Structuur

Omdat Topicus erg veel verschillende klanten en producten heeft, is het belangrijk dat er een duidelijke bedrijfsstructuur is. Deze is bij Topicus dan ook duidelijk aanwezig. Topicus Nederland is de organisatie die bovenaan deze opdeling staat. Binnen deze organisatie zijn er verschillende divisies. Deze afdelingen hebben elk weer sub-divisies/business line en deze hebben elk weer teams. Elk team bevat kleinere sub-teams die elk aan een product bouwen.

De structuur naar het sub-team voor het product ScreenIT waarin ik me tijdens het project in bevind loopt het volgende:



** figuur 1: structuur boven het ScreenIT sub-team.*

Het team *Screening*, waar de stage zich in bevindt, maakt 2 producten. Het ScreenIT product ondersteunt bevolkingsonderzoeken naar verschillende soorten kankers.

3. Opdracht

3.1 Achtergrond

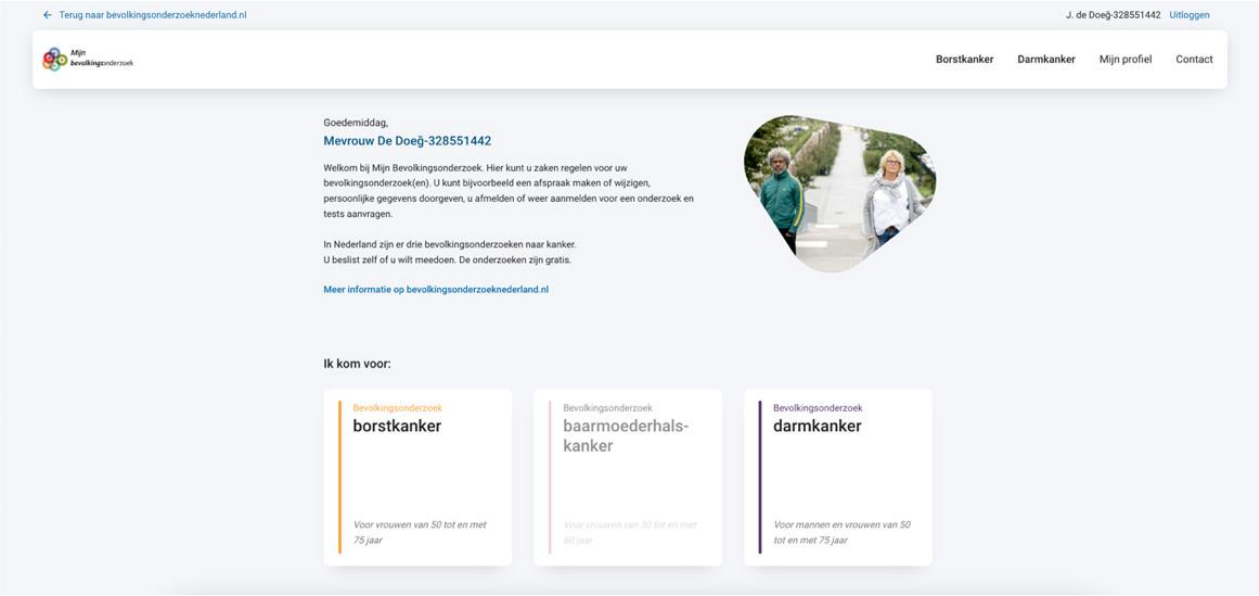
In dit hoofdstuk wordt de achtergrond van het product waar de opdracht zich afspeelt geschetst en wat de aanleiding van de opdracht is. Na de achtergrondomschrijving en de huidige situatie wordt de doelstelling omschreven.

3.1.1 ScreenIT

ScreenIT is een product met allerlei onderhangende applicaties die met elkaar in verbinding staan. Het doel van ScreenIT is het dienen van bevolkingsonderzoeken in Nederland. Op basis van de Basisregistratie Personen (BRP) worden automatisch uitnodigingen verstuurd om deel te nemen aan een bevolkingsonderzoek. Topicus is degene die dit systeem ontwikkeld in opdracht van het RIVM. Het houdt zich momenteel bezig met de opsporing van drie verschillende soorten kankers, namelijk baarmoederhalskanker, borstkanker en darmkanker. Nederlandse mannen en vrouwen in een bepaalde leeftijdsperiode worden aangemeld voor een dergelijk onderzoek. Hierdoor kan dus al in een vroeg stadium verschillende soorten kankers worden opgespoord en worden behandeld. Van de laboranten die röntgenfoto's maken tot aan de radioloog die deze foto's beoordeelt en van de huisarts tot aan de cliënt die inlogt met DigiD om de afspraak te maken, allemaal maken zij gebruik van het ScreenIT systeem.

3.1.2 Cliëntportaal

Het cliëntportaal is een onderdeel van het ScreenIT-systeem. Het cliëntportaal wordt gebruikt door mannen en vrouwen in de Nederlandse bevolking om bijvoorbeeld afspraken voor onderzoeken naar kanker te maken/wijzigen. Voor bijvoorbeeld Baarmoederhalskanker kan er een ZAS (Zelfafname-set) worden aangevraagd als de betreffende vrouw geen uitstrijkje wil maken bij de huisarts. De door de cliënt ingevoerde informatie (zoals telefoonnummer, adres en huisarts) wordt vervolgens opgeslagen in het ScreenIT systeem.



** figuur 2: landingspagina van ScreenIT Cliëntportaal.*

3.2 Doelstelling

Het doel is om gedurende het afstudeerproject te onderzoeken of Kotlin veel voordelen biedt ten opzichte van de gebruikte programmeertalen in de huidige situatie en of er eventuele nadelen aan zitten. Er worden Proof of Concepts gebouwd die aantonen of en op welke manier het mogelijk is om Kotlin als uniforme programmeertaal te gebruiken om codedeling mogelijk te maken. Voordat er aan dit Proof of Concept wordt gebouwd, moet er onderzoek worden gedaan naar de huidige situatie en de structuur die gebruikt zal moeten worden. De resultaten uit het onderzoek en de uitdagingen die gaandeweg aan het licht zijn gekomen zijn gebruikt om het Proof of Concept te verbeteren. Uiteindelijk wordt er een helder advies gegeven waarom wel, of niet Kotlin toe te passen in het complete cliëntportaal en op welke plekken.

3.2.1 Aanleiding

De aanleiding van de opdracht is de eerste verkennende stap om te onderzoeken hoe Kotlin kan bijdragen bij de ontwikkeling van IT-systemen van Topicus. Uiteindelijk is het doel om de TypeScript aan de front-end en de Java aan de back-end te vervangen door Kotlin, om het systeem compleet uniform te maken. De beschikbare opties voor het gebruik van Kotlin voor front-end worden verkend en er wordt gekeken of er voldoende beschikbare frameworks en informatie over is te vinden.

Er is specifiek gekeken naar Kotlin omdat dit al langer bekend staat als een modern en goed alternatief op Java [Singh, V, 2022] en omdat het nu ook beschikbaar is als alternatief op JavaScript voor front-end (KotlinJS) [Jetbrains, 2021b]. Door de multiplatform opzet van JetBrains te gebruiken ontstaat de mogelijkheid tot het delen van code tussen KotlinJVM (voor de back-end) en KotlinJS (voor de front-end).

Het uiteindelijke doel is dus **niet** om frameworks als React en Spring Boot compleet te herschrijven of te vervangen, maar alleen de Topicus geschreven code die de frameworks gebruiken te veranderen naar Kotlin. De logica van de front-end, back-end en gedeelde modules zijn kandidaat voor verandering naar Kotlin. Eén uniforme programmeertaal als Kotlin kan nieuwe mogelijkheden creëren zoals het sneller en makkelijker kunnen switchen tussen modules voor een full-stack ontwikkelaar (door het gebruik van een en dezelfde programmeertaal) en de mogelijkheid tot het delen van code.

3.2.2 Requirements

Voorafgaand in het plan van aanpak zijn de eisen opgesteld waaraan het Proof of Concept moet voldoen. Deze zijn te vinden in onderstaande tabel. Per requirement is een nummer, beschrijving, prioriteit [op basis van MoSCoW, 2015] en of deze functioneel (F) of niet-functioneel (NF) is. De onderstaande requirements zijn alle functionaliteiten en niet-functionele requirements die in het huidige cliëntportaal te vinden zijn. De prioriteit (must requirements) is bepaald op basis van technische inzichten zodat zoveel mogelijk technische basisconcepten zijn geraakt. Deze technische basisconcepten zijn:

- Invullen en valideren van formulieren (bijv. telefoonnummer validatie)
- Communicatie met back-end
- Styling
- Maken van Kotlin *wrappers* voor JavaScript libraries. (bijv. bootstrap componenten)

De must requirements zorgen ervoor dat de technische basisconcepten in de Proof of Concepts worden aangetoond.

	Omschrijving	*Prioriteit*	*F/NF*
1	React met Kotlin	**Must**	**NF**
2	Styling (SASS), Bootstrap met Kotlin	**Must**	**NF**
3	Redux implementeren met Kotlin	**Must**	**NF**
4	Communicatie met back-end realiseren	**Must**	**NF**
5	DTO's, Controllers en Services naar Kotlin (back-end)	**Must**	**NF**
6	Landingspagina (Homepagina)	**Must**	**F**
7	Delen van code tussen front-end en back-end	**Must**	**NF**
8	Telefoonnummer doorgeven/wijzigen	**Must**	**F**
9	Afspraken maken BK/DK	Should	F
10	Afspraken verzetten	Should	F
11	Afspraken annuleren/afmelden	Should	F
12	Overige onderdelen van back-end naar Kotlin	Should	NF
13	Inloggen (zonder DigiD koppeling)	Should	F
14	Adres wijzigen (Validatie- en businesslogica delen)	Should	F
15	Zelfafnameset aanvragen BMHK	Should	F
16	Huisarts doorgeven	Should	F
17	Ontlastingstest aanvragen DK	Should	F
18	DigiD koppeling inloggen	Could	F
19	Definitief afmelden voor onderzoek	Could	F
20	Bezwaar maken	Could	F
21	Nieuwe uitnodigingsbrief aanvragen BMHK	Could	F
22	Deelname uitstellen BMHK	Could	F
23	Privacy/Klachten	Would(n't)	F
24	Contactpagina	Would(n't)	F
25	Overige informatiepagina's (zoals Corona info)	Would(n't)	F

** tabel 1: geprioriteerde requirements*

3.3 Beroepsproducten

De volgende producten worden opgeleverd aan het einde van het project:

- Afstudeerverslag
- Proof of Concepts
    1. Cliëntportaal in Kotlin
    2. Codedeling PoC Maven
    3. Codedeling PoC Gradle
- Presentatie

4. Aanpak

In dit hoofdstuk wordt beschreven op welke manier het project is uitgevoerd en voor welke projectmethodiek er is gekozen. Ook komt het verloop en de gebruikte ontwikkelmethoden aan bod.

4.1 Projectorganisatie

Bij het project waren verschillende partijen betrokken, met elk hun eigen organisatie en rol. In dit geval waren dat 4 partijen/stakeholders, namelijk de student, de bedrijfsbegeleider vanuit Topicus, de stagebegeleider vanuit Saxion Hogeschool en de ontwikkelaars die uiteindelijk met Kotlin gaan werken. De student heeft bij de andere 3 partijen steeds terecht gekund voor eventuele vragen en/of feedback.

In onderstaande tabel meer informatie per betrokkene:

Naam	**Organisatie**	**Rol**	**E-mailadres**
Kevin Scholten	Saxion/Topicus	Student/Afstudeerder	Kevin.scholten@topicus.nl 446792@student.saxion.nl
Robert Spee	Topicus	Bedrijfsbegeleider	Robert.spee@topicus.nl
Dick Heijink	Saxion	Afstudeerbegeleider	j.d.heijink@saxion.nl
Ontwikkelaars	Topicus	Betrokkenen	-

** tabel 2: projectorganisatie*

4.2 Projectmethodiek

Vanaf het plan van aanpak is er gekozen om te werken volgens het Scrum-proces. Er is echter wel van een aantal dingen af geweken. Zo bestond het Scrum team uit 2 deelnemers, namelijk de student als teamlid en de bedrijfsbegeleider als Product Owner.
Een aantal redenen dat voor Scrum is gekozen zijn:

- Het Screening team van Topicus werkt op dit moment ook via Scrum. Hiervoor is vrijwel alles al voor geregeld zoals een digitale backlog en Scrum-bord
- Gedurende de opleiding die de student heeft gevolgd is Scrum ook meerdere keren aan bod gekomen. Hierdoor kon de student erg snel meegaan in het proces.
- Doordat er in het begin geen goede planning kon worden gemaakt wat welke week zou plaatsvinden (doordat er veel moest worden onderzocht, maar onduidelijk was hoelang elk onderdeel zou gaan duren) kon Scrum gebruikt worden om dit bijvoorbeeld op te splitsen en wekelijks bij te sturen.

In het Plan van Aanpak is beschreven hoe het scrumproces tot stand is gekomen en op welke manier deze is uitgevoerd.

4.3 Ontwikkelmethodiek

De ontwikkelmethodiek wijkt af van de gebruikelijke afstudeeropdrachten, omdat er nu is gewerkt aan een bestaand systeem in plaats van een nieuwe applicatie of systeem. Hierdoor is er gekozen voor een verschillende ontwikkelmethodiek.

4.3.1 Ontwerp

Bij het ontwikkelen van nieuwe onderdelen van software is het gebruikelijk dat er eerst een ontwerp wordt gemaakt. Bij de huidige opdracht is de architectuur al bekend, omdat het al een bestaand systeem is. In Onderzoek 1: Huidige en gewenste situaties zijn er ontwerpen gemaakt van de huidige en gewenste situatie van het complete ScreenIT systeem en alle bijbehorende producten.

Er is gekozen voor twee verschillende soorten ontwerpen, namelijk een overzicht van de huidige en gewenste situatie op compile-time en op run-time. Hiervoor is gekozen omdat er op deze manier goed kan worden afgelezen welke modules worden gebruikt door andere modules in het schema op compile-time, en de overige communicatie zoals met de REST back-end op run-time niveau. Op compile-time zijn er aparte ontwerpen voor de huidige- en gewenste situatie en op run-time is er 1 ontwerp voor beiden (omdat hier niets veranderd).

4.3.2 Requirements

De requirements (die zijn te vinden op hoofdstuk 3.2.1) zijn opgesteld tijdens de opstartfase samen met de bedrijfsbegeleider en geprioriteerd via de MoSCoW methode [MoSCoW, 2015). Ook zijn deze eisen ingedeeld als functionele- of niet-functionele eis. Gedurende het project zijn deze eisen bijgesteld, omdat een aantal dingen langer hebben geduurd dan verwacht. Hier is op geanticipeerd en wekelijks op bijgestuurd.

4.3.3 Versiebeheer

Versiebeheer gedurende het hele project is verlopen via Github. Binnen de ScreenIT repository van Topicus is een branch aangemaakt genaamd *stage-kevin*. Binnen deze branch zijn steeds tickets aangemaakt na aanleiding van een ticket op het scrum bord. Nadat de ticket af was, werd hiervoor een *pull request* gemaakt. De bedrijfsbegeleider en een andere medewerker van het Screening team hebben deze steeds beoordeeld en nuttige feedback achtergelaten. Op het moment dat de reviewers akkoord waren, werd deze branch terug in de *stage-kevin* branch *gemerged*.

De Proof of Concepts 2 en 3 (over het codedelen van maven en gradle) zijn beiden op een aparte publieke repository op Github geüpload. Proof of Concept 1 is de code van het cliëntportaal, dat zich bevindt in de privé repository van Topicus. Deze zal worden toegevoegd als bijlage.

4.3.4 Testen

Omdat de back-end al Unit testen bevat en er functioneel niks veranderd, leek het in eerste instantie geen grote uitdaging om deze testen werkend te krijgen met Kotlin klassen. Doordat er is begonnen met het omschrijven naar Kotlin van de stukken code die worden gedeeld, zijn er een aantal testen die hierna niet meer naar behoren werkten. Dit komt door de veranderingen van verwijzingen naar variabelen. Per test zijn deze langsgelopen en (deels) omgeschreven naar Kotlin.

5. Onderzoek

Om aan te tonen waarom Kotlin wel of niet moet worden gebruikt in het cliëntportaal, hebben er een aantal onderzoeken plaatsgevonden. Om aan het eind van het project een goed beeld te krijgen welk advies er aan Topicus zal worden gegeven, kunnen de resultaten van de onderzoeken hier onderbouwing aan geven. In een positieve uitkomst van de onderzoeken en dus ook het advies, zullen de onderzoeksresultaten aangeven hoe het voordeel kan worden bereikt.

5.1 Onderzoeksopzet

Hier wordt beschreven op welke wijze de onderzoeken zijn opgezet. Dit onderdeel bevat de vooraf bepaalde onderzoeksvragen, de onderzoeksmethoden en het verloop van de onderzoeken.

5.1.1 Onderzoeksvragen

Op dit moment is de probleemstelling: de front-end en back-end zijn in verschillende programmeertalen ontwikkeld, waardoor er veel dezelfde onderdelen aan beide kanten bestaan. Dit heeft geleid tot de volgende hoofdvraag:

“Is Kotlin toepasbaar als uniforme programmeertaal met mogelijkheid tot codedeling voor een React front-end en Spring Boot backend?”

Om tot een zo goed mogelijk antwoord op deze hoofdvraag te komen, zijn een aantal deelvragen opgesteld. Elke deelvraag zal een eigen onderzoek bevatten, waar in de conclusie en de onderzoeksresultaten antwoord op wordt gegeven. Dit betreffen de volgende vragen:

1. Wat zijn de huidige en gewenste situaties?
2. Hoe werkt Kotlin samen met verschillende *3rd party frameworks*?
3. Werkt codedeling van React front-end met Spring Boot back-end? Zo ja, op welke manier?
4. Wat zijn de sociale aspecten van implementatie van andere programmeertaal eventuele nadelen die hieruit voortkomen?

5.1.2 Onderzoeksverloop

In dit hoofdstuk wordt beschreven op welke manier de verschillende onderzoeken zijn uitgevoerd. Het onderzoek is namelijk opgesplitst in voor elk van de 4 deelvragen een onderzoek.

5.1.2.1 Vooronderzoek

Om vanaf het begin van het project een goed beeld te krijgen van de huidige situatie, is deze in onderzoek 1 (huidige en gewenste situaties cliëntportaal) onder de loep genomen. Wat is binnen het cliëntportaal de structuur en welke componenten en frameworks zijn hierbinnen gebruikt? Het in kaart brengen van de huidige situatie wordt in dit geval als vooronderzoek gezien. Hiervoor is gekozen omdat het eerst duidelijk moet zijn hoe het huidige systeem eruit ziet, voordat er een dergelijke gewenste/toekomstige situatie kan worden ontworpen.

Onderzoek 1 (huidige en gewenste situaties cliëntportaal) geeft informatie over hoe het React framework en het Spring Boot is toegepast en beschrijft op welke manier Topicus de pagina's en componenten heeft ontworpen.

5.1.2.2 Hoofdonderzoek

Uit het vooronderzoek is gebleken dat de structuur van de React applicatie draait om de 3 bevolkingsonderzoeken met elk andere structuren. Elk bevolkingsonderzoek heeft andere gebruikersacties en is ook op een andere wijze geïmplementeerd. Dit blijkt uit onderzoek 1: De huidige en gewenste situaties.

Het tweede onderzoek gaat over de samenwerking van Kotlin met verschillende frameworks. Hierbij is met name gefocust op de front-end/KotlinJS kant, omdat de KotlinJVM goed samenwerkt met Java frameworks. Dit komt omdat deze beiden worden gecompiled naar JVM. Hoe kunnen JS/TS frameworks worden gebruikt in Kotlin? Zijn er voor de meeste JS frameworks al *wrappers* geschreven voor Kotlin?

Onderzoek 3 heeft zich vooral met het code-deling aspect van de gewenste situatie beziggehouden. Hoe zien bijvoorbeeld de *Data Transfer Object*'s er in de huidige en toekomstige situatie uit.

Doordat de stap naar Kotlin aan zowel de front-end als de back-end voor de betreffende ontwikkelaars erg veel verandering met zich mee kan brengen, heeft er bij onderzoek 4 (sociale aspecten kotlin) een enquête plaatsgevonden onder de ontwikkelaars van het team Screening. Hierbij is onderzocht hoe bekend Kotlin al was en hoe tevreden de ontwikkelaars zijn over de huidige SpringBoot back-end en React front-end. In de vragen van de enquete is veel uitleg gegeven over de geschetste toekomstige situatie en waarom dit veel voordelen kan bieden.

5.2 Onderzoeksruimtes

Volgens stichting HBO-I [HBO-I, 2018] kan de Methoden Toolkit worden gebruikt als een student praktijkgericht onderzoek wil verrichten in de ICT. Om een zo compleet mogelijk eindadvies te geven wordt er aan elke categorie een onderzoek gekoppeld.

5.2.1 Veld

Onderzoeken van toepassing: Onderzoek 4

Onderzoeksruimte veld bevat een verzameling methoden en technieken die ertoe dienen het toepassingsdomein beter te leren kennen. In onderzoek 4 (sociale aspecten kotlin) wordt er een enquête verricht onder ontwikkelaars van het team Screening.

5.2.2 Bieb

Onderzoeken van toepassing: Onderzoek 1, Onderzoek 2, Onderzoek 3

Onderzoeksruimte bieb bevat een verzameling methoden en technieken die dienen tot oriëntatie op beschikbaar werk. Uit vooronderzoek naar Kotlin in combinatie met de verschillende frameworks per module blijkt dat er een gering aantal informatiebronnen beschikbaar waren. Uit onderzoek bleek dat er wel een aantal informatiepunten beschikbaar zijn maar dat er ook een heel aantal geschikt lijken maar dit niet zijn. Dit komt omdat deze uitgelegde informatie alleen geschikt is voor Android, maar niet voor React en dus web.

5.2.3 Werkplaats

Onderzoeken van toepassing: Onderzoek 2, Onderzoek 3

Onderzoeksruimte werkplaats richt zich op het verbeteren van de oplossing zelf, zonder dat het onderzoek nieuwe inzichten over de toepassingscontext oplevert. Door veel te proberen in de huidige omgeving zijn er nieuwe inzichten ontstaan om het Proof of Concept te verbeteren. Zo hoeft er niet meer van programmeertaal worden veranderd als men van de front-end naar de back-end gaat en terug.

5.2.4 Lab

Onderzoeken van toepassing: Onderzoek 3, Onderzoek 4

Onderzoeksruimte Lab bevat methoden die geschikt zijn om de oplossing te toetsen aan een aspect van de toepassingscontext. Doordat de gewenste situatie code deelt tussen modules en er 1 programmeertaal wordt gebruikt, heeft dit invloed op de snelheid van aanpassen. Als de ontwikkelaar een variabele wil toevoegen aan bijvoorbeeld de cliënt, moet dat in dit geval aan beide kanten. Wanneer er 1 klasse is die gedeeld wordt, hoeft dit maar op 1 plek. Ook is het switchen van front-end naar back-end voor een *full stack developer* oogt makkelijker door de ontwikkelaars (gebleken uit onderzoek 4 (sociale aspecten kotlin)) doordat er op beide plekken dezelfde taal wordt gebruikt.

5.2.5 Showroom

Onderzoeken van toepassing: Onderzoek 4, Advies

Onderzoeksruimte showroom bevat een verzameling technieken die als doel hebben de oplossing geschikter te maken voor hergebruik. Het onderzoeken van verschillende frameworks en de impact hiervan op de performance en ervaringen van anderen dragen bij aan de inzet van deze onderzoeksruimte. Uiteindelijk wordt er een prototype en Proof of Concept voorgelegd aan het bedrijf, waarna een pitch plaatsvindt over waarom de stap naar Kotlin een goede of een slechte keuze is.

6. Onderzoeksresultaten

6.1 Onderzoek 1: Huidige en gewenste situaties

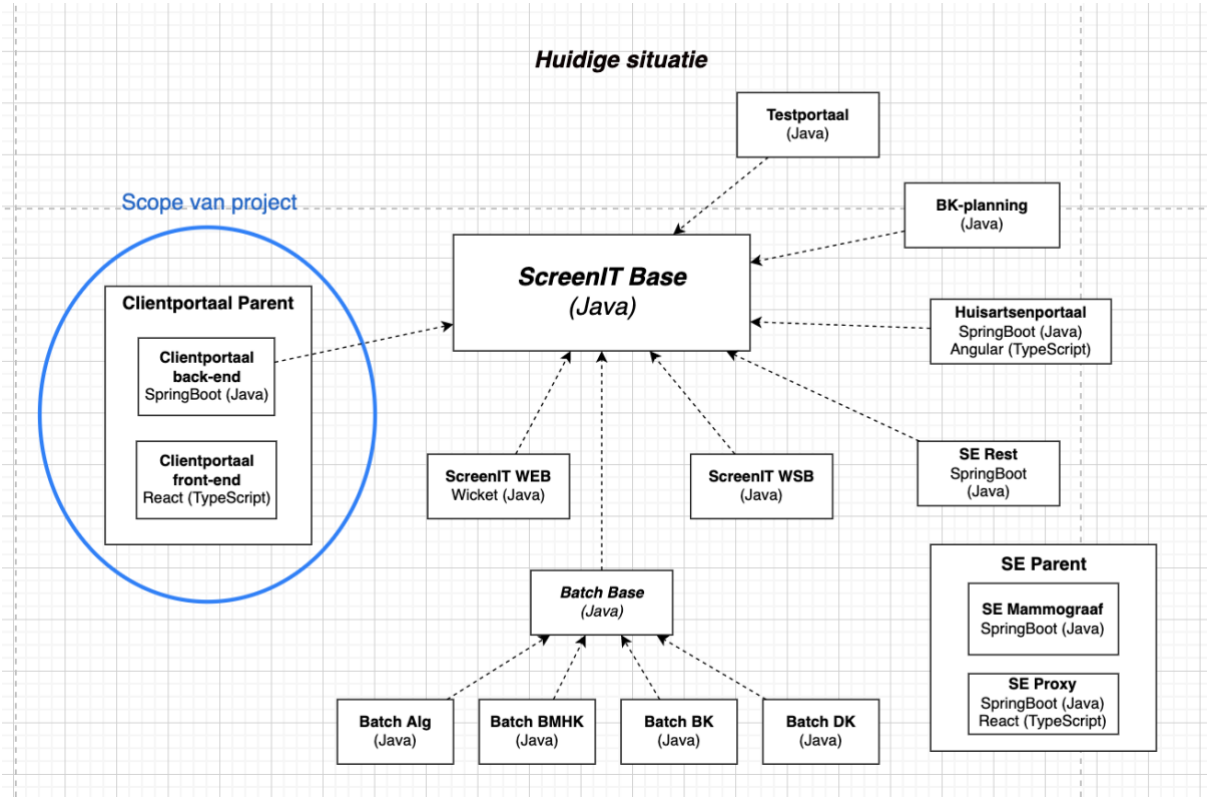
Om een goed beeld te vormen over hoe het huidige systeem eruit ziet en op welke manier er structuur aan is gegeven, wordt tijdens dit onderzoek het cliëntportaalsysteem onder de loep genomen. Het onderzoek bestaat uit twee modules: de front-end en de back-end. Bestaan er al architectuurontwerpen voor het cliëntportaal en zo ja, op welke manier zijn deze tot stand gekomen?

6.1.1 Huidige situatie

6.1.1.1 Uitleg huidige situatie compile time

De huidige situatie zoals te zien in figuur 3 bevat erg veel verschillende modules waarvan de meesten afhankelijk zijn van een of meerdere andere modules. In het midden staat de ScreenIT Base module, waarvan bijna elke verschillende applicatie een dependency naar heeft. Hier zitten onder andere generieke *utilities* in voor meerdere modulen.

De scope van de opdracht bestaat uit de modules omcirkeld met blauw. Deze bevat een clientportaal-parent module met daarin de huidige clientportaal-frontend in React met typescript en de SpringBoot back-end in Java.



** figuur 3: huidige situatie ScreenIT op compile time. Scope van project omcirkeld met blauw.*

6.1.1.2 React front-end

Pagina's en componenten

Voor de front-end is het React framework gebruikt in combinatie met Redux om de *state* van de applicatie bij te houden. Elke pagina van het cliëntportaal heeft de volgende bestanden:

- **JSON:** Elke pagina heeft haar eigen *JSON* bestand met bepaalde variabelen zoals *Strings* die gebruikt kunnen worden in de html elementen.
- **SCSS:** Het SCSS bestand bij elke pagina bevat pagina specifieke styling geschreven in CSS. SCSS (ook wel SASS) wordt gebruikt als uitbreiding op de syntax van CSS. Het ontwerpen wordt hiermee een stuk gemakkelijker, omdat je aanzienlijk meer kan dan met reguliere CSS, bijvoorbeeld: het *nesten* van elementen en het importeren van losse bestanden met variabelen.
- **TSX:** TSX is de Typescript afgeleide van JSX. JSX staat voor Javascript XML en wordt gebruikt in React om html te schrijven in de Javascript code. Het wordt vooral gebruikt om *templates* te maken voor bepaalde pagina's of componenten. In dit bestand staat dus alle markup en Javascript van de cliëntportaalpagina. In deze html kan ook in combinatie met scss direct een *style-klasse* of *id* aan worden geroepen.
- **Test:** Een deel van de pagina's en componenten hebben ook een test TSX bestand. Hierin worden verschillende componenten getest doormiddel van de *TestRenderer* klasse en *Jest*.

API

In de map API zijn verschillende API controllers te vinden die rechtstreeks communiceren met de *Cliëntportaal Back-end.* Deze bestaan uit TSX bestanden waarin synchrone en asynchrone calls worden gedaan naar de back-end. Ook wordt hier de *Dispatch* class gebruikt van het Redux framework, om de state van deze data en de applicatie bij te houden.

Data Transfer Objects (DTO's)

In de map **datatypes** bevinden zich de DTO's (of Data Transfer Objects). Deze objecten zijn van belang tijdens het maken van een nieuwe structuur en de realisatie naar Kotlin. Door deze objecten in Kotlin te schrijven en uiteindelijk in de gedeelde module te plaatsen, worden deze DTO's gedeeld tussen front- en back-end. Dit is mogelijk doordat beide modules Kotlin ondersteunen.

Validators en business logica

In de mappen **validators** en **utils** bevinden zich alle code voor validatie- en businesslogica. Hier worden bijvoorbeeld telefoonnummers en adressen gevalideerd. Deze zijn belangrijk om mee te nemen voor het onderzoek voor het delen van code.

Lifecycle en state (Redux) [ReduxJS, 2015]
Redux is het framework dat binnen React wordt gebruikt om de *state* van de applicatie te beheren. Het bestaat uit een verzameling van bepaalde acties die bepalen welke data de gebruiker op de UI krijgt te zien. De gebruiker kan op zijn plaats via de UI weer de state veranderen. Alle invoeren van de gebruiker en *responses* van de back-end veroorzaken een bepaalde actie die wordt verstuurd (*dispatch*) naar een *Reducer*. Een *reducer* is een functie die de vorige staat van de applicatie meeneemt, en hieruit de nieuwe staat teruggeeft aan de hand van de data die via de actie is doorgegeven. In het geval van gewijzigde of nieuwe data, update de *Reducer* de huidige store. Hierdoor wordt de *state* ook geüpdatet waardoor deze data in de UI ook verandert.

Communicatie met back-end
Voor de communicatie met de back-end vanuit de front-end worden zogenaamde *Thunk actions* gebruikt. Dit is wederom een aspect van Redux dat wordt gebruikt als *middleware* om bijvoorbeeld vertraagde acties op te vangen en af te handelen na contact met de back-end. Dit bevat dus asynchrone acties en API calls. Hiervoor wordt in de huidige situatie het Axios framework (dependency bron 2) gebruikt. Nadat de back-end de opgevraagde data heeft teruggestuurd, kan deze data via Redux worden *gedispatched* en update de UI zichzelf met de juiste data.

6.1.1.3 Spring Boot back-end

Functies bij soort kanker

Om elk bevolkingsonderzoek naar een verschillend soort kanker anders wordt getest of gescreend, heeft elk verschillend soort kanker een aantal unieke functionaliteiten gekoppeld. Deze verschillende functionaliteiten zijn relevant voor een eventuele structuurverandering aan de back-end voor bevolkingsonderzoeken. De code die bij de functionaliteiten horen staan ook kandidaat om naar Kotlin te worden verwerkt. Hieronder zijn deze unieke functionaliteiten weergegeven:

Borstkanker

- **Uitstel:** Bij het borstkanker onderzoek kan uitstel worden gevraagd.
- **Huisarts:** Het koppelen of ontkoppelen van huisarts voor dit specifieke onderzoek.

Darmkanker

- **FIT test (fecal immunochemical test) aanvragen:** Het aanvragen van een FIT test. Dit is een testbuisje waar ontlasting kan worden opgestuurd voor onderzoek door de client.
- **Huisarts:** Het koppelen of ontkoppelen van huisarts voor dit specifieke onderzoek.

Baarmoederhalskanker

- **Brief herdrukken:** Bij het BMHK onderzoek kan de uitnodigingsbrief worden herdrukt. Elke uitnodigingsbrief voor een BMHK onderzoek heeft unieke barcode stickers die de huisarts gebruikt op het potje met weefsel voor de test. Deze moeten altijd uniek blijven. Als je de brief kwijt bent, kan deze dus opnieuw worden besteld.
- **Uitstel:** Het onderzoek kan worden uitgesteld (door bijvoorbeeld zwangerschap).
- **ZAS (Zelfafnameset):** Als de cliënt geen onderzoek door de huisarts wilt ondergaan, kan er een Zelfafnameset worden besteld.

Mapping

Om bepaalde data op de goede manier beschikbaar te hebben voor de front-end, moet er aan de back-end deze data worden ge*mapped*. *Mapping* zorgt ervoor dat de DTO's goed worden omgezet naar database *entities*. In de cliëntportaal back-end wordt *Mapstruct* gebruikt als framework (*Mapstruct, dependency bron 1*).

6.1.2 Gewenste situatie

Door het gebruik van Kotlin verandert er wat in de structuur van de code. Bij onderdeel 1 van dit onderzoek is onderzocht hoe de huidige situatie en structuur er nu uit ziet. Welke onderdelen kunnen worden geschrapt en welke moeten sowieso blijven?

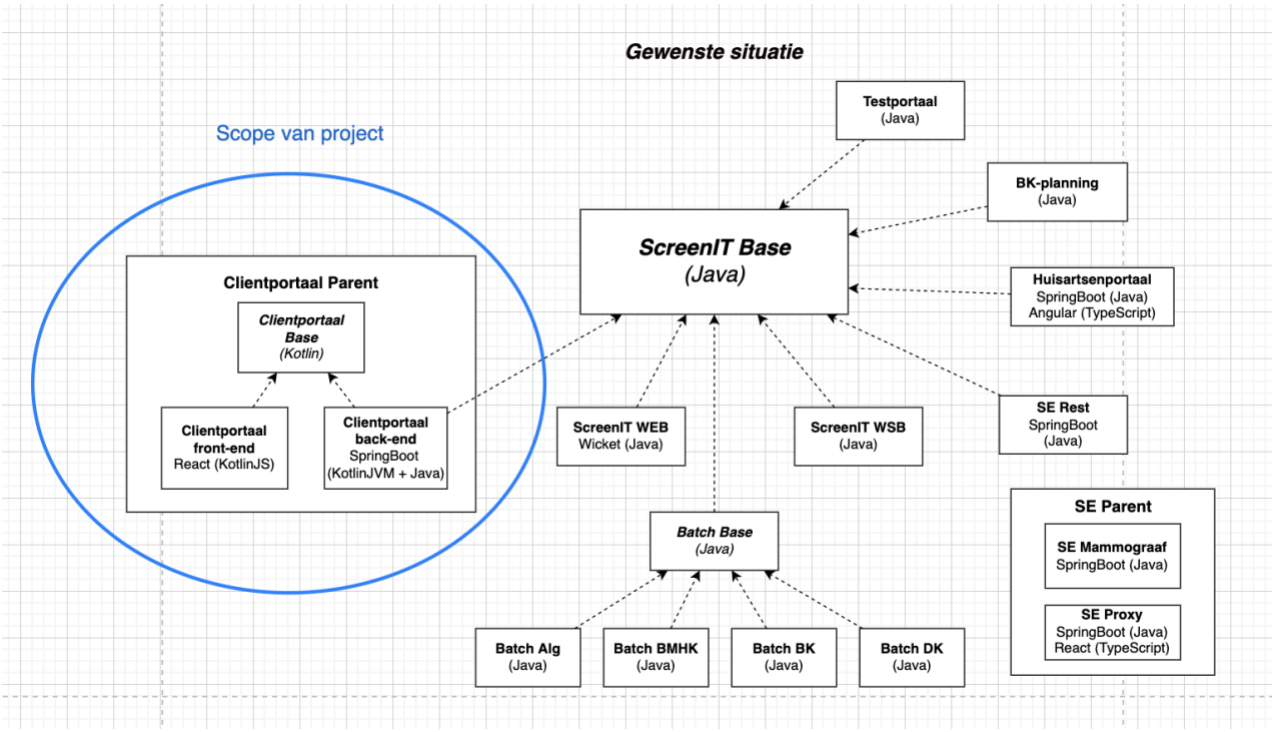
6.1.2.1 Uitleg gewenste situatie compile time

Het verschil tussen de huidige en gewenste situatie bevindt zich in de cliëntportaal module. In figuur 4 is de scope van het project omcirkeld met blauw. Hierin bevindt zich de clientportaal-parent module. Hierin bevinden zich weer 3 andere modules:

- Cliëntportaal Base (Kotlin)
- Cliëntportaal front-end (React met KotlinJS)
- Cliëntportaal back-end (SpringBoot met KotlinJVM)

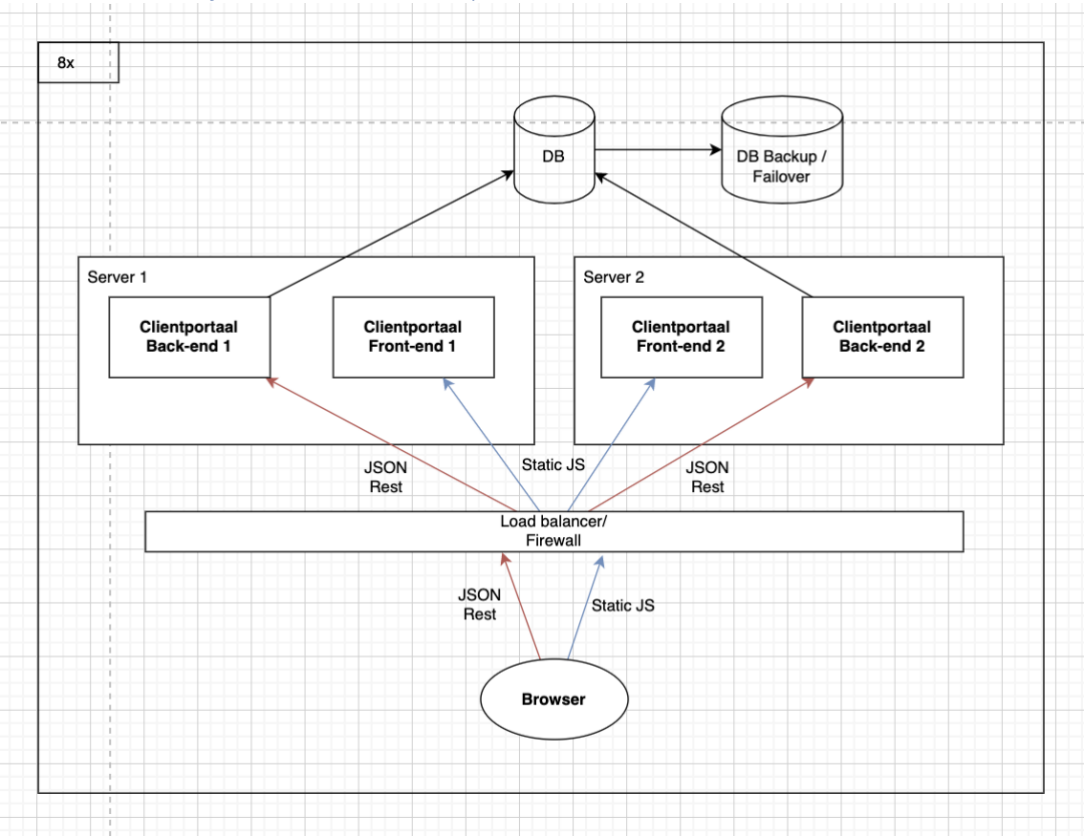
Kotlin code is op compile-time nog aanwezig in de vorm van KotlinJS (front-end) en KotlinJVM (back-end). Deze Kotlin code wordt ingezet **in combinatie met** de bestaande frameworks voor de betreffende modules (React en Spring Boot) en vervangt deze dus niet.

Om het codedelen mogelijk te maken hebben de front-end en de back-end modules een *dependency* naar de Clientportaal Base module (niet te verwarren met de ScreenIT Base module). Deze bevat alle code die kan worden gedeeld en is dus maar één keer aanwezig in plaats van in beide modules apart. De gewenste situatie op compile-time is in figuur 4 in kaart gebracht.



** figuur 4: Gewenste situatie ScreenIT. Scope van project met blauw omcirkeld.*

6.1.2.2 Situaties front- en back-end op run-time



** figuur 5: Huidige en gewenste situatie op run-time.*

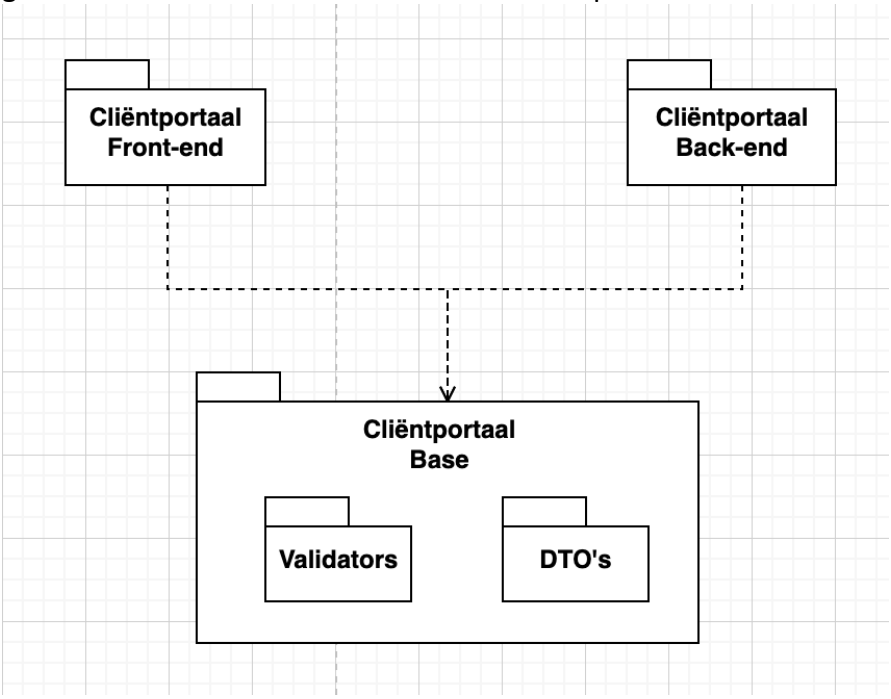
Op bovenstaande figuur 5 is de huidige en gewenste situatie van het cliëntportaal op run-time te zien. In tegenstelling tot de schets op compile-time, verandert er niks tussen de huidige en gewenste situatie op run-time. Dit komt omdat het run-time schema geen *dependencies* laat zien, zoals de *dependency* op de nieuwe cliëntportaal-base module.

Op run-time is er geen Kotlin meer aanwezig, omdat de KotlinJVM aan de back-end is gecompileerd naar JVM Bytecode, en aan de front-end is gecompileerd naar JavaScript.

De gebruiker navigeert via de browser naar het cliëntportaal en vraagt hier het statische Javascript bestand (dat wordt gegenereerd door React) op. Hierna gebeuren er ook een aantal REST calls. Voordat het Javascript bestand wordt opgehaald en hierna REST calls kunnen worden uitgevoerd, loopt het door een *load balancer* heen. Deze *load balancer/firewall* bepaald met welke server de gebruiker gaat verbinden, om zo dus het binnenkomende en uitgaande verkeer te balanceren.

De back-end haalt de data uit de hoofd database met cliëntgegevens. Hierachter hangt ook een andere database met dezelfde data, voor het geval dat de hoofd database niet beschikbaar is. Deze tweede database is dus een zowel een back-up als een failover. Het hele bovenstaande systeem in figuur 5 wordt meerdere keer opgestart, zodat er testomgevingen beschikbaar zijn voor ontwikkelaars om de wijzigingen op te testen.

DTO's
Op dit moment bevinden alle DTO's zich in de "datatype" folder. In de gewenste situatie zal deze map zich buiten de front-end bevinden, zodat zowel de back-end en de front-end deze klasses en types kan gebruiken (doordat Kotlin bij beide wordt herkend). In figuur 4 is deze structuur van de packages te zien ten opzichte van de verschillende projecten. Deze gedeelde code module heet ook wel de cliëntportaal-base module.



** figuur 6: package diagram van gedeelde code gewenste situatie.*

Validators en business logica
Voor de validators geldt hetzelfde als bij de DTO's. De gewenste situatie is dat deze zich buiten de front-end bevinden zodat hier bij beide kanten van het project gebruik van gemaakt kan worden. Een aantal voorzieningen bevinden zich in de **utilities** map die niet allemaal rechtstreeks gedeeld gaan worden. Dit komt omdat bepaalde utilities niet voorkomen in de andere module, zoals alleen front-end gerelateerde logica. Hierdoor blijft het de vraag in hoeverre dit allemaal buiten het front-end project gaat vallen. Er kan voor 3 dingen gekozen worden:

1. Een gedeelde module maken ('base' of 'common') met daarin dezelfde mappenstructuur die wordt aangehouden aan de front- en back-end.
2. Een gedeelde map met alle gedeelde klassen en code.
3. Alles van de back-end delen met de front-end en vice versa.

Deze 3 mogelijke opties zijn gekozen, omdat deze toepasbaar zijn in de huidige situatie van het cliëntportaal. Met de oplossing moet bereikt worden dat de ontwikkelaar gemakkelijk de gedeelde code kan vinden en aanpassen op 1 plek. Optie 1 voldoet hier het beste aan. Ook qua structuur en netheid komt optie 1 het beste uit de verf. Deze 'gedeelde code' map wordt ook wel de 'Base' of 'Common' module genoemd. Dit is terug te zien in figuur 6.

6.2 Onderzoek 2: Samenwerking Kotlin met verschillende frameworks

Omdat het systeem wordt herschreven naar Kotlin, moet het duidelijk worden hoe verschillende frameworks en dependencies er in de huidige situatie worden gebruikt en of deze ook samenwerken met Kotlin. Bepaalde Javascript of Typescript dependencies moeten hiervoor een zogenoemde *wrapper* hebben om deze functionaliteit in Kotlin te gebruiken. Andere dependencies hebben dit weer niet nodig omdat deze wrapper hiervoor al beschikbaar is.

6.2.1 Front-end

6.2.1.2 Dependencies

In onderstaande tabel 3 staan alle frameworks die worden gebruikt in de React (TypeScript) front-end. Er is per framework een beschrijving gegeven van wat het is/doet, of het onderzocht is in het huidige onderzoek en naar welk dependency bron nummer het verwijst. Er is vanwege tijd gekozen om alleen de belangrijke frameworks (die sowieso moeten worden meegenomen om het Proof of Concept te bouwen) te onderzoeken. Dit, om zoveel mogelijk verschillende technische aspecten en functionaliteiten aan te tonen in het PoC.

Front-end npm dependencies die worden gebruikt in de huidige situatie:

Naam	**Beschrijving**	**Onderzocht**	**Dependency Bron nr.**
React	Front-end framework	**Ja**	16
SASS	CSS framework	**Ja**	17
Redux	Lifecycle/State framework voor React	**Ja**	18
Axios	JavaScript http client	**Nee**	2
Bootstrap	Custom css componenten	**Ja**	19
JQuery	JS library voor HTML dom	**Nee**	20
Keycloak	Identiteit toegang manager	**Nee**	12
Jest	JS Test framework	**Nee**	21
Formik	JS Form bouwer	**Ja**	23
Yup	JS Validatie framework	**Ja**	24
Date NFS	JS Date framework	**Nee**	25
TypeScript	Programmeertaal, alternatief voor JS	**Ja**	26
Tiny warning	Warning/melding framework	**Nee**	27
Popper.js	Popup/popover framework	**Nee**	28

** tabel 3: front-end dependencies huidige situatie*

6.2.1.2 Compatibiliteit met Kotlin van belangrijkste frameworks

Een React/Kotlin project kan op drie manieren worden opgezet:

1. Gradle/Groovy met gradle en npm dependencies
2. Gradle/Kotlin met gradle en npm dependencies
3. Yarn (op de oude React manier) met React-scripts-kotlin.

Er is gekozen voor de tweede manier, omdat er bij deze optie zowel Gradle dependencies en NPM dependencies kunnen worden gebruikt. Het build script van Gradle is in dit geval met Kotlin geschreven i.p.v. bij optie 1 met Groovy. Hiervoor is gekozen omdat deze versie nieuwer is en er sowieso al met Kotlin gaat worden gewerkt.

KotlinJS

KotlinJS is de JavaScript variant op de algemene Kotlin. KotlinJS vertaald Kotlin code naar JavaScript code. Als een JavaScript framework moet worden gebruikt in Kotlin, moet er een brug van JavaScript naar Kotlin worden gebruikt. Zo'n brug tussen de twee wordt ook wel een *wrapper* genoemd. Voor de meest bekende JavaScript frameworks zoals React, Redux en Bootstrap zijn al *wrappers* geschreven door JetBrains. Een lijst van deze wrappers is te vinden in bron 13 [Jetbrains, 2017]. Voor de JavaScript frameworks waar geen *wrapper* voor bestaat, moet deze zelf worden geschreven.

SASS

In het cliëntportaal wordt op dit moment gebruik gemaakt van SCSS als uitbreiding op de syntax van CSS. Hiervoor wordt de SASS javascript dependency gebruikt. Net zoals bij de front-end van het bestaande cliëntportaal zou elke component en pagina een eigen SCSS *style* bestand (componentnaam.scss) moeten hebben. Omdat er al styling bestaat in de bestaande front-end, kan deze worden overgenomen naar de nieuwe versie. Per component moet wel worden aangegeven welke styling bij het desbetreffende component hoort. Dit wordt gedaan voor bijvoorbeeld de landingspagina op de volgende manier:

```
fun RBuilder.LandingPage() = child(functionComponent {
    require( o: "css/pages/landing/LandingPage.scss")
```

** figuur 7: Het importeren van het style-bestand in de LandingsPagina.*

Hierna is er in het Gradle build script aangegeven op welke manier de SASS dependency de styling bestanden moet interpreteren.

De KotlinJS React Wrapper (dependency bron 30) heeft ook als alternatief voor css-bestanden, een eigen Kotlin Styling. Hier kan er aan bepaalde markup elementen via Kotlin styling worden toegediend. Doordat Kotlin een *type-safe* taal is, is de css die wordt geschreven hierin dat ook. Op dit moment is hier niet voor gekozen, omdat er al erg veel css beschikbaar is vanuit de huidige situatie.

Bootstrap
De Bootstrap styling werkt gelijk al door het CSS-bestand te importeren vanuit de CDN van Bootstrap. Het belangrijkste onderdeel is dat de bootstrap componenten ook kunnen worden gebruikt in Kotlin, zodat de eerste pagina's van het cliëntportaal kunnen worden gebouwd.

Voor het integreren van bootstrap componenten in de Kotlin/React omgeving was veel research nodig. In eerste instantie is er een Github repository gevonden die precies is wat er wordt gezocht, alleen leek deze na twee keer inspecteren niet echt betrouwbaar. Het is niet van een officiële bron zoals Jetbrains of React maar van een individu. Ook moest er een Github token worden meegegeven dus hierdoor is ervoor gekozen om dit niet te integreren. De documentatie van deze repository is te vinden in dependency-bron 29.

Uiteindelijk is er gekozen voor een *npm dependency* genaamd 'react-bootstrap'. Om deze bootstrap componenten te kunnen gebruiken in de Kotlin componenten, moest er voor elk component een *wrapper* worden gemaakt. Deze *wrapper* is een vertaling van de Javascript component naar het Kotlin component. Dit is op de volgende manier gerealiseerd voor het *Container* component van bootstrap:

```
@JsModule( import: "react-bootstrap/Container")
external val ContainerImport: dynamic
var Container: ComponentClass<ContainerProps> = ContainerImport.default

external interface ContainerProps : Props {
    var className: String? get() = definedExternally; set(value) = definedExternally
    var fluid: Boolean? get() = definedExternally; set(value) = definedExternally
}
```

** figuur 8:* Wrapper voor bootstrap component *'Container'* met bijbehorende *props*.

Redux
Redux als *state container* voor React werkt in Kotlin naar behoren. Wel zijn er een aantal componenten van Redux die op een andere manier moeten worden geinitialiseerd, zoals:

- Het samenvoegen van Reducers (combineReducers)
  Dit lukte in eerste instantie niet via de beschikbare *combineReducers* methode van React-Redux. Dit is opgelost door alle reducers zelf samen te voegen via de *State* van de applicatie. Dit heeft in principe dezelfde werking.
- Het ophalen van data uit de *state* gebeurt via de useSelector *hook*. Dit werkt hetzelfde als in de Typescript versie, alleen moet er rekening worden gehouden met bepaalde Kotlin elementen zoals lambda's.

KotlinJS Debugging
Het debuggen van JVM Java of Kotlin code kan vrij gemakkelijk via IntelliJ door break-points te zetten op locaties in de code. Dit is ook mogelijk in IntelliJ met KotlinJS. Met KotlinJS kan er op drie verschillende manieren worden ge*debugged*:

1. Break points in code (IntelliJ).
2. Log statements in code (loggen naar console)
3. Debuggen via chrome console

Yup
Yup is een JavaScript framework dat de validatie van bepaalde waarden uitvoert. In het cliëntportaal wordt dit gebruikt voor de validatie van de gebruikersinvoeren, bijvoorbeeld:

- Het instellen van telefoonnummers
- Het doorgeven van een tijdelijk woonadres
- Het doorgeven van de huisarts

Het is echter niet makkelijk gebleken om een KotlinJS *wrapper* te schrijven voor Yup, omdat er een groot aantal functies en variabelen worden gebruikt. De validatie is uiteindelijk (te zien in Proof of Concept 3: Codedeling Gradle) gedaan met aparte validatiefuncties.

Formik
Formik wordt gebruikt voor het maken van formulieren die worden gebruikt voor het invoeren van gebruikersgegevens. Het is specifiek gemaakt voor React en ReactNative, en zorgt ervoor dat alle formulieren in de applicatie er hetzelfde uit zien. Ook zorgt formik ervoor dat er validatie plaatsvindt binnen het invulvak. Een voorbeeld hiervan is: Een invulvak mag niet leeg zijn of mag maar een gegeven aantal karakters bevatten. Is dit niet het geval, dan krijgt het invulvak een rode omlijning.

Ook voor formik geldt dat het op dit moment niet mogelijk is om te gebruiken in KotlinJS, omdat er nog geen goede *wrapper* voor bestaat.

6.2.2 Back-end

6.2.2.2 Dependencies

In onderstaande tabel 3 staan alle frameworks die worden gebruikt in de Spring Boot (Java) back-end. Er is per framework een beschrijving gegeven van wat het is/doet, of het onderzocht is in het huidige onderzoek en naar welk dependency bron nummer het verwijst. Er is vanwege tijd gekozen om alleen de belangrijke frameworks (die sowieso moeten worden meegenomen om het Proof of Concept te bouwen) te onderzoeken. Dit, om zoveel mogelijk verschillende technische aspecten en functionaliteiten aan te tonen in het PoC.

Welke frameworks/dependencies worden op dit moment gebruikt aan de back-end?:

Naam	**Beschrijving**	**Onderzocht**	**Dependency Bron nr.**
Spring Boot	Back-end framework	**Ja**	5
JUnit	Java testing framework	**Ja**	8
Lombok	Boilerplate code remover	**Ja**	31
PostgreSQL	Relationeel database system	**Nee**	10
Hibernate	ORM mapper	**Nee**	11
Keycloak	Identiteit toegang manager	**Nee**	12
ActiveMQ	Open Source message broker	**Nee**	7
hyperSQL	Relationeel databasemanagement	**Nee**	13
Mapstruct	Mapping framework	**Nee**	1
JMS	Java message service	**Nee**	14
Mockito	Mocking voor JUnit	**Nee**	15

** tabel 4: back-end dependencies huidige situatie*

KotlinJVM

KotlinJVM (of Kotlin Java Virtual Machine) is ook wel de klassieke variant van Kotlin. KotlinJVM wordt gebruikt als alternatief voor Java, en wordt ook gecompiled naar Java Virtual Machine code. KotlinJVM kan direct worden geconverteerd naar Java, maar wordt afgeraden omdat dan de unieke functies van Kotlin niet optimaal worden benut.

Lombok

Lombok is een framework dat wordt gebruikt in Java om veel *boilerplate code* te elimineren. Via Lombok kan er bijvoorbeeld voor worden gezorgd dat variabelen Non-null worden, zodat er nooit een *null* op het variabele kan staan. Kotlin heeft dit soort opties al door de Kotlin Non-null operator (?) te gebruiken. Ook voor vele andere functionaliteiten van Lombok heeft Kotlin een betere oplossing. Hierdoor is het gebruik van Lombok met Kotlin niet nodig en verouderd. Hierbij is bron 11 geraadpleegd [de Groot, b, 2021].

JUnit

JUnit is een Java test framework dat wordt gebruikt om testen te schrijven om de code te valideren. Het is een belangrijke tool voor test-gedreven ontwikkeling. Elke keer dat de code van het programma wordt gewijzigd, kunnen de testen worden uitgevoerd om te controleren of de code correct werkt. Voor Kotlin werkt dit naar behoren, omdat beiden worden gecompileerd naar bytecode op een Java Virtual Machine.

6.3 Onderzoek 3: Codedeling tussen front- en back-end

Om de maximale efficiëntie uit het gebruik van dezelfde programmeertaal te halen moeten er zoveel mogelijk onderdelen van het systeem worden gedeeld tussen front- en back-end. In dit onderzoek wordt het systeem aan beide kanten onderzocht en wordt er gekeken welke onderdelen vaker terugkomen in front- en back-end. Deze onderdelen worden meegenomen in het onderzoek en kunnen dus na herschrijving worden gedeeld. Figuur 1 van onderzoek 1 (huidige en gewenste situaties cliëntportaal) schetst een beeld van de gewenste situatie op het gebied van codedeling.

6.3.1 Gedeelde onderdelen

Data Transfer Objects (DTO's)

Een Data Transfer Object is een bepaalde opzet voor een object dat gebruikt wordt om over te sturen voor de communicatie tussen front- en back-end. Op dit moment heeft de back-end zelf DTO's geschreven in Java en de front-end DTO's geschreven in Typescript. Er zou gebruikt kunnen worden gemaakt van een tool die Typescript elementen genereert vanuit de Kotlin DTO [Jetbrains, 2022], alleen is ervoor gekozen om beide modules toch om te schrijven. Deze beslissing is tot stand gekomen omdat de voordelen van een uniforme programmeertaal een groot gemak voor de ontwikkelaar met zich meebrengt. Er hoeft immers maar in 1 taal ontwikkeld te worden. De ideale situatie zou dus zijn dat er maar 1 Kotlin object per DTO is die zowel door front- als back-end wordt gebruikt. Voorbeelden van DTO's (die in de back-end voorkomen) zijn: ClientDto en AdresDto

Voorbeeld:

```java
public class ClientDto extends ClientportaalBaseDto
{
    private String voorletters;
    private String aanspreekTussenvoegselEnAchternaam;
    private String bsn;
    private String geboortedatumDisplay;
    private Geslacht geslacht;
    private String adresTekst;
    private String emailadres;
    private TijdelijkAdresDto tijdelijkAdres;
    private String tijdelijkAdresTekst;
    private String telefoonnummer1;
    private String telefoonnummer2;
}
```

Back-end DTO in Java

```typescript
export type Persoon = {
    id: number,
    voorletters: string,
    aanspreekTussenvoegselEnAchternaam: string,
    bsn: string,
    geboortedatumDisplay: string,
    geslacht: Geslacht,
    adresTekst: string,
    emailadres: string,
    tijdelijkAdresTekst: string,
    tijdelijkAdres: TijdelijkAdres,
    telefoonnummer1: string,
    telefoonnummer2: string,
}
```

Front-end DTO in Typescript

** figuur 9: Verschil tussen Client/Persoon DTO per module.*

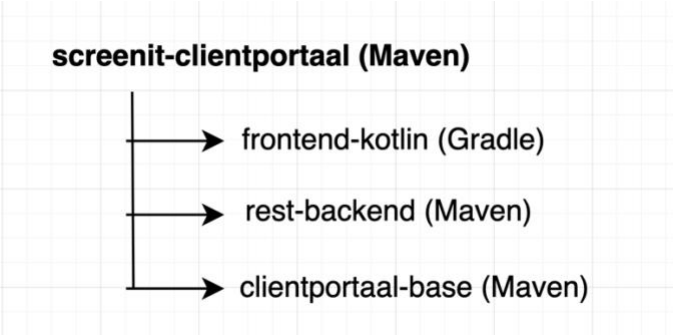
In Kotlin ziet de ClientDto er alsvolgt uit:

```kotlin
data class ClientDto(
        var voorletters: String,
        var aanspreekTussenvoegselEnAchternaam: String,
        var bsn: String?,
        var geboortedatumDisplay: String,
        var geslacht: Geslacht,
        var adresTekst: String,
        var emailadres: String?,
        var tijdelijkAdres: TijdelijkAdresDto?,
        var tijdelijkAdresTekst: String?,
        var telefoonnummer1: String?,
        var telefoonnummer2: String?
): ClientportaalBaseDto()
```

** figuur 10: Gedeelde Persoon/Cliënt DTO voor beide modules.*

6.3.2 Toelichting projectstructuur

Als het delen van code mogelijk wil worden gemaakt, moet er een projectstructuur van modules worden gemaakt binnen de huidige cliëntportaal module. Deze module structuur is te zien op onderstaande figuur.



** figuur 11: module structuur binnen clientportaal (gewenste situatie)*

Om codedeling mogelijk te maken, moet er veel gewerkt worden met 2 verschillende *build automation & dependency tools*: Maven en Gradle. Deze worden onder andere gebruikt om verschillende *dependencies* in te laden en te gebruiken in de code, maar kunnen ook gebruikt worden om een structuur aan het project te geven en codedeling mogelijk te maken.

In figuur 11 is te zien dat op het moment van de ontwikkeling van de cliëntportaal-front-end dit de enige module is die Gradle gebruikt. In de volgende paragraaf (6.3.3) zijn de resultaten te lezen van het onderzoek naar de samenwerking van beide frameworks.

6.3.3 Mogelijke oplossingen codedelen

KotlinJS aan de front-end werkt alleen met Gradle. Om codedeling mogelijk te maken, moet er een *parent-child structure* of een *import structure* worden gerealiseerd. Er zijn 5 verschillende mogelijke oplossingen onderzocht. Per onderzochte mogelijkheid is omschreven op welke manier dit is aangepakt en of de manier als een goede oplossing wordt geacht. **De 5 onderzochte oplossingen zijn de volgende:**

1. Alleen Maven gebruiken
2. Gradle project met een Maven POM bestand
3. Zowel Maven als Gradle gebruiken
4. Alleen Gradle gebruiken
5. Code extern opslaan

6.3.3.1 Alleen Maven gebruiken (gradle project converteren)

Het is bekend dat Maven een goede *parent-child* structuur kent en modules over en weer kunnen worden geïmporteerd als *dependencies*. Ook gebruikt het hele ScreenIT product met alle onderliggende applicaties al Maven. De eerst onderzochte mogelijkheid is dan ook om het frontend-kotlin project dat nu Gradle bevat te converteren naar Maven.

Als eerst is er geprobeerd om de huidige build bestanden van Gradle met de dependencies te converteren naar een maven POM-bestand. Hierdoor kan het project toegevoegd worden aan de *parent* als module en kan het de *base* module (gedeelde bestanden module) importeren als dependency. Via de *maven-publish* plugin [Gradle, z.d.a] voor Gradle kon er een Maven POM bestand worden gegenereerd, echter bevatte deze alleen de *dependencies* en géén plugins en compile opties. Deze konden handmatig wel worden toegevoegd, alleen worden er verschillende *npm packages* gebruikt die hier geen deel van uit konden maken. Hierdoor werden verschillende onderdelen van de huidige applicatie niet meer bruikbaar omdat de *dependency* hiervan miste.

6.3.3.2 Gradle project met een Maven POM

De tweede onderzochte mogelijkheid is het Gradle project laten zoals het is (front-end in Kotlin), maar Maven als framework toevoegen aan deze module. Op deze manier wordt er een POM-bestand gegenereerd die wordt aangepast naar de goede data. Vanuit hier zou dit POM-bestand dus kunnen deelnemen als *child* module bij de *parent.* Voor de maven projecten onderling werkt dit perfect (zie 15.2 Proof of Concept 2), maar omdat het project nog steeds als een Gradle module wordt gebouwd zijn de gedeelde bestanden toch niet bereikbaar. Dit komt doordat de Kotlin front-end alleen beschikbaar is in Gradle, hierdoor kan dit project niet bij de door Maven gedeelde code.

6.3.3.3 Zowel Maven als Gradle gebruiken

De derde onderzochte mogelijkheid is het gebruik van Maven en Gradle op dezelfde module. De gedachtegang hierachter is dat de Maven projecten via de bijbehorende *parent* en *shared* module de codedeling plaatsvindt en dat voor de Gradle projecten er een soortgelijke structuur wordt opgezet dwars over de Maven structuur heen. Hierdoor heeft elke module een eigen aanspreekpunt die wordt herkend.

6.3.3.4 Alleen Gradle gebruiken

Om het systeem en de bijbehorende modules zo uniform mogelijk te houden, is de optie om alle onderliggende modules te converteren naar Gradle ook onderzocht. Hierbij zou het in het geval van het cliëntportaal neerkomen op het converteren van de huidige back-end naar Gradle. De *shared* module zou in dit geval ook via Gradle worden gebouwd. Een voorbeeld hiervan kan worden gevonden in 15.3 Proof of Concept 3.

Deze optie wordt op dit moment niet geïmplementeerd in het cliëntportaal, doordat de modules hierin dependencies hebben op de ScreenIT Base module (voor overzicht zie figuur 4). Hierdoor treedt dus hetzelfde probleem op.

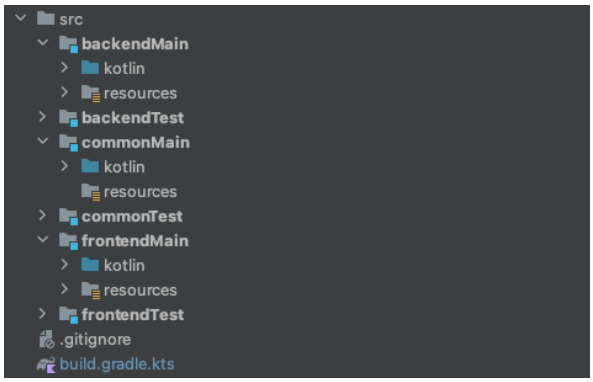
6.3.3.5 Code extern opslaan (niet wenselijk)

De vijfde optie is het extern opslaan van code, waardoor de connectie via een *build automation tool* zoals Gradle of Maven niet nodig is. Deze optie is niet wenselijk als oplossing binnen het cliëntportaal, omdat er hierdoor de gedeelde bestanden alsnog moeten worden opgehaald uit een externe bron. Dit zou kunnen via een externe *repository* (Topicus gebruikt Nexus) of via een API endpoint.

6.3.4 Uitleg oplossing Proof of Concept

Van alle bovenstaande oplossingen is oplossing 4: alleen gradle gebruiken uitgewerkt in Proof of Concept 3. Dit proof of concept is te vinden in bijlage 15.3 Proof of Concept 3. Voor deze optie is gekozen omdat dit proof of concept (los van het cliëntportaal) goed laat zien dat het via een multi-platform project mogelijk is om code te delen tussen een Javascript web module (KotlinJS) en een JVM back-end module (KotlinJVM).

In Proof of Concept 3 is te zien dat er één gezamelijke Gradle build file is, waar via *source sets* dependencies zijn opgedeeld. Hier bestaan er 3 benoemenswaardige modules: een front-end KotlinJS module, een back-end KotlinJVM module en een JS/JVM Common module. Deze common module bevat dependencies die speciaal gemaakt zijn voor deze *common* module. Hier kan code in worden gezet dat gedeeld wordt tussen de JS en JVM-modulen. Zie figuur 12 voor de mappenstructuur binnen het Proof of Concept 3.



** figuur 12: mappenstructuur Gradle project met gedeelde code.*

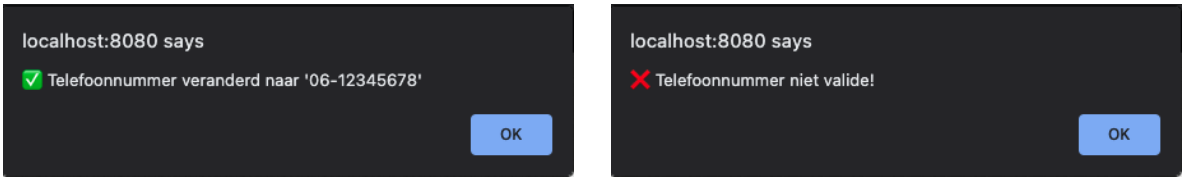
Proof of Concept 3 is een React/Redux front-end in Kotlin geschreven die met een simpele call naar de back-end de gegevens ophaalt van een werknemer. Deze front-end laat de gegevens van een werknemer zien die uit de back-end is gehaald. Hierbij verloopt deze informatie ook via de normale manier van Redux. Ook is het mogelijk om het telefoonnummer te wijzigen. De validatie van dit telefoonnummer is ook gedeeld tussen front- en back-end en wordt op beide modules gecontroleerd.

In de User Interface is erg simpel alle informatie af te lezen. Er is echter niet veel aandacht besteed aan de styling, maar meer aan de technische werking van het PoC. De UI ziet er als volgt uit:

** figuur 13: de User Interface van Proof of Concept 3.*

De back-end bestaat uit een Spring Boot webserver, welke een werknemer object bevat die via een endpoint kan worden opgehaald. Het 'Werknemer' object staat in de *common* module en wordt dus gebruikt door zowel de front-end en de back-end. Ook is er een end-point om het telefoonnummer te wijzigen. Meer informatie op de Github repository pagina te vinden in [Proof of Concept 3.]

De validatie van het correcte telefoonnummer formaat wordt gedaan vanuit een aantal functies overgenomen vanuit het cliëntportaal, herschreven in Kotlin. Wanneer de gebruiker een foutief werknemer telefoonnummer invult, verschijnt er een melding. Wanneer het telefoonnummer correct is, ook. De meldingen zijn te zien in figuur 14.



** figuur 14: meldingen over validatie telefoonnummer aan de front-end.*

Dit Proof of Concept 3: Codedeling Kotlin Gradle toont dus aan dat het mogelijk is om Kotlin code te delen tussen een KotlinJS React front-end en een KotlinJVM Spring Boot back-end. De gedeelde code is de Werknemer klasse (DTO) en de telefoonnummervalidatie.

6.4 Onderzoek 4: Sociale aspecten verandering naar Kotlin

Naast de functionele voor- en nadelen van het uniform gebruiken van een programmeertaal, kan het voor de ontwikkelaars die aan het systeem werken ook sociale voor- en nadelen hebben. De ontwikkelaars zijn immers degenen die er later aan verder moeten werken en het zou geen zin hebben om het complete systeem om te bouwen naar Kotlin als de ontwikkelaars hier niet blij mee zijn.

Er is met behulp van de onderzoeksruimte Veld een anonieme enquête gehouden onder de ontwikkelaars om deze mening te peilen. Ook is er in de enquête aangegeven dat als er eventuele vragen zijn dat er contact kan worden opgenomen om deze te stellen. Er zijn geen verdere vragen binnengekomen, wel een aantal tips.

De URL naar de enquête is te vinden in bijlage 1.

6.4.1 Samengevatte reacties op vragen enquête

Er zijn 6 reacties binnengekomen van ontwikkelaars van het team *Screening*. Verrassend is dat alle reacties op de enquête positief zijn over de verandering naar Kotlin op beide modules. In deze paragraaf worden de antwoorden per vraag samengevat en kort beschreven wat dit betekent voor de conclusie van dit onderzoek.

6.4.1.1 Ben je bekend met Kotlin als programmeertaal?

Deze vraag is gesteld om te pijlen hoe bekend de ontwikkelaars zijn met Kotlin an sich.

Antwoorden samengevat

Deze vraag is een gesloten vraag, met de volgende antwoordmogelijkheden: “Ja”, “Nee” en “Een beetje”. Alle ontwikkelaars gaven aan bekend te zijn met Kotlin (100%). Dit antwoord is positief voor de uitkomst van de enquête, omdat het niet veel zin heeft als er “Nee” antwoorden tussen zitten.

6.4.1.2 Heb je al eens gebruik gemaakt van Kotlin? Zo ja, in combinatie met welk framework/platform?

Deze vraag is gesteld als opvolging van de vorige, om te pijlen of de ontwikkelaars al eens met Kotlin hebben gewerkt. Zo ja, is dat dan met Kotlin op zichzelf of in combinatie met een framework/platform?

Antwoorden samengevat

Veruit de meeste antwoorden gaven aan Kotlin te hebben gebruikt in combinatie met Android. Dit is niet verrassend, omdat Kotlin op dit platform het grootst is. Naast Android gaven ook een aantal ontwikkelaars aan het in combinatie met een back-end framework te hebben gebruikt (zoals Spring Boot). Een iemand gaf aan het niet gebruikt te hebben, maar wel het een en ander erover heeft gezien.

6.4.1.3 Ben je tevreden met de huidige versie van het cliëntportaal? (met betrekking tot de gebruikte programmeertalen front- en back-end)

Deze vraag is gesteld om te pijlen of ontwikkelaars blij zijn met de huidige situatie van het cliëntportaal aan de front- en back-end. Zijn de ontwikkelaars in voor een verandering of blijven ze vrij conservatief met de huidige situatie?

De respondenten hadden de keuze om een cijfer te kiezen van 1 tot en met 5. 1 staat voor 'niet tevreden' met de huidige situatie van het cliëntportaal en 5 staat voor 'zeer tevreden'.

Antwoorden samengevat

Vrijwel iedereen heeft het cijfer 4 gegeven (4 / 5). Dit geeft aan dat er een tevredenheid heerst over het huidige cliëntportaal, maar dat er dus nog ruimte is voor verbetering. Het uniform maken van de programmeertaal in het cliëntportaalsysteem kan hier dus aan bijdragen.

6.4.1.4 Wat zou je veranderen aan de front-end (React/TypeScript)?

Deze vraag is gesteld zoals de titel zegt: wat zouden de ontwikkelaars anders zien aan de huidige front-end die is geschreven in React/TypeScript? Geven ze bepaalde nieuwe inzichten die ik wellicht kan meenemen in het project?

Antwoorden samengevat

De antwoorden op deze vraag lopen wat verder uiteen dan voorgaande vragen. Een aantal ontwikkelaars geven inderdaad aan dat het mooi zou zijn als er code gedeeld zou gaan worden zodat code duplicaten voorkomen worden. Verder worden er verschillende andere frameworks genoemd die in de plaats voor anderen zouden kunnen komen en worden begrippen als *null* en *undefined* genoemd. Dit is ook een probleem dat Kotlin zou kunnen oplossen doordat er met Kotlin erg *null-safe* kan worden geprogrammeerd.

6.4.1.5 Wat zou je veranderen aan de back-end (SpringBoot/Java)?

Deze vraag is gesteld om te peilen wat de ontwikkelaars anders zouden willen zien aan de huidige back-end van het cliëntportaal, die nu is gebouwd in Spring Boot met Java.

Antwoorden samengevat

Verrassend was dat bij de back-end, die op dit moment is gebouwd in Java met Spring Boot, er niet echt toe was aan een drastische verandering. Als enige werd Kotlin genoemd als alternatief voor Java en of er eventueel *cloud based microservices* konden worden geïmplementeerd.

6.4.1.6 Zou je openstaan om in het vervolg Kotlin te gaan gebruiken in het cliëntportaal? Waarom wel/niet?

Deze vraag is gesteld om te peilen of de ontwikkelaars open staan om in het vervolg te werken met Kotlin aan de front- en back-end in plaats van TypeScript en Java.

Antwoorden samengevat

Vrijwel alle antwoorden gaven aan dat als het leidt tot minder dubbele code, dat het antwoord Ja is op de vraag. Een antwoord gaf aan dat als het te moeilijk wordt om 3rd-party componenten te integreren met Kotlin, dat het antwoord nee is. De uitkomsten van Onderzoek 2: Hoe werkt Kotlin samen met verschillende frameworks? geven hier een goed beeld over.

6.4.1.7 Zijn de voordelen van het gebruik van Kotlin (codedeling) genoeg om de overstap te maken?

Deze vraag is gesteld om te peilen of de ontwikkelaars de voordelen van het gebruik van Kotlin zwaarder vinden wegen dan de nadelen, en dus de overstap durven te maken.

Antwoorden samengevat

Alle antwoorden geven aan dat het afhangt van de tijd die het kost om alles om te schrijven en de complexiteit van de componenten in Kotlin. Kan de huidige situatie redelijk snel worden omgeschreven, dan zijn de meeste respondenten voor het idee van Kotlin. Weer komt het aspect van het elimineren van dubbele code erbij kijken. Het code delen, wat dit dus tegengaat wordt gezien als een groot voordeel.

6.4.1.8 Heb je nog vervolgvragen voor mij over dit onderwerp? (Vermeldt je naam bij in de vragen).

Deze vraag is gesteld voor mensen met eventuele vervolgvragen of feedback op de enquête.

Antwoorden samengevat

Eén iemand was benieuwd naar de mogelijkheden van Kotlin met betrekking tot UI en de functionaliteiten die niet mogelijk zijn. Verder was er nog een tip voor de volgende keer om wat meer gesloten vragen toe te voegen, zodat er grafieken en diagrammen konden worden gemaakt.

6.4.2 Conclusie sociale aspecten/meningen ontwikkelaars

Uitkomst onderzoek meningen verandering naar Kotlin: Vrij positief

Vrijwel alle antwoorden op de enquête zijn vrij positief over de verandering naar Kotlin. De ontwikkelaars zien de voordelen en zijn van mening dat deze zwaarder wegen dan de nadelen. Wel werd er aangegeven dat als de conversie te lang duurt of dat blijkt dat het te complex wordt, dan er wellicht een negatieve uitkomst kan worden gegeven. Ook speelt de ondersteuning van Kotlin bij verschillende frameworks een rol op deze uitkomst.

7. Advies

Om Topicus na het project een goed beeld te geven over de uitkomsten en conclusies van de onderzoeken, wordt er in dit hoofdstuk per onderzoeksvraag een advies geformuleerd. Dit advies kan worden gebruikt in de doorontwikkeling van het Kotlin cliëntportaal en voor eventuele toekomstige projecten die worden geschreven in Kotlin. Ook zijn de uitkomsten van het onderzoek naar het delen van code interessant voor toekomstige projecten. In Hoofdstuk 8: Implementatieplan wordt uitgelegd op welke manier dit kan worden gerealiseerd.

7.1 Advies per onderzoeksvraag/conclusie

Per onderzoeksvraag is er een onderzoek uitgevoerd. Deze resultaten worden samengevat als conclusies en hieruit kan een advies worden geformuleerd.

7.1.1 Huidige en gewenste situatie

Onderzoek 1 (huidige en gewenste situaties cliëntportaal) gaat over het in kaart brengen van de huidige situatie van het ScreenIT systeem en met name het cliëntportaal wat hieronder valt. Doordat de huidige situatie in kaart is gebracht was het al gauw duidelijk wat de gewenste situatie moest inhouden. Het advies luidt dus:

Om de structuur op te zetten voor het delen van code is het van belang dat er een module wordt ingericht die de gedeelde code gaat huishouden. Hiervoor wordt er in de parent module een *Base module* aangemaakt die de andere modules als dependency hebben. Het advies is dus de structuur aan te houden die in 6.1.2 Gewenste situaties beschreven is.

7.1.2 Kotlin samen met verschillende frameworks

Onderzoek 2 gaat over de compatibiliteit van Kotlin in combinatie met de verschillende frameworks die zijn gebruikt om het cliëntportaal mee te bouwen. Hierbij is het duidelijk geworden dat na enig uitzoekwerk het wel mogelijk was om de hoofd frameworks die in het cliëntportaal worden gebruikt toe te passen in Kotlin. Voor bijvoorbeeld bootstrap was het nog wel nodig dat er een *wrapper* moest worden geschreven voor de gebruikte componenten. KotlinJS is nog vrij nieuw en er is nog geen hoog *framework adoption rate*. Hierdoor heeft het gedurende de realisatiefase voor sommige frameworks erg lang geduurd voordat er een oplossing is gevonden. TypeScript is dus, voor een snellere implementatie van frameworks (die aan bod zijn gekomen tijdens onderzoek 2 (kotlin met verschillende frameworks)), een betere oplossing (voor nu).

7.1.3 Codedelen front- en back-end

Onderzoek 3 gaat over de mogelijkheid tot codedelen tussen front- en back-end modules. Uit dit onderzoek is gebleken dat de mogelijkheid er is om code te delen tussen een KotlinJS en KotlinJVM module, maar dat hiervoor een multiplatform project moet worden gemaakt. Op dit moment is het alléén mogelijk met Gradle als *build script*. Codedelen met maven is mogelijk, maar alleen tussen JVM modules (geen JS en dus geen front-end). Voor het cliëntportaal is het dus van belang dat de modules Gradle als *build script* hebben. Dit kan een goede oplossing zijn aangezien Gradle het modernere alternatief van Maven is [Gradle, z.d.b].

7.1.4 Sociale aspecten Kotlin

De reacties op de enquête over de verandering naar Kotlin onder de ontwikkelaars waren vrijwel allemaal positief. Er is weinig tot geen weerstand tegen de migratie naar Kotlin. Wel gaven zij aan dat als de migratie van het complete cliëntportaal te lang duurt of het te complex wordt, dit een negatief effect kan hebben. Ook gaven de ontwikkelaars aan dat als de ondersteuning van Kotlin bij frameworks niet of matig aanwezig is, dit ook kan leiden tot een negatieve uitkomst van de beslissing tot migratie. Terugkoppelend op de conclusie van onderzoek 2 (kotlin met verschillende frameworks) en deze sociale aspecten mee te wegen, is het dus wellicht te vroeg om de migratie op dit moment te maken.

7.2 Eindadvies met conclusie

Tijdens de realisatiefase is gebleken dat erg veel dingen nog niet vanzelfsprekend waren, waar dat met TypeScript vaak wel het geval was. Op het moment van onduidelijkheden voor vrij generieke problemen in React was er met Kotlin vaak geen directe oplossing, waar dat met TypeScript wel het geval zou zijn. Dit komt omdat KotlinJS nog erg nieuw is en door niet veel mensen wordt gebruikt als vervanger voor JavaScript of TypeScript. De onderzochte frameworks (React, Redux, Bootstrap etc.) zijn echter wel werkend gekregen in combinatie met Kotlin. Ook uit het onderzoek over de sociale aspecten komt terug dat men er wel voor open staat, tenzij de migratie teveel tijd in beslag neemt of het systeem er te complex door wordt.

Het advies luidt dus als volgende: de front-end omschrijven naar Kotlin wordt op dit moment afgeraden, en eventuele vervolgprojecten met Kotlin als front-end kunnen beter *on-hold* worden gezet. Een goede optie zou zijn om deze kwestie over een jaar te herevalueren, omdat er tegen die tijd wellicht meer *wrappers* beschikbaar zijn en het bekender is onder ontwikkelaars. De back-end daarentegen is klaar voor migratie naar Kotlin, omdat dit direct wordt vertaald naar JVM (net zoals Java), in plaats van JS.

8. Implementatieplan

Om voor vervolgprojecten duidelijk te hebben op welke manier een multi-platform project kan worden ingericht, bevat dit hoofdstuk een implementatieplan. Op welke manier kan er code worden gedeeld tussen JVM (back-end) en JS (front-end) modules en hoe kan dit structureel het beste worden gerealiseerd?

8.1 Implementatiekeuzes

Er zijn vooraf een aantal keuzes gemaakt qua frameworks, build scripts en programmeertalen. Deze worden behandeld in de huidige paragraaf. In onderstaande tabel staan deze keuzes en een beschrijving over waarom dit de beste keuze is.

Naam	**Soort**	**Beschrijving**	**Gebruikte Versies**
React	Front-end framework	Het React framework werkt het beste samen met KotlinJS.	17.0.2
KotlinJS	Programmeertaal	Kotlin als uniforme programmeertaal tussen modules.	1.6.10
KotlinJVM	Programmeertaal	Kotlin als uniforme programmeertaal tussen modules.	1.6.10
Project JDK	Development Kit	Versie JDK-11 is gekozen omdat deze keuze ook is gemaakt in het complete huidige ScreenIT systeem.	Openjdk-11
Gradle	Build System	KotlinJS aan de front-end is alleen beschikbaar met Gradle als Build System.	7.4
Spring Boot	Back-end Framework	Spring Boot en Kotlin werkt goed samen. Kotlin als alternatief op Java.	2.6.4

** tabel 5: implementatiekeuzes frameworks/talen*

8.2 Vervolgplan

Om erachter te komen op welke manier het cliëntportaal de conversie naar Kotlin verder moet ondergaan, is er in deze paragraaf een kort vervolgplan geschreven.

1. Front-end (React) doorbouwen (van Proof of Concept 1) naar Kotlin.
2. Back-end (Spring Boot) converteren naar Gradle.
3. Back-end (Spring Boot) per klasse naar Kotlin schrijven.
4. Modules in Multiplatform Gradle plaatsen (zie 8.3 Stappenplan voor uitleg)
5. Gedeelde code plaatsen in *Common* module.

8.3 Stappenplan codedelen nieuw project

Om een nieuw project op te zetten moeten er een aantal stappen worden gevolgd. IntelliJ heeft een gemakkelijke functie om een multi-platform project op te zetten.

Stap 1: Maak via IntelliJ een nieuw project via de IntelliJ 'New Project' wizard.
Stap 2: Selecteer Kotlin uit de linker rij.
Stap 3: Selecteer als Project Template 'multiplatform – Full stack web application'
Stap 4: Selecteer als Build System 'Gradle Kotlin'
Stap 5: Selecteer als Project JDK 'openjdk-11'
Stap 6: Klik op volgende
Stap 7: Selecteer voor elke module de gewenste *templates*.

Het multi-platform project is nu gemaakt. In de JVM-map wordt de back-end geïmplementeerd, in de JS map de front-end. Alle dependencies en plugins staan gezamenlijk in de parent *build.gradle.kts* bestand. Per module kunnen deze hier worden aangegeven.

9. Definition of Done

Om later vast te stellen of er aan de eisen is voldaan, moet vooraf een *definition of done* worden opgesteld. Per software gerelateerde ticket wordt dit bijgehouden door middel van een tabel, waar duidelijk is vastgesteld wanneer er is voldaan aan de eis en deze dus kan worden afgesloten. Ook wordt er vastgesteld welke test(en) er uitgevoerd moeten worden om aan deze eis te voldoen.

Let op: dit geldt vooralsnog alléén voor de *Must requirements.*
De Definition of Done omschrijvingen zijn te vinden in bijlage 3.

9.1 Behalen van Definition of Done requirements

Vanaf het begin van het project zijn er een aantal eisen gesteld. Deze eisen zijn geprioriteerd met behulp van MoSCoW. Meer over deze eisen en de prioritering hiervan is te vinden onder de paragraaf 3.2.1 Requirements. Hieronder is de voortgang van de requirements aangetoond in een tabel. Ook is er beschreven wat er precies is afgerond en of er aan de definition of done is voldaan.

#	Naam	Omschrijving afgerond	DoD behaald
1	**Opzetten React/Kotlin branch**	Initieel project	**Ja**
2	**Opzetten Redux**	Redux en alle bijbehorende componenten zijn opgezet binnen het clientportaal-kotlin project. Alle cliëntdata verloopt via Redux states.	**Ja**
3	**Landingspagina**	Landingspagina is zichtbaar voor vooraf bepaalde cliënt met ingevoerd BSN nummer. Afhankelijk van geslacht en leeftijd van cliënt zijn bepaalde bevolkingsonderzoeken zichtbaar op Landingspagina.	**Ja**
4	**Code delen**	Codedeling is mogelijk tussen verschillende modules (JS of JVM) en aangetoond in Proof of Concept 2	**Deels:** Aangetoond in PoC 2, niet in cliëntportaal
5	**Telefoonnummer doorgeven/wijzigen**	Telefoonnummer wijzigen is mogelijk via de ‘profiel’ pagina.	**Ja**
6	**Inlogfunctionaliteit (zonder DigiD)**	-	**Nee**
7	**Afspraken maken BK/DK**	-	**Nee**
8	**Afspraken verzetten**	-	**Nee**
9	**Afspraken annuleren/afmelden**	-	**Nee**
10	**Adres wijzigen/doorgeven tijdelijk adres**	-	**Nee**

** tabel 6: behaling definition of done requirements.*

10. Reflectie

In dit hoofdstuk blik ik terug op de stageperiode met betrekking tot het onderzoek en de dingen die ik heb geleerd van het scriptieproces. Welke onderdelen gingen goed en welke verbeterpunten kan ik meenemen voor een volgend project?

10.1 Discussie

Ter discussie staat het verloop van de onderzoeken en de methodes die hiervoor gebruikt zijn.

10.1.1 Onderzoek 1: Huidige en gewenste situaties

Het onderzoek naar de huidige en gewenste situaties is naar mijn mening goed verlopen. Het onderzoek heeft mij een duidelijk beeld gegeven over de stand van zaken met betrekking tot het huidige ScreenIT systeem en met name het cliëntportaal. Als het over kon worden gedaan, had ik wellicht een betere structuur op moeten zetten voor op welke manier de onderzoeksresultaten worden verwerkt. De schema's van de huidige en gewenste situaties schetsen voor mijn gevoel een goed beeld van het cliëntportaal op compile- en run-time.

10.1.2 Onderzoek 2: Samenwerking Kotlin met verschillende frameworks

In onderzoek 2 is de werking van Kotlin met verschillende front- en back-end frameworks uitgezocht. In dit onderzoek zijn de hoofd frameworks die gebruikt getest met Kotlin en de manier van implementatie omschreven. Voor bijvoorbeeld Bootstrap onderdelen waren een aantal extra stappen nodig (zoals het schrijven van *wrappers*). Hiervoor was extra onderzoek nodig in het framework. In de onderzoeksresultaten is uitgelegd op welke manier dit kan worden geïmplementeerd. Dit zie ik als een goed onderdeel van het totaalonderzoek, omdat dit in een eventueel tweede geval niet overnieuw hoeft worden onderzocht.

10.1.3 Onderzoek 3: Codedeling tussen front- en back-end

Onderzoek 3 gaat over op welke manier codedeling tussen front- en back-end mogelijk gemaakt kan worden met Kotlin. Uit onderzoek bleek dat dit alléén mogelijk was met Gradle, omdat de KotlinJS front-end alleen Gradle ondersteund. Ook voor het delen van code tussen JS en JVM modulen kan alleen Gradle worden gebruikt. Het is dus op het moment nog niet mogelijk om code te delen tussen de huidige maven back-end en de cliëntportaal-kotlin front-end. Om de onderzoeksresultaten toch waarde te geven zijn er twee externe Proof of Concepts gemaakt in de gevallen dat er alleen Maven of alleen Gradle wordt gebruikt. Deze proof of concepts tonen aan dat codedeling wel degelijk mogelijk is. Een verbeterpunt voor dit onderzoek zal zijn dat ik wellicht te lang heb geprobeerd om het alsnog werkend te krijgen in het cliëntportaal terwijl al eerder duidelijk was dat het niet mogelijk was met Gradle en Maven door elkaar.

10.1.4 Onderzoek 4: Sociale aspecten verandering naar Kotlin

Onderzoek 4 gaat over de sociale aspecten voor ontwikkelaars die komen kijken bij een verandering van programmeertaal. Hierbij is er een enquête gedeeld. Hierbij was het wellicht beter geweest als ik wat meer gesloten vragen met meerdere antwoorden had verwerkt, zodat de resultaten konden worden verwerkt als diagrammen. Op dit moment bestaat de enquête vooral uit open vragen. Voor de volgende keer moet ik wellicht wat meer aandringen op de ontwikkelaars om de enquête in te vullen, zodat ik wat meer reacties krijg. In dit geval waren het 6 reacties. Het feit dat de ontwikkelaars grotendeels vanuit huis werkten hielp niet mee met het aantal reacties naar boven brengen. Deze reacties bevatten echter wel waardevolle informatie waarom het wel of juist geen goed idee is om te migreren naar Kotlin en welke redenen dit zou kunnen hebben.

10.2 Reflectie proces

Gedurende de hele projectperiode is er gebruik gemaakt van een Scrum-manier van werken. Elke week was een sprint die duurde van maandag 12:00 tot maandag 12:00. Op de maandag is er telkens samen met de bedrijfsbegeleider een *sprint retrospective* gehouden en zijn er positieve en verbeterpunten van die week benoemd. Samenvattingen van elke sprint retrospective zijn te vinden in Bijlage 2.

Doordat de positieve en negatieve punten elke week zijn bijgehouden, is het goed terug te vinden welke leerprocessen ik heb meegemaakt. Gedurende het project heb ik gemerkt dat ik met het opstellen van opdrachten (tickets) aan het begin van een sprint, moeite had met het vaststellen van de grootte van de ticket. Hierdoor moesten bepaalde tickets worden doorgeschoven naar de volgende sprint of terug worden gezet op de backlog. Gedurende het leerproces is dit aanzienlijk verbeterd, door de opdracht/ticket onder te verdelen in kleinere, meer schaalbare deelopdrachten. Hierdoor was de grootte beter te bepalen en werden er minder tickets doorgeschoven.

Een ander deel van dit leerproces is het preciezer vaststellen van de Definition of Done bij elke ticket. Wanneer is een ticket echt af? Welke onderdelen moeten daarvoor gerealiseerd worden?

Qua technische kennis ben ik erg veel vooruitgegaan op het in kaart brengen van een groot project en het structureren van multi-module projecten door middel van een *build script*, zoals Maven of Gradle. Hier was ik voor de stage nog niet bekend mee. Ook was ik nog niet bekend met de fundamenten van het React + Redux framework.

11. Nawoord

Aan het eind van de projectperiode en na het schrijven van deze scriptie is mijn conclusie dat het een erg leerzaam en interessant project was. Ik ben erg blij met de kennis die ik tot me heb kunnen nemen en de nieuwe dingen die ik geleerd heb.

Gedurende de projectperiode zijn een aantal dingen aangepast ten opzichte van het plan van aanpak, maar dit heeft de voortgang van het project en het proces niet belemmerd. Een aantal voorbeelden zijn: de initiële planning van het uitvoeren van de onderzoeken en de requirements die zijn bijgeschaald op haalbaarheid. Door het uitvoeren van een enquête onder de ontwikkelaars heb ik een kijk gekregen op de denkwijze van de mensen die al langer in het vak zitten en hoe zij nieuwe kansen en veranderingen benaderen. Door mijn onderzoek naar de mogelijkheden konden de ontwikkelaars met meer kennis van zaken een mening vormen. De overige resultaten van de onderzoeken zijn erg interessant voor Topicus voor eventuele vervolgprojecten met de gebruikte frameworks.

De verandering van de oorspronkelijke planning gaf mij een minder prettig gevoel, omdat ik het liefst de oorspronkelijke planning aan wilde houden. Echter heeft dit de voortgang van het project en de hoofdonderzoeken niet belemmerd en is er een mooi advies gevormd aan de hand van de onderzoeksresultaten.

Ik wil Topicus en in het bijzonder Robert Spee (bedrijfsbegeleider) bedanken voor de goede begeleiding die ik heb genoten tijdens de projectperiode. Ook wil ik Dick Heijink als stagebegeleider bedanken voor de begeleiding vanuit Saxion.

Kevin Scholten

Deventer, april 2022

12. Literatuuroverzicht

	Bron
1	HBO-I. (2018, 18 januari). *Methoden toolkit*. HBO-I. Geraadpleegd op 14 feb 2022, van https://onderzoek.hbo-i.nl/index.php/Methoden_Toolkit_HBO-i#:~:text=Methoden%20Toolkit%20HBO%2Di&text=De%20toolkit%20biedt%20docenten%20en,deze%20kunnen%20updaten%20en%20onderhouden.
2	ReduxJS. (2015, 2 juni). *Redux: A predictable state cointainer for JS apps*. Redux. Geraadpleegd op 16 februari 2022, van https://redux.js.org/
3	MoSCoW. (2015). In Wikipedia. https://en.wikipedia.org/wiki/MoSCoW_method
4	Jetbrains. (2022, 9 februari). *Generate TS files when Kotlin is compiled to JS.* Geraadpleegd op 16 februari 2022, van https://youtrack.jetbrains.com/issue/KT-16604
5	Jetbrains. (2021a, 1 december). *Create a new project.* Geraadpleegd op 21 februari 2022, van https://www.jetbrains.com/help/idea/new-project-wizard.html
6	Jetbrains. (z.d.). *Kotlin docs.* Geraadpleegd op 23 november 2021, van https://kotlinlang.org/docs/home.html
7	Jetbrains. (2021b, 28 oktober). *Kotlin for JavaScript. KotlinLang.* Geraadpleegd op 23 november 2021, van https://kotlinlang.org/docs/js-overview.html
8	Gradle. (z.d.a). *Maven publish plugin.* Geraadpleegd op 23 februari 2022, van https://docs.gradle.org/current/userguide/publishing_maven.html
9	Mozilla. (2022, 21 februari). *Cross-Origin Resource Sharing (CORS).* Geraadpleegd op 17 maart 2022, van https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
10	Gradle. (z.d.b) *Gradle vs Maven comparison.* Geraadpleegd op 17 maart 2022, van https://gradle.org/maven-vs-gradle/
11	de Groot, B. (2021, 25 mei). *Kotlin makes Lombok obsolete*. Geraadpleegd op 18 maart 2022, van https://levelup.gitconnected.com/kotlin-makes-lombok-obsolete-9ed3318596cb
12	Singh, V. (2022, 1 maart). *Kotlin vs Java. Important differences you need to know.* Geraadpleegd op 29 maart 2022, van https://hackr.io/blog/kotlin-vs-java#:~:text=Kotlin%20is%20a%20statically%20typed,like%20it%20has%20for%20Java.
13	Jetbrains. (2017, 3 juni). *Kotlin Wrappers*. Geraadpleegd op 29 maart 2022, van https://github.com/JetBrains/kotlin-wrappers

12.1 Dependency bronnen

[1] Mapstruct: https://mapstruct.org/
[2] Axios: https://axios-http.com/docs/intro
[3] Create React Kotlin App: https://github.com/JetBrains/create-react-kotlin-app
[4] Yarn Package manager: https://yarnpkg.com/
[5] SpringBoot: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
[6] SpringBoot Security: https://spring.io/guides/gs/securing-web/
[7] SpringBoot Starter/ActiveMQ: https://www.geeksforgeeks.org/spring-boot-starters/
[8] Junit: https://junit.org/junit5/
[9] Junit Jupiter: https://developer.ibm.com/tutorials/j-introducing-junit5-part1-jupiter-api/
[10] PostgreSQL: https://www.postgresql.org/docs/
[11] Hibernate: https://hibernate.org/
[12] Keycloak: https://www.keycloak.org/documentation
[13] HyperSQL: http://hsqldb.org/
[14] JMS: https://www.oracle.com/technical-resources/articles/java/intro-java-message-service.html
[15] Mockito: https://site.mockito.org/
[16] ReactJS: https://reactjs.org/docs/getting-started.html
[17] SASS: https://sass-lang.com/
[18] React Redux: https://react-redux.js.org/
[19] Bootstrap: https://getbootstrap.com/docs/5.1/getting-started/introduction/
[20] JQuery: https://jquery.com/
[21] Jest: https://jestjs.io/
[22] Prop-types: https://www.npmjs.com/package/prop-types
[23] Formik: https://formik.org/docs/overview
[24] Yup: https://www.npmjs.com/package/yup
[25] DateFNS: https://date-fns.org/docs/Getting-Started
[26] TypeScript: https://www.typescriptlang.org/docs/
[27] Tiny Warning: https://www.npmjs.com/package/tiny-warning
[28] Popper.JS: https://popper.js.org/docs/v2/
[29] React Bootstrap Kotlin: https://bjoernmayer.github.io/kotlin-react-bootstrap/docs/getting-started/introduction
[30] Kotlin React CSS: https://mvnrepository.com/artifact/org.jetbrains.kotlin-wrappers/kotlin-react-css
[31] Lombok: https://projectlombok.org/

13. Versiebeheer

Versie	Datum	Omschrijving
0.1	15-11-2021	Initiële versie afstudeerdossier
0.5	14-02-2022	Tussentijdse evaluatie februari
0.9	11-03-2022	Pre-conceptversie afstudeerdossier (beoordeeld door Bedrijfsbegeleider)
1.0	20-03-2022	Conceptversie afstudeerdossier
2.0	03-04-2022	Definitieve versie afstudeerdossier

14. Bijlages

14.1 Document bijlages

Bijlage 1: Enquête sociale aspecten Kotlin

Deze link bevat de Google Survey enquête over de sociale aspecten van het gebruik van Kotlin.

Google Survey URL:
https://docs.google.com/forms/d/e/1FAIpQLSeYwGhhGk7gx__zdudIsLd-0M1uZ4_2tFMlWLXmQwQXjvSE7Q/viewform?usp=sf_link

Bijlage 2: Samenvattingen sprint/week retrospectives

Dit document bevat samenvattingen van de sprint retrospectives die aan het eind van elke sprint geschreven zijn. Per week is te volgen wat er goed ging en waar de verbeterpunten liggen.

Bestand: **Sprint retrospectives.pdf**

Bijlage 3: Definition of Done omschrijvingen

Dit document bevat omschrijvingen van alle Definition of Done's van alle Must Requirements.

Bestand: **Definition of Done omschrijvingen.pdf**

14.2 Proof of Concepts

PoC 1: Het Cliëntportaal in Kotlin

Dit bestand bevat de nieuwe versie van het cliëntportaal-systeem. Dit is zowel SpringBoot Kotlin/Java back-end als de React/KotlinJS front-end.

Bestand: **ProofOfConcept1.zip**

PoC 2: Codedeling tussen Maven modules

Dit bestand bevat een Proof of Concept voor het delen van code tussen Maven modules.

Bestand: **ProofOfConcept2.zip**

Github URL: **https://github.com/kevin-scholten/kotlin-codesharing-maven**

PoC 3: Codedeling tussen Gradle modules

Dit bestand bevat een Proof of Concept voor het delen van code tussen Gradle modules door middel van een multi-platform project.

Bestand: **ProofOfConcept3.zip**

Github URL: **https://github.com/kevin-scholten/kotlin-codesharing-gradle**