

Visualisatie biologische data

Realisatiedocument

Jef Ceulemans
Student Bachelor in de Toegepaste Informatica – Applicatieontwikkeling

Inhoudsopgave

1. INLEIDING	3
2. ANALYSE	4
2.1. Front-end framework	4
2.2. Code editor	6
2.3. Visualisation Libraries	8
3. UITWERKING	10
3.1. Achtergrondinformatie data	10
3.2. De Applicatie in het geheel	11
3.3. Data loading	13
3.3.1. Mijn bijdrage aan data loading	13
3.4. Side-panel	15
3.4.1. Mijn bijdrage aan de side-panel	15
3.5. Icicle Plot	18
3.5.1. Mijn bijdrage aan de Icicle Plot	19
3.6. Parallel Coordinates Plot	24
3.6.1. Mijn bijdrage aan de Parallel Coordinates Plot	24
3.7. Boxplot	26
3.7.1. Mijn bijdrage aan de boxplot	27
3.8. Spreadsheet	29
3.8.1. Mijn bijdrage aan de spreadsheet	30
3.9. ColorService	32
3.9.1. Mijn bijdrage aan de colorservice	32
4. BESLUIT	34
LITERATUURLIJST	35
BIJLAGEN	36

1. Inleiding

Mijn stage vond plaats bij LaBRI (Laboratoire Bordelais de Recherche en Informatique), een onderzoeksinstituut gespecialiseerd in computerwetenschappen. Deze stage richtte zich op de visualisatie van biologische data. Meer specifiek focuste het project op de analyse en vergelijking van de distributie van bacteriën, de zogenoemde metagenomische data (data die genetisch materiaal van volledige microbiële gemeenschappen bevat), die gevonden worden in bloedmonsters van patiënten. Deze analyse is van waarde voor het begrijpen van de respons van patiënten op behandelingen, zoals vaccins. Om zinvolle vergelijkingen mogelijk te maken, spelen de taxonomische relaties tussen bacteriën georganiseerd in een hiërarchische structuur een centrale rol.

De dataset bevat per patiënt waarden die corresponderen met verschillende taxonomische niveaus van bacteriële soorten. Gezamenlijke analyse kan worden uitgevoerd op diverse granulariteitsniveaus om te onderzoeken of groepen bacteriën geassocieerd zijn met specifieke patiëntstatussen of fenotypes. Binnen deze structuur bezit elke patiënt waarden op bepaalde taxonomische niveaus.

We werken in dit project met een vrij complexe dataset. In dit document leg ik uit hoe we hiermee aan de slag zijn gegaan. De bedoeling van onze applicatie is om gebruikers toe te laten zo'n dataset te uploaden en via interactieve grafieken en verschillende soorten weergaven nuttige inzichten te verkrijgen.

Nu er een eerste idee van het project is verkregen, geef ik in de volgende hoofdstukken een gedetailleerde toelichting. Ik bespreek onder andere welke tools we gebruikt hebben, hoe de dataset is opgebouwd en hoe de applicatie technisch in elkaar zit.

Tot slot nog dit: ik heb aan dit project samengewerkt met Stijn De Busser. Echter, in deze paper zal ik mij enkel focussen op het werk dat ik zelf heb uitgevoerd.

2. Analyse

Hieronder leg ik uit welke tools en technologieën we hebben onderzocht en vergeleken, en wat de argumenten waren voor de uiteindelijke keuzes die zijn gemaakt.

Als framework hebben we gekozen voor Angular, voor de visualisatie gebruiken we D3.js, en als editor Visual Studio Code.

Om dit verder toe te lichten, heb ik de verschillende opties geëvalueerd aan de hand van een weighted decision matrix.

2.1. Front-end framework

Voor het front-end framework hebben we een afweging gemaakt tussen React, Angular en Vue.

Weighted decision matrix front-end framework:

Criteria	Weight	Angular	React	Vue
Team experience	4	5	4	2
Ecosystem and extensibility	4	5	4	3
Performance	5	4	5	5
Features	5	5	5	4
Scalability	4	4	3	3
Debugging & testing	4	4	4	3
Community & support	3	4	5	3
Security	5	5	4	3
Total	170	154	145	113

Tabel 1: WDR front-end

Zoals hierboven te zien is, hebben we de belangrijkste criteria voor de keuze van het front-end framework opgeliist.

Hieronder licht ik toe waarom het ene framework op bepaalde criteria een hogere score heeft gekregen dan het andere.

- Team experience
 - Dit jaar heb ik zowel een cursus Angular als React gevolgd. Voor Angular hadden we een cursus van 6 studiepunten, terwijl de cursus voor React slechts 4 studiepunten omvatte. Daarnaast moesten we voor het vak Project 4.0 in groep een project realiseren, waarbij we ook gekozen hebben voor Angular als framework. Hierdoor heb ik duidelijk meer ervaring opgebouwd met Angular.
Van Vue hebben we enkel een klein deel behandeld binnen de PHP-lessen, waardoor de ervaring hiermee beperkt is.
Op basis van deze factoren krijgt Angular de hoogste score van 5 op het criterium Team experience.
- Ecosystem and extensibility
 - Angular is een volledig framework en biedt alles wat nodig is out-of-the-box. Het integreert vlekkeloos met andere technologieën zoals TypeScript, RxJS en Angular Material Design. Veel uitbreidingen zijn officieel ontwikkeld en worden onderhouden door een team van Google.

React daarentegen biedt meer vrijheid, maar vereist dat men zelf libraries kiest voor bijvoorbeeld routing, state management en formulieren. Hoewel React een groot ecosysteem heeft, is het minder geïntegreerd dan bij Angular.

Vue beschikt over een kleiner ecosysteem dan Angular en React, en er zijn ook minder officiële extensies beschikbaar in vergelijking met React

Op basis van deze factoren krijgt Angular een score van vijf op dit criterium.

- Performance

- Voor dit onderdeel hebben zowel React als Vue de hoogste score gekregen vanwege hun snelheid. Het is algemeen bekend dat Vue een snel en lichtgewicht framework is, wat de prestaties ten goede komt.

React biedt ook goede prestaties, mede dankzij de virtual DOM.

Angular is niet traag, maar is iets trager dan de andere twee. Dit komt voornamelijk doordat Angular een volledig framework is.

- Features

- Zoals hierboven al vermeld, is Angular een volledig framework. Het biedt veel ingebouwde functionaliteiten, zoals routing, dependency injection, formulieren, enzovoort. Het beschikt ook over zowel template-driven als reactive forms, wat krachtige tools zijn voor het werken met formulieren.

React wordt vaak een library genoemd, maar het biedt ook veel opties en verschillende features die benut kunnen worden. Het kan goed worden aangepast aan de behoeften van elk project.

Vue biedt minder ingebouwde features dan de andere twee.

Om deze redenen krijgen zowel React als Angular de hoogste score voor dit criterium.

- Scalability

- In alle drie de frameworks is scalability mogelijk, maar Angular maakt het makkelijker door de ingebouwde tooling en structuur die het biedt.

In React en Vue is schaalbaarheid ook mogelijk, maar de implementatie kan complexer zijn, vooral omdat er zelf meer beslissingen genomen moeten worden over de architectuur en tools.

Daarom krijgt Angular de hoogste score voor scalability.

- Debugging & testing

- Voor unit testen heeft Angular uitstekende tools zoals Jasmine en Karma. Deze tools maken het testen van code en het debuggen ervan veel aangenamer.

In React wordt het ook gemakkelijk gemaakt, aangezien er verschillende libraries geïnstalleerd kunnen worden voor testing, bijvoorbeeld Jest en Enzyme.

Vue, aan de andere kant, biedt Vue DevTools, Jest en Vue Test Utils. Deze zijn ook goede test- en debuggingtools, net als de andere, maar ze zijn minder uitgebreid dan de tools van Angular en React.

Hierdoor krijgen React en Angular de hoogste score.

- Community & support

- React is één van de grootste en populairste front-end technologieën ter wereld. Het heeft een enorm aantal gebruikers en een grotere ontwikkelaarsgemeenschap. Ook zijn er veel online resources beschikbaar, zoals tutorials, blogs, enz.

Angular, aan de andere kant, heeft ook een grote community en een uitgebreide officiële documentatie. Alleen is de community en de hulp die online te vinden is niet zo groot als die van React. Een reden hiervoor kan zijn dat Angular complexer is dan React.

Vue, aan de andere kant, heeft de kleinste community van deze drie. Er zijn ook minder tutorials te vinden dan van de andere twee, hoewel het een uitstekende officiële documentatie heeft. Aangezien Vue minder vaak wordt gebruikt, is het logisch dat de community en support kleiner zijn dan die van de andere twee.

Daarom krijgt React hier het hoogste punt, aangezien de community duidelijk groter is.

- Security
 - Angular biedt uitstekende ingebouwde bescherming tegen XSS, dependency management, bescherming tegen injection-aanvallen en nog veel meer. De andere frameworks bieden ook beveiligingsopties tegen dit soort aanvallen, maar deze zijn niet zo makkelijk te implementeren als bij Angular. Vue krijgt hier een iets lagere score omdat de beveiligingsmaatregelen iets minder robuust zijn dan die van Angular. Om deze reden krijgt Angular het hoogste punt vanwege het gebruiksgemak.

Samenvattend, de combinatie van onze ervaring, het complete en geïntegreerde ecosysteem, de rijke set aan features, de schaalbaarheid en de robuuste beveiligingsprotocollen maakte Angular de optimale keuze voor de ontwikkeling van deze applicatie.

2.2. Code editor

Voor de code editor hebben we een vergelijking gemaakt tussen Visual Studio Code, Eclipse en IntelliJ IDEA.

Weighted decision matrix code editor

<u>Criteria</u>	<u>Weight</u>	VSCode	Eclipse	IntelliJ IDEA
<i>Ease of use</i>	5	5	3	4
<i>Performance</i>	5	5	3	4
<i>Features</i>	5	4	4	5
<i>Customizability</i>	4	5	4	4
<i>Cost</i>	4	5	5	4
<i>Integrations</i>	4	5	3	5
<i>Debugging</i>	4	4	3	5
<i>Community and support</i>	3	5	4	4
Total	170	161	122	149

Tabel 2: WDR code editor

Zoals hierboven te zien is, hebben we de belangrijkste criteria voor de keuze van de code editor opgeliist. Hieronder licht ik toe waarom de ene code editor op bepaalde criteria een hogere score krijgt dan de andere.

- Ease of use
 - In de opleiding hebben we het meeste al gewerkt met de Visual Studio Code editor. Hierdoor ben ik al zeer vertrouwd met deze editor. Ook staat Visual Studio Code bekend als lichtgewicht, makkelijk te installeren en biedt het een fijne gebruikerservaring. We hebben ook kort met IntelliJ IDEA mogen werken tijdens onze opleiding, en deze staat ook bekend om zijn georganiseerde interface. Eclipse heeft een steilere leercurve dan de andere twee door zijn complexere interface en configuraties. Door deze bevindingen heeft VSCode de hoogste score gekregen, voornamelijk omdat we hier meer mee hebben gewerkt dan met IntelliJ IDEA.
- Performance
 - Doordat VSCode lichtgewicht is, start deze heel snel op en gebruikt het minder systeembbronnen. Hierdoor zou deze zelfs op oudere of minder krachtige machines op een normale manier kunnen werken. Aan de andere kant staat Eclipse bekend om zijn hogere resourcegebruik, wat leidt tot langere opstarttijden op minder krachtige machines.

IntelliJ IDEA is dan weer tussen de twee in, het is sneller dan Eclipse, maar trager dan VSCode. Dit komt door de uitgebreide functionaliteit van IntelliJ IDEA.
Om deze redenen krijgt VSCode de maximale score van vijf.

- Features
 - VSCode heeft een breed scala aan extensies en ingebouwde functionaliteiten. Voor de meer geavanceerde toepassingen moet echter nog worden gezocht naar plug-ins, deze zijn vaak meer taalspecifiek.
Eclipse is dan weer gespecialiseerd in Java-ontwikkeling, maar kan worden uitgebreid naar Angular via plug-ins, waarna het goed werkt. Over het algemeen bevat de editor veel features, niet per se voor Angular, maar wel voor andere talen. Hierdoor krijgt Eclipse hier alsnog een hoge score.
IntelliJ IDEA biedt ingebouwde ondersteuning voor Angular, met features zoals dependency injection-analyse, codegeneratie, enzovoort.
Doordat dit ingebouwd is en veel features standaard aanwezig zijn, krijgt IntelliJ IDEA de hoogste score hier.
- Customizability
 - VSCode is extreem aanpasbaar aan de eigen noden. Zowel het uiterlijk en de workflow kunnen volledig naar wens worden aangepast met behulp van instellingen en diverse extensies.
Eclipse biedt ook veel aanpassingsmogelijkheden via plug-ins en configuraties in de instellingen, maar het is minder gebruiksvriendelijk in vergelijking met VSCode.
IntelliJ IDEA kan ook wel worden aangepast aan de voorkeur, alleen minder diep in detail zoals bij de andere. Het richt zich meer op de out-of-the-box-ervaring.
Om deze redenen krijgt VSCode de maximumscore.
- Cost
 - VSCode is helemaal gratis en open source, waardoor het zeer toegankelijk is voor iedereen.
Eclipse is ook gratis en open source.
IntelliJ IDEA, langs de andere kant, is wel betalend. Ze hebben ook een gratis versie, maar die mist de geavanceerde functies. Voor ons als studenten maakt dit momenteel niet zoveel uit, want wij kunnen de Ultimate Edition gratis krijgen.
Ondanks dat geef ik toch VSCode en Eclipse de hoogste score.
- Integrations
 - Zoals hierboven al te lezen was, is er voor Angular gekozen. Nu wordt onderzocht hoe goed de editor werkt met dit framework. Als we kijken naar VSCode, dan zien we dat er uitstekende ondersteuning is voor Angular via verschillende extensies. Het werkt ook naadloos samen met Git en de testing frameworks van Angular.
Eclipse kan ook gebruikt worden, maar dit is wel wat lastiger, want deze editor is vooral gericht op Java-ontwikkeling. Hoewel er een plugin beschikbaar is voor Angular, is deze minder efficiënt en gebruiksvriendelijk dan bij andere editors.
IntelliJ IDEA is ook perfect geschikt voor ons framework. Het heeft ingebouwde ondersteuning voor Angular, met verschillende soorten integraties. Net zoals VSCode heeft het een naadloze integratie met de debuggingtools.
Om deze redenen krijgen VSCode en IntelliJ IDEA de maximale score.
- Debugging
 - Zoals hierboven al kort vermeld was, werken de debuggingtools zowel voor VSCode als voor IntelliJ IDEA goed. Bij VSCode is er ondersteuning via extensies, zoals de Angular Language Service. Debugging is zeker mogelijk, alleen is er wel extra configuratie nodig.
Voor Eclipse is debugging ook mogelijk, maar het is minder uitgebreid en gebruiksvriendelijk dan bij de andere editors. Hiervoor kan de plugin CodeMix worden gebruikt.

IntelliJ IDEA biedt eigenlijk de beste debugervaring. Er zijn hier amper configuraties nodig, het werkt meteen out of the box, wat de ervaring een stuk makkelijker maakt. Hierdoor krijgt IntelliJ IDEA de hoogste score op dit onderdeel.

- Community and support
 - VSCode heeft een enorme en actieve community, met uitgebreide documentatie, allerlei soorten extensies en een grote hoeveelheid hulpmiddelen. Eclipse heeft ook een actieve community, alleen is deze logischerwijs kleiner dan die van VSCode. Hierdoor zullen er minder extensies en minder onlinehulp beschikbaar zijn. IntelliJ IDEA heeft ook een sterke community en uitstekende documentatie, maar omdat de uitgebreide versie betrekend is, zal de support voor ons framework iets minder toegankelijk zijn. Bijna alle handige extensies voor Angular zitten namelijk in de betrekende versie. Daarom kom ik tot de duidelijke conclusie dat VSCode hier de meeste punten verdient.

Concluderend, de combinatie van gebruiksgemak, snelle prestaties, uitgebreide aanpasbaarheid, kosteloosheid en sterke community-ondersteuning maakte Visual Studio Code de optimale keuze voor de ontwikkeling van deze applicatie.

2.3. Visualisation Libraries

Voor visualisatielibraries hadden onze stagebegeleiders twee opties naar voren geschoven: D3.js en Circos. Om tot een weloverwogen keuze te komen, hebben we deze twee libraries vergeleken.

Weighted decision matrix visualisation libraries:

<u>Criteria</u>	<u>Weight</u>	D3.js	Circos
Ease of use	4	2	3
Customizability	5	5	3
Documentation	4	4	3
Integration ability	5	5	3
Performance	5	5	4
Community & support	3	5	2
Total	130	114	80

Tabel 3: WDR visualisation libraries

Zoals hierboven te zien is, hebben we de belangrijkste criteria voor de keuze van een visualisatielibrary opgeliist.

Hieronder licht ik toe waarom de ene library een hoger cijfer krijgt dan de andere op elk criterium.

- Ease of use
 - D3.js is een zeer krachtige tool, maar hierdoor is de leercurve ook heel steil. Het vereist een goed begrip van JavaScript en SVG's. Omdat ik deze library nog nooit gebruikt heb, zal het voor mij in het begin lastiger zijn om deze goed te begrijpen. Voor Circos geldt eigenlijk hetzelfde, alleen is deze iets eenvoudiger voor cirkelvormige visualisaties. Hierdoor krijgt Circos een iets hoger punt dan D3.js, omdat de leercurve minder steil is.
- Customizability
 - Zoals hierboven al vermeld werd, is Circos enkel geschikt voor cirkelvormige visualisaties. Bovendien biedt het, buiten het standaard visualisatieformaat, beperkte aanpassingsmogelijkheden. Bij D3.js is dit veel beter, aangezien volledige controle over de visualisatie mogelijk is: de vorm, kleur, stijl, interactie, enzovoort, kunnen zelf bepaald worden. Om die reden krijgt D3.js hier de maximale score, aangezien alles volledig zelf kan worden aangepast, terwijl men bij Circos vastzit aan de cirkelvormige structuur.

- Documentation
 - Voor D3.js is er meer documentatie beschikbaar, wat logisch is aangezien D3.js verschillende vormen van visualisatie ondersteunt. Daardoor is er vanzelfsprekend ook meer documentatie nodig. De documentatie die ik tot nu toe bekeken heb, is zeer duidelijk en goed opgebouwd.
Bij Circos is er minder documentatie beschikbaar en deze is ook minder uitgebreid dan die van D3.js.
Dit is dan ook de reden waarom D3.js op dit vlak een hogere score krijgt dan Circos.
- Integration ability
 - Aangezien D3.js een JavaScript-library is, zal deze perfect integreren met het Angular-framework. Mede dankzij de ondersteuning van Angular voor externe bibliotheken zal dit geen enkel probleem vormen. Bovendien biedt D3.js mogelijkheden om DOM-elementen te manipuleren, wat goed past binnen de componentgebaseerde architectuur van Angular. Circos werd oorspronkelijk ontworpen als een standalone tool voor cirkelvormige visualisaties. Hierdoor is het minder geschikt om te integreren in een webapplicatie. Daarnaast is de JavaScript-ondersteuning bij Circos beperkt. Het is dus vooral veel lastiger en meer werk om Circos te integreren in een Angular werkomgeving.
Om deze redenen krijgt D3.js hier de maximale score van vijf.
- Performance
 - D3.js werkt rechtstreeks met SVG's en DOM-elementen, wat zorgt voor een efficiënte rendering van grafieken. Daarnaast is D3.js ontworpen om grote hoeveelheden data te verwerken en te visualiseren.
Circos biedt eveneens goede prestaties voor zijn specifieke doel, namelijk cirkelvormige visualisaties. Echter, Circos is minder geoptimaliseerd dan D3.js voor het verwerken van grote datasets, zeker wanneer deze dynamisch zijn.
Om die reden krijgt D3.js hier de hoogste score.
- Community & support
 - De community van D3.js is zeer groot, het is een van de meest actieve communities in de wereld van data-visualisatie. Hierdoor zijn er veel forums, video's en andere vormen van hulp beschikbaar.
Bij Circos is de community en de support veel kleiner, wat voornamelijk komt doordat Circos zich richt op één specifieke doelgroep. Hierdoor is de community kleiner dan die van D3.js en is er minder online hulp beschikbaar.
Om deze redenen krijgt D3.js hier de hoogste score.

Tijdens de analyse werd het al snel duidelijk dat de functionaliteit van Circos beperkt is tot circulaire visualisaties, wat niet voldeed aan de eisen van dit project. Daarom is uiteindelijk gekozen voor D3.js.

3. Uitwerking

In dit deel van de paper ga ik toelichten hoe de applicatie in zijn totaliteit werkt, welke delen ik specifiek heb gemaakt. Ik zal ook achtergrondinformatie geven over de dataset die wordt gebruikt en hoe deze in elkaar steekt.

3.1. Achtergrondinformatie data

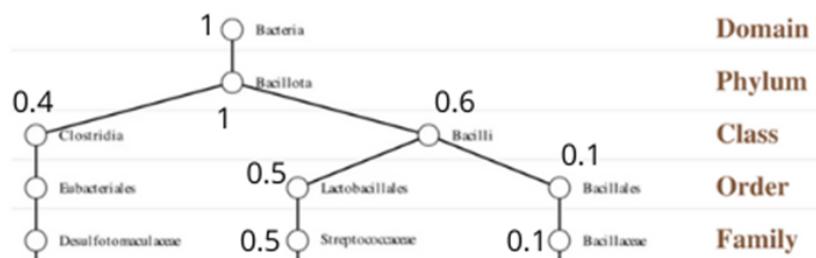
Het is belangrijk om inzicht te krijgen in de achtergrond van de data waarmee we hebben gewerkt, aangezien deze data centraal staat in de werking van onze applicatie. Ik begin met een algemene uitleg over het type data, gevolgd door een toelichting op de specifieke dataset die wij hebben gebruikt.

De data zijn afkomstig uit DNA-sequencing, een technologie waarmee genetisch materiaal van organismen kan worden uitgelezen. Binnen dit project gaat het specifiek om metagenomische sequencing, waarbij al het DNA in een biologisch monster (bijvoorbeeld bloed) wordt geanalyseerd. Deze techniek maakt het mogelijk om een volledig beeld te krijgen van de microbiële samenstelling van het monster. Metagenomische sequencing wordt veel gebruikt in microbiologisch onderzoek.

Kingdom	Eukaryota	Bacteria	De datasets die hieruit voortkomen, hebben een hiërarchische structuur. Zoals te zien is op de afbeelding, worden de taxa hiërarchisch weergegeven. Een taxon (meervoud: taxa) is een algemene term voor een taxonomische eenheid of groep op elk niveau in deze hiërarchische classificatie, beginnend bij het hoogste niveau (bijvoorbeeld Domain of Kingdom) en doorlopend tot het laagste niveau: de soort (Species). Voor bacteriën (Kingdom) kan een species bijvoorbeeld een specifieke <i>Salmonella</i> -variant zijn. Bovendien is er per monster microbiële data beschikbaar per taxonomische eigenschap. Een veelgebruikte methode voor dit soort analyses is 16S rRNA-sequencing.
Phylum	Chordata	Pseudomonadota	
Class	Mammalia	Gammaproteobacteria	
Order	Primates	Enterobacteriales	
Family	Hominidae	Enterobacteriaceae	
Genus	Homo	Salmonella	
Species	Homo sapiens	Salmonella enterica	

Figuur 2: Hiërarchische structuur

Zoals vermeld, wordt microbiële data geordend volgens taxonomische classificaties. Dit leidt tot een boomstructuur met ouder-kindrelaties. Data op hogere niveaus worden verkregen door aggregatie van de onderliggende niveaus. Om bijvoorbeeld de waarde op het Domain (of Bacteria- niveau te berekenen, moeten alle onderliggende taxa worden opgeteld. Dit principe is visueel weergegeven in de figuur hieronder.



Figuur 2: Hiërarchische structuur met aggregatie

De verkregen data zijn geëxtraheerd uit bloedmonsters van patiënten gediagnosticeerd met HIV (Human Immunodeficiency Virus) en die CART ondergaan (Combination Antiretroviral Therapy). Deze patiënten zijn onderverdeeld in twee groepen: Responders, die reageren op de behandeling, en Non-Responders, die dat niet doen.

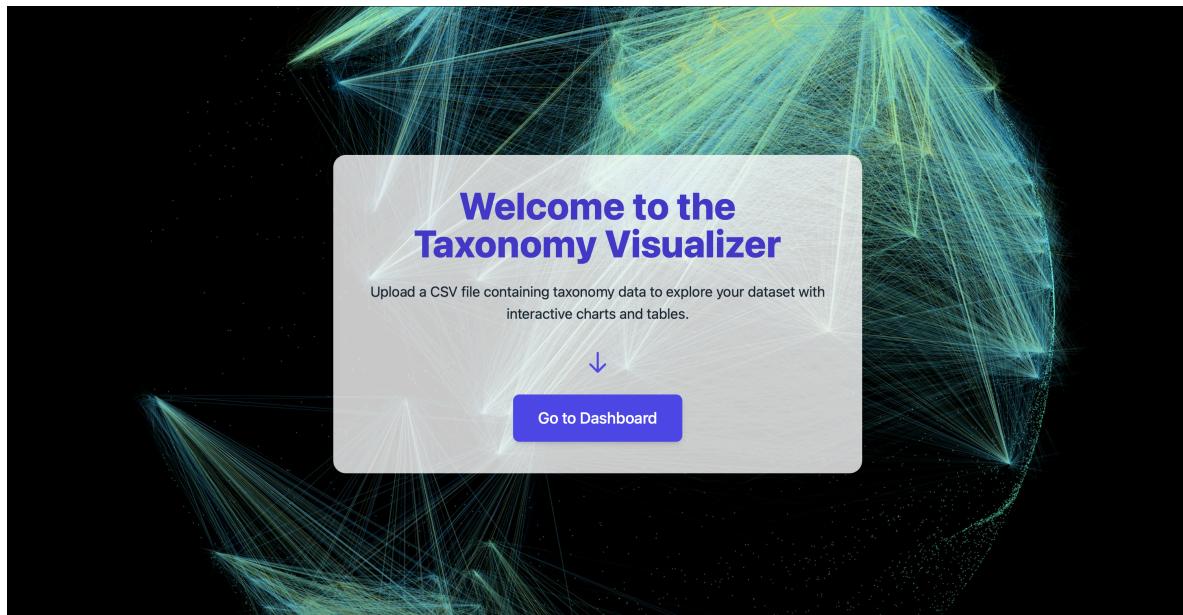
Zoals verwacht is een dergelijke dataset tamelijk complex. Dit komt vooral door de hiërarchische structuur van de taxonomische data die gevisualiseerd moet worden en de interactieve analyse van de data moet begeleiden. Hierdoor kunnen niet snel conclusies worden getrokken en ontbreekt er visuele ondersteuning.

Daarom is deze visualisatietool ontwikkeld. Hiermee kan deze lastige CSV-dataset met taxonomische data worden omgevormd naar duidelijke, interactieve visualisaties. De verschillende hiërarchische niveaus zullen dynamisch in plots te zien zijn. Het einddoel is dat de distributie van een specifiek taxon tussen verschillende groepen van samples kan worden vergeleken.

3.2. De Applicatie in het geheel

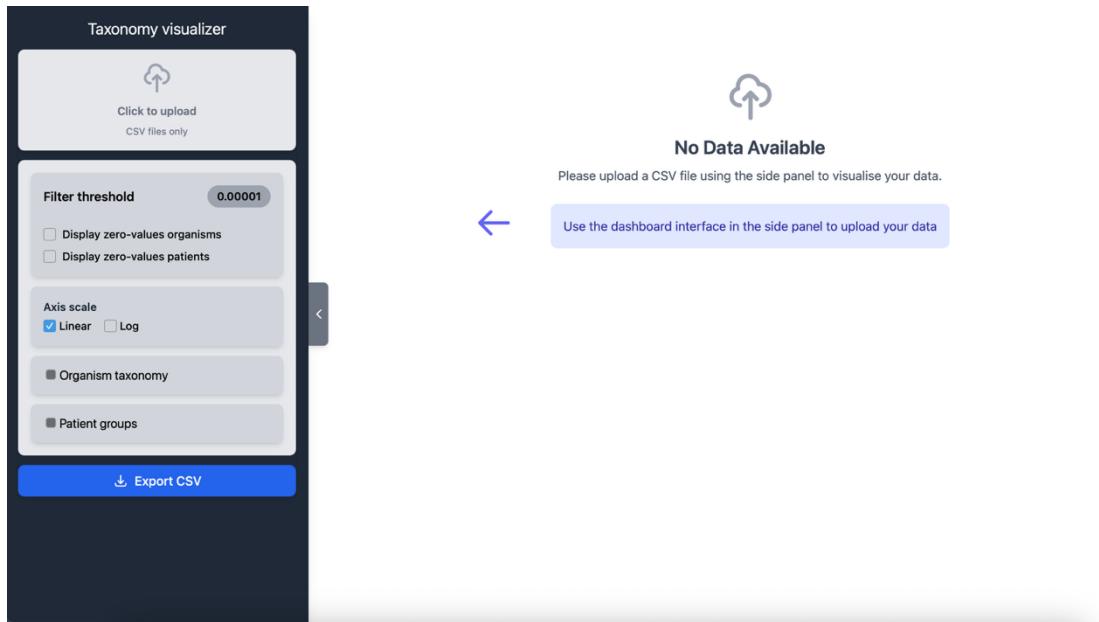
Zoals eerder al werd vermeld, is deze applicatie het resultaat van een samenwerking met een andere student. In dit gedeelte zal ik de werking van de totale applicatie uitleggen, zonder te diep op de details in te gaan. Het doel is om jullie een goed overzicht te geven van hoe de applicatie werkt. Vervolgens zal ik per hoofdonderdeel dieper ingaan op de specifieke onderdelen die ik heb ontwikkeld, zodat duidelijk wordt wat mijn precieze bijdrage is geweest.

Wanneer de applicatie wordt geopend, verschijnt er een startscherm waarop eventueel hulp beschikbaar is over het gebruik van de applicatie. Er is ook een startknop: als hierop wordt gedrukt, wordt men naar het dashboard gebracht.



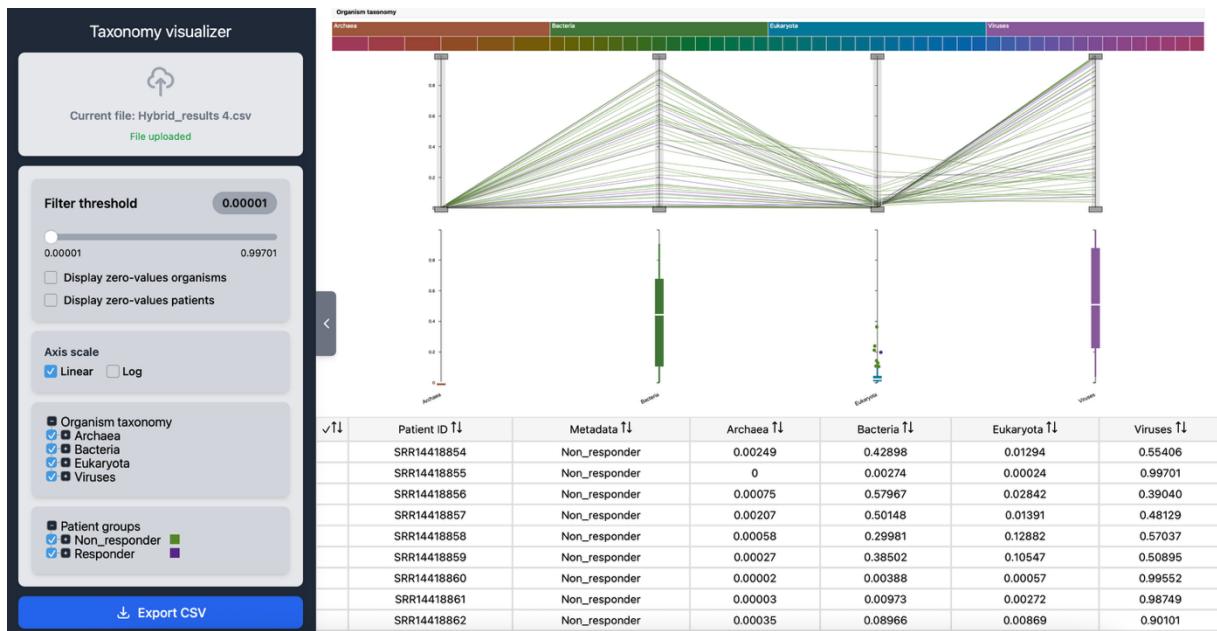
Figuur 3: Startscherms applicatie

Als daarop wordt gedrukt, komt men op het volgende scherm terecht:



Figuur 4: Startscherm selecteer file

De gebruiker begint met het uploaden van een CSV-bestand met microbiële data. Dit gebeurt door links bovenaan te klikken op "Click to upload", en vervolgens het gewenste CSV-bestand te selecteren. Nadat dit is gedaan, verschijnt het volgende scherm:



Figuur 5: Volledige applicatieweergave

De volledige applicatieweergave is nu zichtbaar in Figuur 5. Hieronder ga ik dieper in op elk component dat getoond wordt. Dit betreft de Icicle Plot, de Parallel Coordinates plot, de boxplot, de spreadsheet en de side-panel.

Om alvast een tipje van de sluier op te lichten, de applicatie is interactief. Dit betekent dat de weergave zich voortdurend aanpast aan jouw behoeften. Er zijn meer dan genoeg filteropties beschikbaar om jou de juiste en waardevolle inzichten te bieden.

3.3. Data loading

Data loading is iets wat niet visueel zichtbaar is, maar wat achter de schermen gebeurt. Het is echter zeer belangrijk voor onze applicatie, omdat de manier waarop de data geordend wordt in de dataservicefile een grote invloed heeft op hoe eenvoudig we in de verschillende componenten de data kunnen tonen op de diverse manieren die nodig zijn.

3.3.1. Mijn bijdrage aan data loading

Ik heb ervoor gezorgd dat de patiënten ook dynamisch worden "opgeslagen". Momenteel bestaat de dataset uit twee patiëntengroepen, ingedeeld op basis van hun respons op de behandeling (gemeten aan de hand van de immuunrespons), namelijk de Responders en de Non-Responders. Ik heb het nu zo opgezet dat het niet uitmaakt welke naam deze patiëntengroepen krijgen. Als een groep bijvoorbeeld effectgroep heet in plaats van Responders, dan zal effectgroep correct weergegeven worden in de applicatie. Dit is mogelijk doordat de structuur nu dynamisch werkt.

Zoals hieronder zichtbaar, heeft de interface vijf variabelen:

```
export interface Patient {  
    name: string;  
    path: string;  
    children?: Patient[];  
    taxonomyValues: Map<TaxonomyNode, value>;  
    selected?: boolean;  
}
```

Figuur 6: Interface Patient

- name
 - Houdt de naam van de node bij. In onze huidige dataset is dit bijvoorbeeld Responder voor de bovenste laag, en de patiënt-ID voor de child nodes.
- path
 - Het volledige pad in de boomstructuur, bijvoorbeeld Responder/PatientID.
- children
 - Wordt gebruikt voor child nodes, bijvoorbeeld de patiënten binnen een respondergroep. Deze variabele is alleen aanwezig bij groepsnodes.
- taxonomyValues
 - De waarden per taxonomynode voor deze patiënt tonen welke waardes de patiënt had voor specifieke organismen.
- selected
 - Houdt bij of de patiënt geselecteerd is in de UI. Dit wordt voornamelijk gebruikt voor filtering.

Ook heb ik een andere interface voor patiënten geïmplementeerd, namelijk RawPatient. De reden dat we deze hebben aangemaakt, was om het Single Responsibility Principle toe te passen, en daardoor het voor ons makkelijker te maken. Hierdoor heeft de datastructuur slechts één reden om te veranderen. Patiënt stelt een node voor in de patiënt-hiërarchieboom voor de visualisatie terwijl RawPatient de ruwe data voorstelt die we van de databron (een CSV-bestand) krijgen. Deze twee interfaces staan wel met elkaar in verbinding. De dataflow gaat van de CSV-data naar RawPatient-objecten en vervolgens naar een patiënt-hiërarchie. Beide datastructuren worden gesynchroniseerd gehouden.

```

export interface RawPatient {
  id: string;
  taxonomyValues: Map<TaxonomyNode, value>;
  selected?: boolean;
  responderStatus?: string;
}

```

Figuur 7: Interface RawPatient

- id
 - Deze dient voor de unieke identificatie van de patiënt.
- taxonomyValues
 - De waarden per taxonomynode voor deze patiënt. Hiermee kan worden gezien welke waardes een patiënt had voor specifieke organismen.
- selected
 - Houdt bij of de patiënt geselecteerd is in de UI. Dit wordt voornamelijk gebruikt voor filtering.
- responderStatus
 - Dit houdt de tekstuele status bij van de patiënt. Bijvoorbeeld als hij Responder is zal hier Responder komen te staan. Deze variabele wordt gebruikt voor groepering in de visualisaties.

In de dataservicefile zitten ook andere interessante functies die ik gemaakt heb. Deze zijn logischerwijs voor patiënten. Hieronder worden de meest interessante functionaliteiten toegelicht:

- BuildPatientHierarchy
 - Deze functie zorgt ervoor dat de platte lijst van de RawPatient-objecten naar een hiërarchische boomstructuur wordt getransformeerd.
 - Deze maakt een map aan die als sleutel de responderStatus gebruikt en als waarde een array van patiënten met die status. Voorbeeld ResponderGroups:

```

Map (2)
sleutel "Non_responder"
waarde Array (44)
  0 Object
    id: "SRR14418854"
    responderStatus: "Non_responder"
    selected: true
    taxonomyValues: Map {name: "Archaea", path: "Archaea", children: Array, aggregatedValue: 0.092188932173, selected: true} => 0.00249209727...
  1 Object prototype
  
```

Figuur 8: Structuur voorbeeld ResponderGroups (zonder path)

- Dan wordt er een hiërarchische boomstructuur opgebouwd. Dit zal als resultaat een tweelaagse hiërarchie (gedefinieerd door het 'Path' veld) geven waarin patiënten zijn gegroepeerd op basis van hun responderstatus.

```

Object
  name: "SRR14418882"
  path: "Non_responder|SRR14418882"
  selected: true
  taxonomyValues: Map {name: "Archaea", path: "Archaea", children: Array, aggregatedValue: 0.092188932173, selected: true} => 0.002693266061, {name: "Eu...
  
```

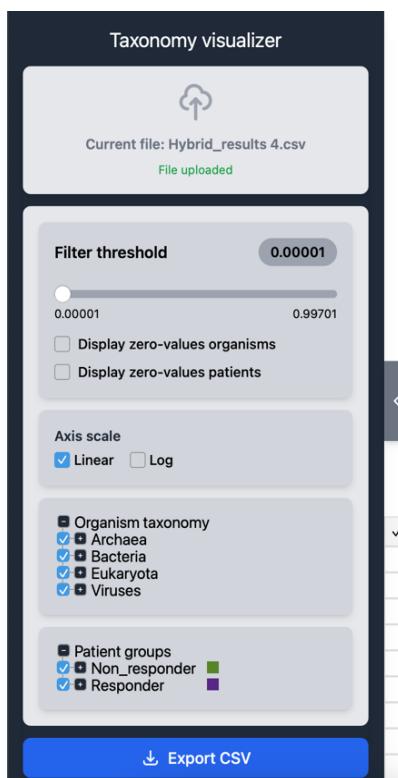
Figuur 9: Structuur voorbeeld tweelaagse hiërarchie (met path)

- updateSinglePatientSelection
 - Deze werkt de selectiestatus bij van een individuele patiënt. Deze logica bestond al, maar ik heb hieraan de logica bijgevoegd dat de selectiestatus in zowel de hiërarchische als de platte lijst van de patiënten wordt bijgewerkt.

- updatePatientSelection
 - Deze werkt net zoals de functie hierboven, alleen voor meerdere patiënten tegelijk. Deze zorgt op een efficiënte manier voor het bijwerken van meerdere patiënten. Mijn bijdrage hierbij was ook dit werkend te krijgen voor beide interfaces.

3.4. Side-panel

De side-panel biedt de mogelijkheid om via interactie te bepalen welke data in de hoofdweergave wordt gevisualiseerd. De getoonde informatie kan op de volgende manieren worden ingesteld:



Figuur 10: Foto side-panel

- **File uploader:** Hiermee kan een CSV-bestand met microbiële distributiedata worden geüpload.
- **Filter threshold:** Hier kan de gebruiker een drempelwaarde instellen om data met een bepaalde waarde uit te filteren. De maximale waarde wordt altijd berekend op basis van de maximale waarde die beschikbaar is in de specifieke weergave waarin de gebruiker zich bevindt. Er kan ook voor gekozen worden om de organismen en/of de patiënten die zijn weggefilterd terug te halen, deze zullen dan echter alleen nulwaarden tonen.
- **Axis scale:** Dit is voor de boxplot en Parallel Coordinates plot. De schaal van de as kan worden aangepast naar lineair of logaritmisch. Een logaritmische schaal geeft waarden weer op basis van verhoudingen (bijvoorbeeld maal 10) in plaats van op absolute verschillen. Dit is vooral handig als er sprake is van data die sterk varieert of uitschieters bevat.
- **Organism taxonomy:** De hiërarchische structuur van de organisme-taxonomie is hier zichtbaar. De selectievakjes kunnen worden geopend, waarna specifiek kan worden gekozen welke taxa in de hoofdweergave getoond moeten worden. Bijvoorbeeld: als *Archaea* niet getoond hoeft te worden, vinkt de gebruiker deze uit.
- **Patient groups:** Deze logica werkt hetzelfde als hierboven, maar dan voor patiënten. Wanneer een groep is gekozen en geselecteerd, worden alle patiënten die tot die groep behoren weergegeven. Ook hier kan de gebruiker selecteren, bijvoorbeeld: als alleen de Responder-patiënten getoond moeten worden, vinkt de gebruiker de Non-Responder-groep uit.
- **Export CSV:** Hiermee kan de data van de huidige weergave worden geëxporteerd naar een CSV-bestand. Er is dan nog de keuze om alleen de selectie te exporteren of alles.

3.4.1. Mijn bijdrage aan de side-panel

Ik heb het volledige ontwerp van de side-panel gemaakt, hiervoor heb ik Tailwind CSS gebruikt. Zoals op de foto zichtbaar is, zijn er drie panelen in de side-panel, waarbij het tweede deel verder is onderverdeeld. Dit is zo gedaan dat het middelste paneel verantwoordelijk is voor het veranderen van de hoofdweergave. Deze worden dan weer visueel onderscheiden van elkaar door een andere rechthoek eromheen te hebben.

Ook de uitklapbare selectievakjes-boomstructuur (checkboxes tree) is door mij gemaakt, zowel voor de organismetaxonomie als voor de patiëntengroepen. Hiervoor heb ik twee ViewChildren aangemaakt: taxonomyPanel en patientsPanel. Een ViewChild is hier nodig, omdat voor het renderen van D3.js-visualisaties directe toegang tot het DOM (Document Object Model) noodzakelijk is. Door ViewChild in

Angular te gebruiken, kan op een Angular-specifieke manier directe toegang tot de DOM-elementen worden verkregen.

Zoals net vermeld, hadden we een taxonomyPanel en een patientsPanel. Hierin zaten natuurlijk andere functies die voor het renderen van deze panelen zorgden. Voor de taxonomie heb ik een renderTaxonomyPanel gemaakt. Hier is een beetje achtergrondinformatie zeker gepast: ik heb een nieuwe interfacefile gemaakt met een HierarchyNode en een ExtendedHierarchyNode-interface. Dit deed ik met als hoofdreden dat ik scheiding wilde tussen het datamodel en het visualisatiemodel, mede omdat het hier om een complexere visualisatie gaat, namelijk een hiërarchische boomstructuur.

```
export interface HierarchyNode {  
    name: string;  
    children?: HierarchyNode[];  
    selected?: boolean;  
    aggregatedValue?: number  
}
```

Figuur 11: Interface HierarchyNode

Het verschil met TaxonomyNode is dat het path is weggelaten, omdat we dat niet meer nodig hebben. De eigenschap selected is toegevoegd, omdat we moeten bijhouden welke nodes door de gebruiker zijn geselecteerd (filtering op basis van de selectie). Ik heb dit zo gedaan, omdat dit exact is wat ik nodig heb voor de visualisatie. Een nieuwe interface creëren is hierdoor meer gestroomlijnd voor de specifieke use case in de side-panel

```
export interface ExtendedHierarchyNode extends d3.HierarchyNode<HierarchyNode> {  
    index: number;  
    _children?: d3.HierarchyNode<HierarchyNode>[];  
}
```

Figuur 12: Interface ExtendedHierarchyNode

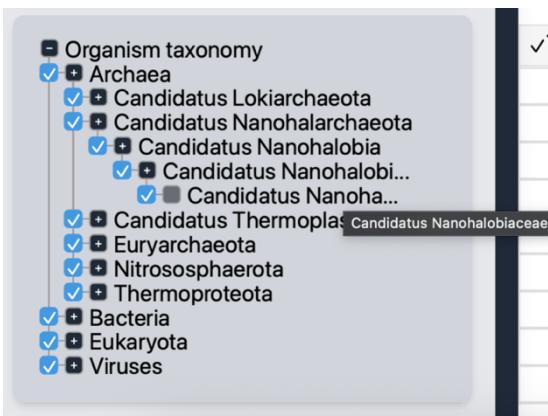
De ExtendedHierarchyNode breidt de standaard D3 HierarchyNode uit met extra eigenschappen, deze zullen specifiek nodig zijn voor de visualisatielogica.

- index
 - Dit nummer bepaalt de positie van de node in de visualisatie.
- _children
 - Dit dient als een cache voor de children van een node wanneer deze is ingeklappt.

Nu deze achtergrondinformatie over de renderTaxonomyPanel-functie bekend is, is het goed om te weten dat deze verantwoordelijk is voor de belangrijkste logica van het taxonomische paneel. De functie creëert een interactieve visualisatie met selecteerbare nodes, die ook in- en uitgeklapt kunnen worden. Hierbij is uiteraard ook hiërarchisch gedrag geïmplementeerd.

Als ik hieronder bijvoorbeeld de eerste opengeklapte child van *Archea* (oftewel *Candidatus Nanohalarchaeota*) zou deselecteren, dan zullen alle nodes die onder deze child vallen, eveneens gedeselecteerd worden. Dit werkt ook in de omgekeerde richting.

Bovendien is er aandacht besteed aan de kleine details, zoals de afhandeling van namen die buiten de beschikbare ruimte vallen. Deze worden afgekort met ‘...’ en de volledige naam is dan zichtbaar via een geïmplementeerde tooltip. Ten slotte wil ik nog vermelden dat er ook is nagedacht over het ontwerp van de selectievakjes. Zoals te zien was in het overzicht van het hele side-panel, was er een selectievakje voor de Axis scale opties, dat gebruikmaakte van Angular’s standaard checkbox. Deze verschildde aanzienlijk van de selectievakjes die door D3.js werden gerenderd. Ik heb ze daarom consistent gemaakt voor D3.js door de hoogte, breedte, kleur en de X- en Y- coördinaten van de vulling van het selectievakje aan te passen. Dit was een proces van veel ‘trial and error’ totdat het perfecte resultaat was bereikt. Hieronder volgt een klein stukje code dat een deel hiervan illustreert, het eindresultaat is zichtbaar in figuur 13.



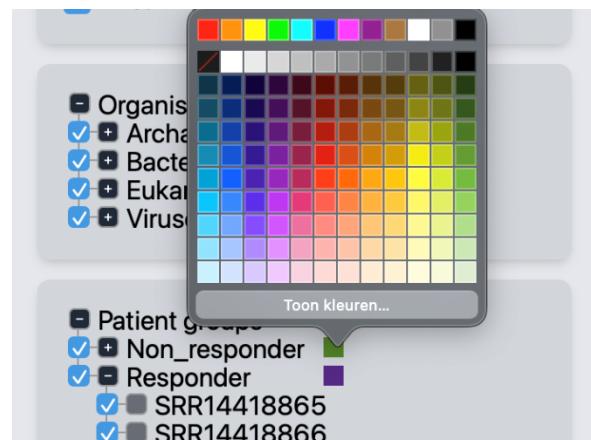
Figuur 13: Selectievakjes

Er is nog meer code die ervoor zorgt dat de links (lijnen tussen de selectievakjes) correct worden gerenderd. Deze code garandeert dat de lijnen elke keer in het midden worden berekend en niet door de selectievakjes heen lopen.

```
nodeEnter.append("rect")
    .attr("class", "selection-checkbox")
    .attr("x", (d: ExtendedHierarchyNode) => d.depth * nodeSize - 22.5)
    .attr("y", -7)
    .attr("width", 12)
    .attr("height", 12)
    .attr("rx", 3)
    .attr("ry", 3)
    .attr("fill", (d: any) => d.data.selected ? "#4299E1" : "transparent")
    .attr("stroke", (d: any) => d.data.selected ? "#4299E1" : "#999")
```

Figuur 14: Stuk code voor selectievakje te berekenen

Voor de renderPatientsPanel-functie is een vergelijkbaar idee toegepast. Deze toont eerst de patiëntengroepen waartoe de patiënten behoren, met daaronder de individuele patiënten die tot deze groepen horen. Het verschil met de taxonomische boomstructuur is dat deze specifieke boom minder complex is. Ik gebruik hier geen aparte interface voor, aangezien de getPatientTree-functie in de dataservice al de patiënten organiseert op basis van hun responderstatus. De structuur omvat enkel de patiëntengroepen met daaronder de bijbehorende patiënten. Het gedrag is echter vergelijkbaar met dat van het taxonomische paneel. De aanpassing van het ontwerp van de selectievakjes is hier op dezelfde manier geïmplementeerd. Het enige verschil is dat de kleur van de groep kan worden aangepast.



Figuur 15: Kleurenkiezer

Ik heb hiervoor een color picker (kleurenkiezer) geïmplementeerd. Deze maakt gebruik van de standaardfunctionaliteit van de browser en stelt de gebruiker in staat om de kleur aan te passen die wordt getoond voor de patiëntengroepen. De gekozen kleur zal direct reflecteren op de andere componenten, waarop waarover verder meer.

Zoals hier te zien is, kan de kleur van de patiëntengroepen worden aangepast door op de voorgedefinieerde HSV-kleur te klikken ([meer hierover in de ColorService](#)). Dit biedt de flexibiliteit om indien gewenst, een andere kleur te kiezen.

Wanneer bijvoorbeeld de Responder-groep wordt uitgevouwen (zoals hierboven getoond), is het paneel vergelijkbaar opgebouwd als de taxonomische weergave, met als verschil dat het de patiënten bevat die tot de Responder-groep behoren.

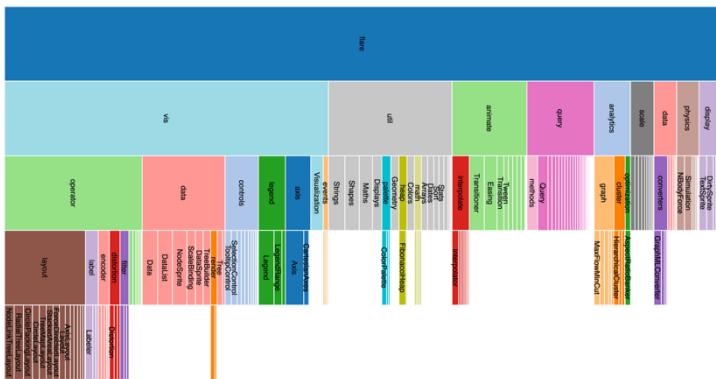
Wat ook relevant is om te vermelden, is dat wanneer de side-panel geopend is, de gehele applicatie naar rechts verschuift. Hiervoor heb ik een simpele service ontwikkeld. Het doel van deze service was het bijhouden en beheren van de open- of gesloten status van de side-panel binnen de applicatie. Hierdoor konden andere componenten zich op deze service abonneren om zo de status van het side-panel te ontvangen en hierdoor de visualisatie te laten krimpen.

Een aspect dat minder direct zichtbaar is, maar cruciaal voor de cross-browser compatibiliteit, is dat de kleurenkiezer eerst niet werkte in Safari. Het toonde afwijkend gedrag bij `input.type = 'color'`. Hier moest dus een workaround voor gevonden worden. De oplossing bestond eruit dat het kleurenkiezerelement onzichtbaar, maar wel klikbaar werd gehouden. Hierna werd het door middel van een timing geactiveerd. Deze oplossing zorgde ervoor dat de kleurenkiezer nu werkt in Safari.

3.5. Icicle Plot

Voorgeschiedenis: Wat is een Icicle Plot?

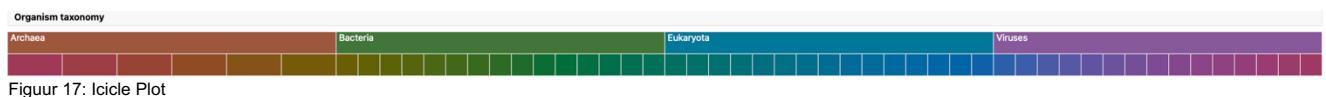
Waarschijnlijk zijn velen van jullie onbekend met de term Icicle Plot. Voor aanvang van deze stage was ik ook niet bekend met deze visualisatie, iets wat gelukkig nu wel het geval is. Een Icicle Plot is een methode voor het presenteren van hiërarchische data. De implementatie in deze applicatie wijkt echter af van de traditionele weergave. Een normale Icicle Plot ziet er als volgt uit:



Figuur 16: Originele Icicle Plot

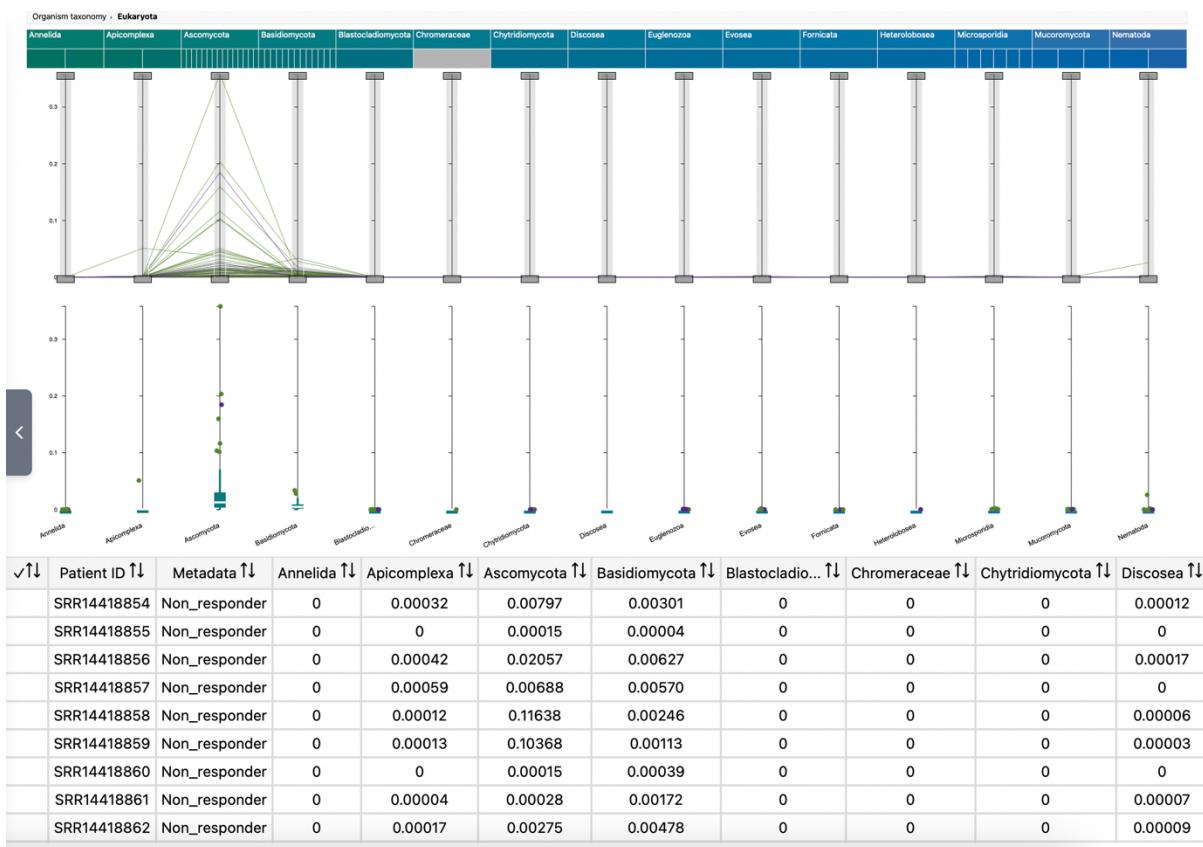
Zoals op figuur 16 te zien is, toont een traditionele Icicle Plot hiërarchische data, waarbij rekening wordt gehouden met de waarde van elke node. Hoe groter de waarde, des te breder de node zal worden gemaakt. Standaard zijn in deze weergave alle niveaus uitgeklapt, van de ouder (parent) tot het laagste kind (child) dat het kan bevatten. Een dergelijk concept is bruikbaar voor onze applicatie, maar het vereist wel bepaalde aanpassingen. Het basisidee is dus perfect.

De Icicle Plot zoals geïmplementeerd in deze applicatie ziet er totaal anders uit, maar volgt wel hetzelfde basisidee .



Figuur 17: Icicle Plot

In deze Icicle Plot is direct zichtbaar dat de weergave gelimiteerd is tot twee niveaus tegelijk. Gebruikers kunnen bijvoorbeeld op *Archea* klikken, waarna de onderliggende children worden getoond. Dit proces kan herhaald worden om dieper in de hiërarchie te navigeren. Het gevolgde pad wordt bovenaan de visualisatie weergegeven als een breadcrumb-navigatie. Tegelijkertijd wordt de hoofdweergave dynamisch bijgewerkt met de data die op dat moment in de Icicle Plot wordt getoond. De onderstaande afbeelding illustreert dit concept visueel (figuur 18):



Figuur 18: Icicle Plot in een child level

In dit voorbeeld is te zien dat ik op *Eukaryota* in de Icicle Plot heb geklikt, wat wordt bevestigd door de bijgewerkte breadcrumb-navigatie bovenaan. Na deze selectie worden alle children direct onder *Eukaryota* getoond (namelijk de phylums van *Eukaryota*). Vervolgens worden de andere visualisaties (de parallel coordinates plot, de boxplot en de spreadsheet) dynamisch bijgewerkt om alleen de data te tonen die op dat moment zichtbaar is in de Icicle Plot. Dit zorgt voor een consistente weergave van de gekozen data in de Icicle Plot over de verschillende componenten heen.

De Icicle Plot zal voor de datalevering gebruikmaken van de `getTaxonomyRoot`-functie uit de dataservice. Deze functie is verantwoordelijk voor het opbouwen en beschikbaar stellen van de hiërarchische structuur van taxonomische data als een `TaxonomyNode`-boom. Elke node in deze boom bevat de geaggregeerde waarden op basis van de ingelezen classificaties en patiënten.

3.5.1. Mijn bijdrage aan de Icicle Plot

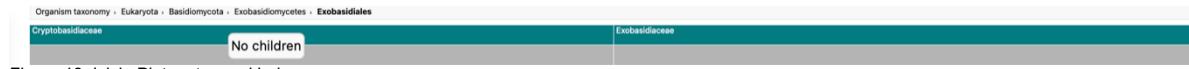
Mijn voornaamste bijdrage aan de Icicle Plot omvatte het gehele ontwerp en de aanpassing van de functionaliteit. Oorspronkelijk was de visualisatie ontworpen om alle hiërarchische niveaus tegelijk te tonen, waarbij de breedte van elk element werd bepaald door de bijbehorende waarde.

Ik heb dit systeem aangepast naar een twee-niveausysteem, wat betekent dat de gebruiker maximaal twee niveaus tegelijk kan zien. Door op een node in het bovenste niveau te klikken, worden de directe children van die node zichtbaar, en dit proces kan herhaald worden om dieper in de hiërarchie te navigeren. Om de gebruiker visueel te informeren over de huidige positie in de hiërarchie, heb ik bovenaan de visualisatie een breadcrumbs-navigatie toegevoegd. Hiermee kan worden geïnteracteerd: door op een naam te drukken, wordt er teruggevaren naar het niveau waar die node zich bevindt.

Verder heb ik de breedteberekening van de nodes aangepast. In tegenstelling tot de traditionele weergave, waar de breedte afhangt van de waarde, worden de nodes nu gelijkmatig verdeeld over de beschikbare breedte. Er is echter een limiet ingesteld op het aantal nodes dat tegelijk wordt getoond: maximaal 15.

Deze limiet geldt per niveau, wat betekent dat elke parent-node maximaal vijftien van zijn directe children kan tonen. Als er meer dan vijftien nodes op een bepaald niveau zijn, worden alleen de eerste vijftien getoond om de leesbaarheid en de overzichtelijkheid te garanderen. De beschikbare ruimte wordt dus gedeeld door het aantal weergegeven nodes (met een maximum van vijftien), waarbij elke node een gelijke breedte krijgt.

Daarnaast is een grijze balk toegevoegd aan het einde van een hiërarchisch pad, voor het geval een node geen verdere children bevat. Deze toevoeging dient twee doelen: het consistent houden van het design door steeds twee niveaus te tonen, en het visueel duidelijk maken dat er geen onderliggende elementen zijn. Zoals in de afbeelding te zien is, verschijnt er een tooltip met de tekst "no children" wanneer er over dit grijze vlak gehoverd wordt.



Figuur 19: Icicle Plot met geen kinderen

Daarnaast zijn er tooltips toegevoegd aan alle nodes in de Icicle Plot. Wanneer de gebruiker over een node hovert, toont een tooltip de volledige naam van de betreffende node extra duidelijk.

Een interessante en cruciale functie in de implementatie is de `createTwoLevelView`:

```
private createTwoLevelView(node: HierarchyNode): HierarchyNode {
  const rootCopy: HierarchyNode = {
    name: node.name,
    children: [],
    aggregatedValue: node.aggregatedValue
  };

  if (node.children && node.children.length > 0) {
    const sortedChildren = [...node.children].sort((a, b) =>
      a.name.localeCompare(b.name)
    );
    const topChildren = sortedChildren.slice(0, this.MAX_CHILDREN_PER_LEVEL);
  }

  private readonly MAX_CHILDREN_PER_LEVEL = 15;
```

Zoals hier te zien is, wordt in dit deel een kopie van de `HierarchyNode` gemaakt. Deze kopie bevat de `name` en de `aggregatedValue`, maar exclusief de `children`. ([Hierover meer](#))

Dit deel controleert eerst of de node nog `children` heeft. Indien dit het geval is, wordt een alfabetisch gesorteerde kopie van deze `children` in een array opgeslagen.

De `children` van de node die hier verwerkt worden, zijn voor weergave beperkt tot maximaal 15. Deze limiet wordt bepaald door de constante `MAX_CHILDREN_PER_LEVEL`, die op 15 is gedefinieerd.

```

rootCopy.children = topChildren.map(child => {
  let grandchildren: HierarchyNode[] = [];
  if (child.children && child.children.length > 0) {
    grandchildren = [...child.children]
      .sort((a, b) => a.name.localeCompare(b.name))
      .slice(0, this.MAX_CHILDREN_PER_LEVEL)
      .map(grandchild => ({
        name: grandchild.name,
        children: [],
        aggregatedValue: grandchild.aggregatedValue
      }));
  } else {
    grandchildren = [{name: 'No children',
      children: [],
      aggregatedValue: 0
    }];
  }
  return {
    name: child.name,
    aggregatedValue: child.aggregatedValue,
    children: grandchildren
  };
};

return rootCopy;
}

```

Figuur 20: Create twoLevelView functie

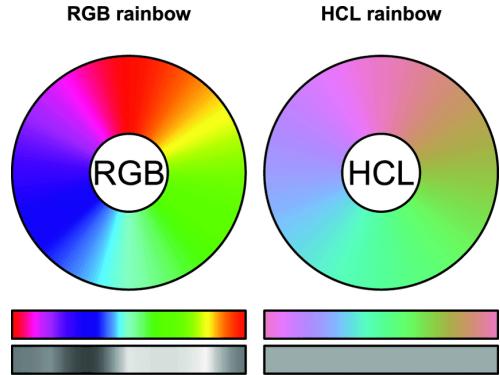
In dit deel van de functie wordt elk child doorlopen uit de topChildren-array. Voor elke child wordt gecontroleerd of er onderliggende grandchildren aanwezig zijn. Indien dit het geval is, worden deze grandchildren alfabetisch gesorteerd. Deze grandchildren worden eveneens beperkt tot maximaal vijftien voor weergave. Deze beperking geldt voor elk niveau. Er wordt vervolgens een vereenvoudigde versie van deze grandchildren gecreëerd, om te garanderen dat zij geen onderliggende niveaus bevatten in de weergave. Indien een node geen grandchildren heeft om te tonen op het volgende niveau, wordt er dummy-data ("No children") toegevoegd. Dit genereert het eerder beschreven grijze vlak en waarborgt zo de consistentie in de weergave, zelfs als er geen verdere data is. Tot slot retourneert de functie de rootCopy met een aangepaste structuur. Deze structuur omvat de root node met maximaal vijftien children, waarbij elk child op zijn beurt maximaal vijftien grandchildren (of een dummy node) bevat. Ter informatie: de root node, genaamd 'Organism taxonomy' wordt bovenaan weergegeven in de breadcrumb-navigatie.

Kleurenschema's voor de Icicle Plot

Het kleurenschema dat op de Icicle Plot wordt toegepast, is zorgvuldig gekozen en getest. Het keuzeproces omvatte het verkennen van twee kleurenschema's, namelijk HCL en HSV, met als doel de meest optimale visualisatie te vinden.

Ik ben begonnen met het testen met het HCL-kleurenschema, wat staat voor Hue (tint), Chroma (verzadiging) en Luminance (helderheid). Dit kleurensysteem is specifiek ontworpen om een uniforme perceptie van kleurverschillen door het menselijk oog te garanderen. Kleurveranderingen worden daardoor gelijkmatiger waargenomen, wat cruciaal is voor effectieve datavisualisatie, met name binnen hiërarchische structuren. (Voor meer details over kleurgebruik in boomstructuren, [zie Bijlage 1.](#))

Zoals de naastgelegen afbeelding illustreert, vertoont het HCL-kleurenschema, in tegenstelling tot het RGB-schema, zachtere en meer gelijkmatige kleurverlopen. Met name de gelijkmatigheid in helderheid (Luminance), zichtbaar in de grijswaardenbalken onder de kleurencirkels, draagt bij aan deze consistentie perceptie, wat bij RGB minder het geval is. Dit resulteert in een meer gebalanceerde en prettigere kleurpercepcie voor het menselijk oog.



Figuur 21: RGB en HCL

Vervolgens heb ik ook het HSV-kleurenschema getest. Voor de implementatie van dit schema was het noodzakelijk om een functie te ontwikkelen die HSV-waarden kon omzetten naar RGB. Waar D3.js een ingebouwde HCL-functie biedt die automatisch de omzetting verzorgt voor het werken met het fill-attribuut (voor het kleuren van de Icicle Plot), was het voor HSV noodzakelijk om zelf een conversiefunctie te implementeren.

```

const hsvToRgb = (h: number, s: number, v: number): string => {
  let r, g, b;
  const i = Math.floor(h * 6);
  const f = h * 6 - i;
  const p = v * (1 - s);
  const q = v * (1 - f * s);
  const t = v * (1 - (1 - f) * s);

  switch (i % 6) {
    case 0: r = v; g = t; b = p; break;
    case 1: r = q; g = v; b = p; break;
    case 2: r = p; g = v; b = t; break;
    case 3: r = p; g = q; b = v; break;
    case 4: r = t; g = p; b = v; break;
    case 5: r = v; g = p; b = q; break;
    default: r = v; g = t; b = p;
  }
}

```

Figuur 22: hsvToRgV functie

Deze functie zorgt ervoor dat HSV-kleuren correct worden omgezet naar RGB. Eerst worden de waarden voor h (hue), s (saturation) en v (value) gedefinieerd. Vervolgens wordt de kleurencirkel opgedeeld in 6 sectoren. Daarna wordt de fractie binnen de huidige sector berekend. Vervolgens wordt een waarde met minimale verzadiging (p) berekend. Daarna worden twee tussenwaarden (q en t) berekend die afhankelijk zijn van de fractie en de verzadiging. Afhankelijk van de sector, worden de RGB-waarden toegewezen door een specifieke combinatie van de componenten v, p, q en t.

Tot slot worden de RGB-waarden vermenigvuldigd met 255, omgezet naar hexadecimale waarden en samengevoegd tot een hex-kleurcode.

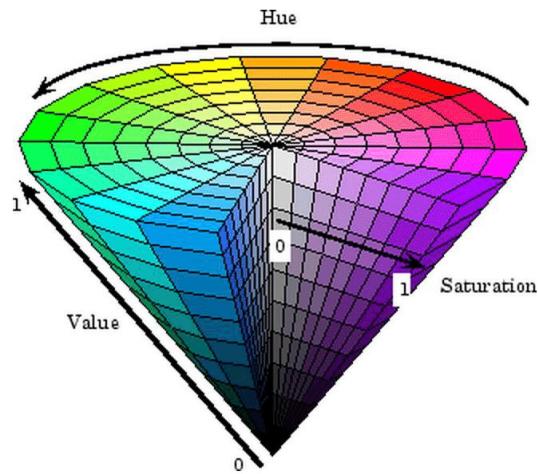
```

  return `#${Math.round(r * 255).toString(16).padStart(2, '0')}#${Math.round(g * 255).toString(16).padStart(2, '0')}#
#${Math.round(b * 255).toString(16).padStart(2, '0')}`;
}

```

Figuur 23: Return naar hexcode

Hieronder wordt het HSV-kleurenschema kort toegelicht. HSV staat voor Hue (tint), Saturation (verzadiging) en Value (helderheid). Dit kleurenschema wordt meestal voorgesteld in een kleurencilinder of driehoek.



Figuur 24: HSV kleuren cirkel

De cirkel rond de as (hue) bepaalt de gewenste kleurtoon. De afstand van het midden naar de rand toe (saturation) bepaalt de verzadiging. Deze varieert van grijs in het midden tot de pure kleur aan de rand. De hoogte (value) bepaalt de helderheid van de kleur, waarbij de onderkant zwart is en de bovenkant maximale helderheid vertegenwoordigt. Dit model is eenvoudig te implementeren, omdat hiermee de verzadiging van een kleur gemakkelijk kan worden aangepast zonder de kleurtoon te wijzigen. Op vergelijkbare wijze kan de helderheid eenvoudig worden aangepast zonder de kleurtoon te beïnvloeden.

Gezien de besproken achtergrondinformatie bleek het HCL-kleurenschema geschikter voor de Icicle Plot. In de Icicle Plot worden maximaal twee levels getoond, waarbij elk level een maximum van vijftien children kan bevatten. Met het oog op optimale visuele esthetiek bleek het HCL-kleurenschema dan ook beter te passen.

Daarentegen vertoonde het HSV-kleurenschema te abrupte overgangen tussen de verschillende nodes en een oververzadigd, visueel minder prettig uiterlijk. Om deze redenen heb ik gekozen voor het HCL-kleurenschema.

De lettertypekleur van de namen op de Icicle Plot wordt aan de hand van een formule berekend. Dit is nodig, aangezien de achtergrond een lichtere of donkere tint kan hebben. Hierdoor is een witte lettertypekleur vaak een betere optie bij een donkere achtergrond terwijl zwart de voorkeur heeft bij een lichtere achtergrond. De luminantieformule wordt hiervoor toegepast.

Eerst wordt de achtergrondkleur van de node opgehaald. Deze wordt vervolgens omgezet naar D3-

```
const backgroundColor = d3.select(this).select('rect').attr('fill');

const color = d3.color(backgroundColor);
let brightness = 0;

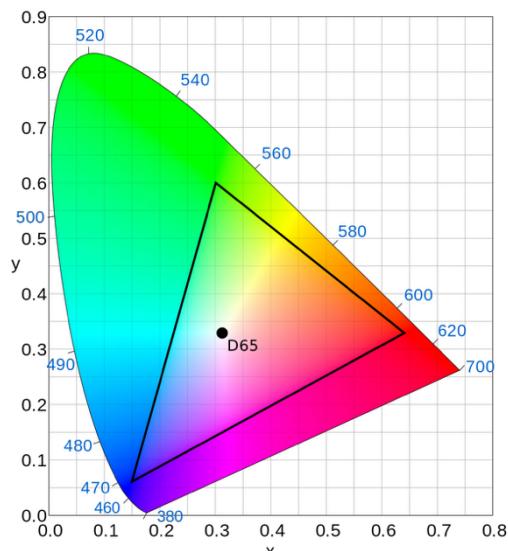
if (color) {
  const rgb = color.rgb();
  brightness = (rgb.r * 0.2126 + rgb.g * 0.7152 + rgb.b * 0.0722);
}

const textColor = brightness > 128 ? 'black' : 'white';
```

Figuur 25: Code met luminantieformule

kleurobject, wat de verdere verwerking vereenvoudigt. Hierna wordt de helderheid (luminantie) van de RGB-kleur berekend. Deze formule geeft meer gewicht aan de kleur groen, omdat het menselijk oog gevoeliger is voor groen licht dan voor andere kleuren. De lettertypekleur (textColor) wordt bepaald: zwart indien de helderheid groter is dan 128, anders wit.

De gewichten die in de formule worden gebruikt, zijn niet willekeurig gekozen, maar komen uit de luminantieformule van het sRGB-kleurmodel. Dit is een standaardformule voor het berekenen van de helderheid zoals deze door het menselijk oog wordt waargenomen. Deze gewichten vinden hun oorsprong in de officiële standaard van ITU-R BT.709. Deze standaard wordt voornamelijk toegepast voor HDTV(High Definition Television).



Figuur 26: CIE 1931 xy chromaticiteitsdiagram

Het getoonde kleurspectrum represeneert het volledige bereik dat het menselijke oog kan waarnemen, afgebeeld in het CIE 1931 xy chromaticiteitsdiagram. De x- en y-assen vertegenwoordigen hierin de chromaticiteitscoördinaten. De gebogen buitenrand, met aanduidingen in nanometers, staat voor het spectrum van monochromatisch licht. De hierin afgebeelde driehoek geeft het kleurengamma weer van ITU-R BT.709. De punten op deze driehoek vertegenwoordigen de primaire kleuren. Het punt D65 wordt beschouwd als het witpunt. Met deze waarden wordt een matrix geconstrueerd die een transformatie uitvoert naar het CIE-XYZ-kleurensysteem. De Y-component in CIE-XYZ represeneert de helderheid. Deze wordt vaak gebruikt voor WCAG (Web Content Accessibility Guidelines), om het contrast tussen de tekst en achtergrond te berekenen.

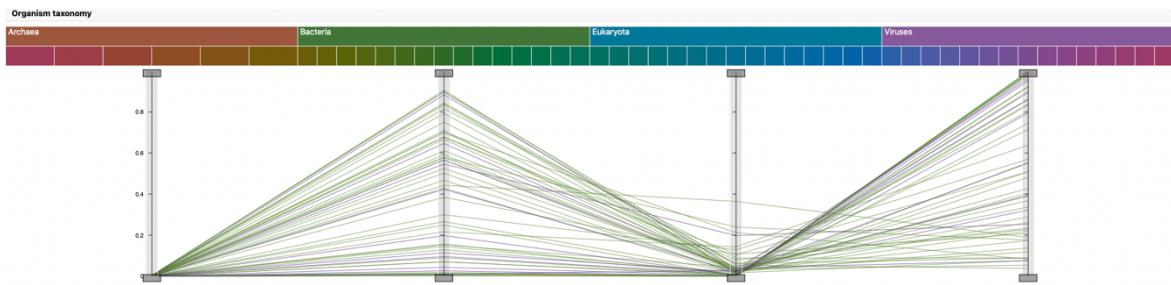
Het is ook relevant om te vermelden dat wanneer de tekst te lang is om binnen een node op de Icicle Plot te passen, de weergave wordt aangepast. Er wordt eerst gecontroleerd of de tekst binnen de node past, indien dit het geval is, wordt de tekst volledig getoond. Indien de tekst niet in één keer past, wordt deze opgesplitst in woorden, waarbij zoveel mogelijk woorden per regel worden geplaatst. Wanneer de tekst over meerdere regels verdeeld moet worden, wordt tevens de lettergrootte verkleind om ervoor te zorgen dat de gehele tekst zichtbaar blijft binnen de node.

Candidatus Chazhemtobacteraceae	Candidatus Melainabacteria
------------------------------------	-------------------------------

Figuur 27: Icicle Plot tekstafbreking

3.6. Parallel Coordinates Plot

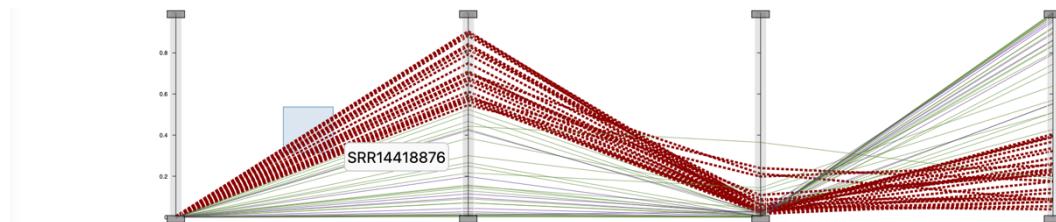
De Parallel Coordinates Plot dient om inzicht te geven in de verdeling van de patiëntdata over de verschillende taxa. De structuur van de assen in de Parallel Coordinates Plot correspondeert met de hiërarchie die in de Icicle Plot wordt weergegeven. Bijvoorbeeld, indien *Archaea* als eerste taxon in de Icicle Plot verschijnt, dan vertegenwoordigt de eerste y-as van de Parallel Coordinates Plot de patiëntdata voor dit taxon. Op deze wijze wordt de verdeling van de patiëntdata over de verschillende taxaniveaus inzichtelijk gemaakt. In onderstaand voorbeeld betreft dit de taxaniveaus: *Archaea*, *Bacteria*, *Eukaryota*, en *Viruses*.



Figuur 28: Icicle Plot met Parallel Coordinates Plot

De patiëntwaarden zijn ook geaggregeerd, zoals eerder vermeld. Dit houdt in dat alle onderliggende taxa binnen bijvoorbeeld *Archea* voor een specifieke patiënt worden gesommeerd om de totale waarde voor dat taxaniveau te bepalen. De kleuren van de parallellijnen zijn gebaseerd op de patiëntgroep waartoe de patiënt behoort. Deze kleurtoewijzing kan worden geconfigureerd via het side-panel ([hierover meer](#)) van de visualisatie. Elke y-as is tevens voorzien van een brush (een grijze selectievak). Deze brush dient voor het filteren van de patiëntdata, door de brush te verslepen, verdwijnen patiënten buiten het geselecteerde bereik van de Parallel Coordinates Plot. Dit maakt het mogelijk om gerichte analyse uit te voeren op specifieke segmenten van de data.

Naast filteren is het ook mogelijk om een selectie van patiënten te maken door een rechthoek te tekenen over de parallellijnen. Het is ook mogelijk om een individuele lijn te selecteren. De geselecteerde patiënten worden vervolgens duidelijk gemarkeerd in de bijbehorende spreadsheet en op de Parallel Coordinates Plot. Dit maakt het mogelijk om snel groepen patiënten, specifieke patiënten of outliers te identificeren.



Figuur 29: Selectie op Parallel Coordinates Plot

3.6.1. Mijn bijdrage aan de Parallel Coordinates Plot

De initiële opzet van de Parallel Coordinates Plot, oftewel het skelet van deze visualisatie, is door mij gerealiseerd. Hieraan ging onderzoek naar de optimale implementatiestrategie. Tevens heb ik de visualisatie van de patiëntdata in de Parallel Coordinates Plot geïmplementeerd. Deze functionaliteit was echter nog hardgecodeerd, wat betekende dat een interactieve drill-down vanuit de Icicle Plot nog geen dynamische aanpassing van de Parallel Coordinates Plot tot gevolg had.

Aansluitend heb ik de brushes geïmplementeerd met behulp van de brush-logica van D3. Hiervoor was het vereist een object (private brushes) te initialiseren dat voor elke dimensie (y-as) een corresponderend D3.BrushBehavior-object opslaat.

```
private brushes: { [key: string]: d3.BrushBehavior<unknown> } = {};
```

Figuur 30: Private brushes

Daaropvolgend is met behulp van d3.brushY() de functionaliteit geconfigureerd die effectief een verticale brush op elke as van de Parallel Coordinates Plot tekent. Hierbij wordt een selectiegebied gedefinieerd dat de volledige lengte van elke y-as beslaat. Tevens zijn twee event handlers geconfigureerd: start, geactiveerd bij aanvang van het slepen, en end, geactiveerd wanneer de brush wordt losgelaten. Verder in de applicatie is logica geïmplementeerd die bij het loslaten van de brush de data filtert. Hierdoor worden enkel de patiënten getoond die binnen het geselecteerde bereik vallen. Deze berekening wordt pas bij het loslaten van de brush uitgevoerd. Dit is voornamelijk performance-gerelateerd, aangezien een live-berekening per lijn tot performanceproblemen leidde doordat bij elke beweging van de brush de volledige dataset opnieuw gefilterd en gevisualiseerd moest worden.

Ook heb ik de initiële selectiefunctionaliteit geïmplementeerd. Hiervoor heb ik een Angular service genaamd HoverService ontwikkeld, die gebruikmaakt van RxJS's BehaviorSubject om bij te houden welke patiënt momenteel is geselecteerd op de Parallel Coordinates Plot. Deze service is vervolgens geïmplementeerd in de spreadsheet-component. Dit resulteerde in het markeren van de geselecteerde patiënt in de spreadsheet. Dit vormde de basis voor de selectiefunctionaliteit, waarvoor reeds een solide concept was ontwikkeld. Deze service kan tevens in andere componenten worden geïntegreerd om de functionaliteit uit te breiden.

Wat de lijnen betreft: deze vertegenwoordigen individuele patiënten, die elk tot een specifieke patiëntgroep behoren. Daarom is ervoor gekozen om de kleur van de lijnen te baseren op de toegewezen patiëntgroep. Zoals eerder vermeld, kan deze kleurtoewijzing worden geconfigureerd via het side-panel. Deze kleuren worden vervolgens via de ColorService doorgegeven aan de Parallel Coordinates Plot.

Allereerst is het noodzakelijk om de verschillende patiëntgroepen binnen de Parallel Coordinates Plot te onderscheiden. De onderstaande functionaliteit haalt de patiëntgroepen op uit een boomstructuur

```
const updateLines = () => {
  const patient2Root = this.dataService.getPatient2Tree();
  if (patient2Root?.children) {
    const groupNames = patient2Root.children.map(group => group.name);
    this.colorService.assignColorsToGroups(groupNames);
  }
}
```

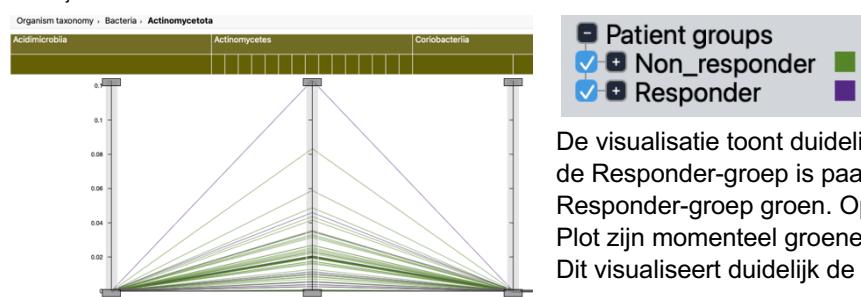
Figuur 31: Functie updateLines

(Patient2Tree) en gebruikt de ColorService om een unieke kleur aan elke groep toe te wijzen. Dit maakt het visueel mogelijk om individuele patiënten te koppelen aan hun respectieve groep binnen de plot.

```
.style('stroke', d => {
  const patientId = d.patientId;
  for (const group of patient2Root.children || []) {
    const patientInGroup = group.children?.find(p => p.name === patientId);
    if (patientInGroup) {
      return this.colorService.getColorForGroup(group.name);
    }
  }
  return 'black';
})
```

Figuur 32: Stijlfunctie

Binnen deze stijlfunctie wordt voor elke lijn de patiëntgroep bepaald waar de patiënt toe behoort. Vervolgens wordt de kleur opgehaald die door de ColorService aan deze groep is toegewezen. Indien de patiënt niet aan een groep kan worden gekoppeld, wordt de lijnkleur standaard zwart.

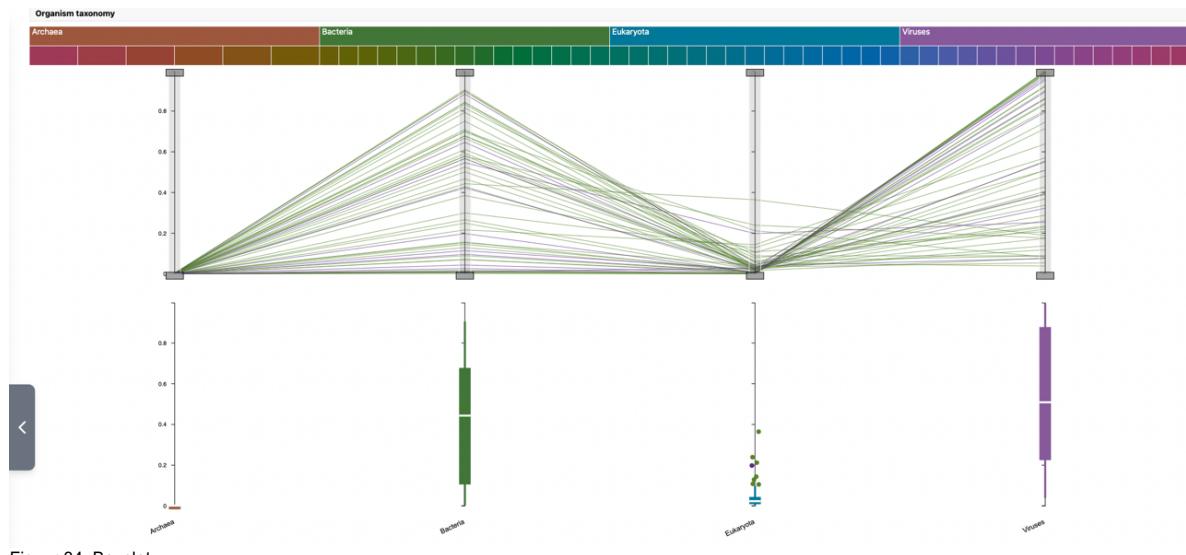


Figuur 33: Toegewezen kleuren patient groups

De visualisatie toont duidelijk de toegewezen kleuren: de Responder-groep is paars gekleurd en de Non-Responder-groep groen. Op de Parallel Coordinates Plot zijn momenteel groene en blauwe lijnen zichtbaar. Dit visualiseert duidelijk de aanwezige patiëntgroepen.

3.7. Boxplot

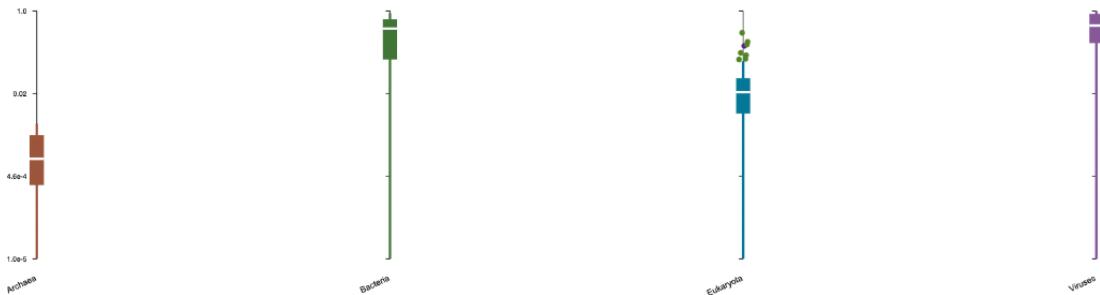
De boxplot is een veelgebruikte visualisatie vorm, die wordt ingezet voor het weergeven van de distributie van de patiëntwaarden per taxa. Zoals hieronder weergegeven is de boxplot afhankelijk van de icicle plot, net zoals de Parallel Coordinates Plot dat was.



Figuur 34: Boxplot

Voor taxa zoals *Archea* en *Eukaryota* zijn de lijnen op de Parallel Coordinates Plot laag geconcentreerd, wat resulteert in een minimale boxplot-weergave. Het inschakelen van de [logaritmische schaal](#) kan hierbij uitkomst bieden. Daarentegen tonen *Bacteria* en *Viruses* wel een duidelijke boxplot, aangezien de corresponderende lijnen op de Parallel Coordinates Plot een breder scala aan waarden (laag en hoog) beslaan. Ook worden outliers duidelijk gevisualiseerd als individuele punten op de boxplot.

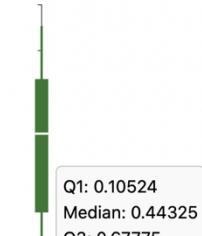
Met een logaritmische schaal wordt de weergave als volgt, waardoor het mogelijk is om voor elke node een boxplot te observeren:



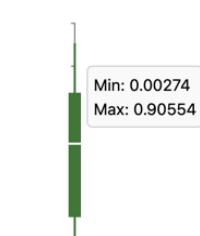
Figuur 35: Boxplot met logaritmische schaal

3.7.1. Mijn bijdrage aan de boxplot

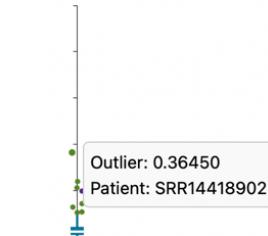
Aan de boxplots is een tooltip-functionaliteit toegevoegd. Bij hover over een boxplot wordt relevante informatie getoond. Deze omvat de waarde voor het eerste kwartiel, de mediaan en het derde kwartiel (figuur 35). Ook is het mogelijk om bij hover over de whiskers de minimum- en maximumwaarden van deze whisker te raadplegen (figuur 36). Een aparte tooltip is ook beschikbaar voor outliers. Bij hover over een outlier wordt deze tooltip weergegeven, die de waarde van de outlier toont, samen met de naam van de corresponderende patiënt (figuur 37).



Figuur 36: d



Figuur 37: d



Figuur 38: d

Daarnaast is er interactie geïmplementeerd voor de boxplot. Het is mogelijk om op de verschillende kwartieren, de whiskers, de mediaan en de outliers te klikken. Vervolgens wordt deze selectie zichtbaar op de Parallel Coordinates Plot. Bij selectie van bijvoorbeeld het eerste kwartiel worden op de Parallel Coordinates Plot alle lijnen van de patiënten geselecteerd die binnen het bereik van dat kwartiel vallen. Indien de mediaan wordt gekozen, worden de patiënten geselecteerd wier waarden zich bevinden binnen een bereik van $\pm 5\%$ rond de mediaan. Alle selecties die op de Parallel Coordinates Plot worden weergegeven, worden ook in de spreadsheet-component gemarkeerd.

Hier voor is gebruikgemaakt van de HoverService, die reeds door mijn medestudent was uitgebreid. Mijn bijdrage omvatte het toevoegen van de selectie-logica in de boxplot-component, zodat de selectie ook werd teruggekoppeld (reflected) op de andere componenten. Hiertoe diende ik twee functies toe te voegen aan de HoverService: addSelectedUser en isUserSelected.

De functie addSelectedUser dient voor het toevoegen van een patiënt aan de huidige selectie.

De functie isUserSelected was noodzakelijk om, bij het selecteren van meerdere kwartieren van diverse taxa, de reeds geselecteerde patiënten te onthouden en deze correct toe te voegen. Zonder deze functionaliteit zouden patiënten die in meerdere geselecteerde kwartieren aanwezig zijn, mogelijk uit de selectie worden weggelaten.

```
if (event.ctrlKey || event.metaKey) {
  patientsInRange.forEach(id => {
    if (!this.hoverService.isUserSelected(id)) {
      this.hoverService.addSelectedUser(id);
    }
  });
} else {
  const currentSelection = this.hoverService.getSelectedUsers();
  const allAlreadySelected = patientsInRange.every(id =>
    currentSelection.includes(id)
  ) && patientsInRange.length === currentSelection.length;

  if (allAlreadySelected) {
    this.hoverService.clearSelectedUsers();
    svg.selectAll('.line-highlight').remove();
  }

  } else {
    this.hoverService.clearSelectedUsers();
    patientsInRange.forEach(id => this.hoverService.addSelectedUser(id));
  }
}
```

Figuur 39: d

Deze code implementeert de selectie op de boxplot. Deze bestaat uit verschillende stappen. Vergelijkbaar met de functionaliteit in Excel, dient de Ctrl-toets ingedrukt te worden wanneer diverse onderdelen geselecteerd dienen te worden. Zonder deze toets resulteert dit in een enkele selectie. Deze logica is in de code toegepast. De controle op de metaKey waarborgt functionaliteit op Mac-systemen, aangezien Mac-gebruikers de Command-toets hanteren in plaats van de Control-toets. Wanneer deze wordt ingedrukt, voegt de code de patiënten toe aan de bestaande selectie. Bij een enkele selectie wordt de vorige selectie opgeheven en wordt

uitsluitend het zojuist aangeklikte onderdeel geselecteerd. Indien nogmaals op hetzelfde element wordt geklikt, wordt dit gedeselecteerd door de allReadySelected-logica.

De kleuren voor de boxplot zijn overgeërfd van de Icicle Plot. Om meer consistentie binnen de applicatie te waarborgen, draagt dit bij aan een gelijkmatige uitstraling. Om dit te realiseren, is een extra functie toegevoegd aan de DataService: setNodeColor, die dient voor het opslaan van de kleur voor een specifieke node.

```
public setNodeColor(nodeName: string, color: string): void {
    this.nodeColors.set(nodeName, color);
}
```

Deze functie slaat een kleur op voor een specifieke node. Deze zal worden gebruikt in de Icicle Plot.

```
public getNodeColor(nodeName: string): string {
    return this.nodeColors.get(nodeName) || '#69b3a2';
}
```

```
svg.append('rect')
    .attr('class', 'boxplot-element')
    .attr('x', xPos - boxWidth / 2)
    .attr('y', yScale(q3) + dynamicYOffset)
    .attr('height', boxHeight)
    .attr('width', boxWidth)
    .style('fill', this.dataService.getNodeColor(dimension))
```

Figuur 40: Functie voor kleur boxplot

Deze functie haalt de opgeslagen kleur voor de opgegeven node op. Als er geen opgeslagen kleur is, of bij een fout, wordt een standaardkleur (default kleur) gebruikt. Deze wordt gebruikt in de boxplot.

Zoals hiernaast afgebeeld in het stukje code, wordt de getNodeColor gebruikt voor de fill-eigenschap van de rechthoek (de box van de boxplot). Hierbij wordt dimension als variabele meegegeven. Deze variabele bevat de naam van de huidige boxplot, waardoor de naam overeenkomt met die in de Icicle Plot.

Voor de kleur van de mediaanlijn in de boxplot wordt dezelfde logica gebruikt als voor de kleuren van de namen op de Icicle Plot. Hierbij wordt de luminantieformule toegepast.

Resterend zijn de outliers op de boxplot, het toekennen van de kleur van de Icicle Plot zou ongewenst zijn. De outliers zijn voorzien van de kleur van de patiëntengroep in de side-panel. Dit concept komt overeen met de kleurbepaling in de Parallel Coordinates Plot, hierbij wordt de groupName van de outlier meegegeven.

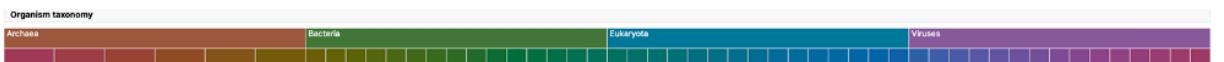
```
const outlierColor = this.colorService.getColorForGroup(groupName);
```

Figuur 41: Outlier kleur

Vervolgens wordt de outlier gevuld met deze kleur. De groupName wordt bepaald op basis van de individuele outlier. Eerst wordt de patientId uit de outlier opgehaald. Deze patientId wordt vervolgens opgezocht in de boomstructuur van patiënten (getPatientTree) om de corresponderende groep te vinden. Zodra deze gevonden is, wordt de groupName hierop ingesteld. Dit proces werkt hetzelfde als dat van de Parallel Coordinates Plot. Het enige verschil is dat de outlier terugvalt op de default kleur van de ColorService, terwijl de Parallel Coordinates Plot direct een zwarte kleur toekende.

3.8. Spreadsheet

De spreadsheet dient om de onbewerkte patiëntendata weer te geven in tabelvorm, tezamen met taxonomische classificaties en geassocieerde, geaggregeerde waarden voor elke patiënt. Voor de spreadsheet is een template toegepast van PrimeNG.



✓↑↓	Patient ID ↑↓	Metadata ↑↓	Archaea ↑↓	Bacteria ↑↓	Eukaryota ↑↓	Viruses ↑↓
	SRR14418854	Non_responder	0.00249	0.42898	0.01294	0.55406
	SRR14418855	Non_responder	0	0.00274	0.00024	0.99701
	SRR14418856	Non_responder	0.00075	0.57967	0.02842	0.39040
	SRR14418857	Non_responder	0.00207	0.50148	0.01391	0.48129
	SRR14418858	Non_responder	0.00058	0.29981	0.12882	0.57037
	SRR14418859	Non_responder	0.00027	0.38502	0.10547	0.50895
	SRR14418860	Non_responder	0.00002	0.00388	0.00057	0.99552
	SRR14418861	Non_responder	0.00003	0.00973	0.00272	0.98749
	SRR14418862	Non_responder	0.00035	0.08966	0.00869	0.90101

Figuur 42: Icicle plot en spreadsheet

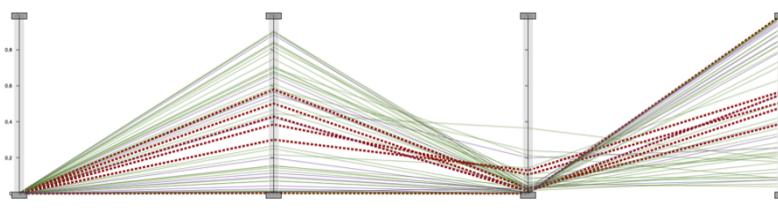
Zoals op bovenstaande afbeelding te zien is, bevat de tabel verschillende kolomnamen. De selectiekolom, Patient ID en metadata zijn statisch gedefinieerd. Dit betekent dat deze namen permanent aanwezig zullen zijn. De aangrenzende kolomnamen zijn afhankelijk van de visualisatie op de Icicle Plot. De hiërarchische niveaus van de Icicle Plot corresponderen met de kolomnamen naast de metadata. Deze worden dynamisch geplaatst.

Indien de naam van een taxonomische classificatie te lang is om volledig weer te geven (bij een lengte van meer dan 15 tekens), dan wordt deze afgekapt. Vervolgens worden de eerste tekens, gevolgd door een ellips (...), getoond, en een tooltip wordt beschikbaar gemaakt voor deze kolom, zoals hieronder afgebeeld.



Figuur 43: Spreadsheet tooltip

Wanneer meerdere patiënten in de Parallel Coordinates Plot worden geselecteerd, wordt deze selectie nu duidelijk weergegeven op de plot. De selectiefunctie werkt eveneens in twee richtingen: het is mogelijk om een patiënt in de spreadsheet aan te duiden, waarna deze patiënt wordt aangeduid op de Parallel Coordinates Plot.



In de spreadsheet is het mogelijk om meerdere patiënten te selecteren met behulp van de Shift-toets. Deze selectie wordt vervolgens ook duidelijk weergegeven op de Parallel Coordinates Plot. Zoals op de afbeeldingen te zien is.

✓↑↓	Patient ID ↑↓	Metadata ↑↓	Archaea ↑↓	Bacteria ↑↓	Eukaryota ↑↓	Viruses ↑↓
✓	SRR14418854	Non_responder	0.00249	0.42898	0.01294	0.55406
✓	SRR14418855	Non_responder	0	0.00274	0.00024	0.99701
✓	SRR14418856	Non_responder	0.00075	0.57967	0.02842	0.39040
✓	SRR14418857	Non_responder	0.00207	0.50148	0.01391	0.48129
✓	SRR14418858	Non_responder	0.00058	0.29981	0.12882	0.57037
✓	SRR14418859	Non_responder	0.00027	0.38502	0.10547	0.50895
	SRR14418860	Non_responder	0.00002	0.00388	0.00057	0.99552
	SRR14418861	Non_responder	0.00003	0.00973	0.00272	0.98749
6	SRR14418862	Non_responder	0.00035	0.08966	0.00869	0.90101

Figuur 44: Interactie spreadsheet en Parallel Coordinates Plot

3.8.1. Mijn bijdrage aan de spreadsheet

Aanvankelijk werkte de applicatie nog niet met dynamische patiëntendata. Om dit op te lossen, is een metadata-kolom toegevoegd. Deze kolom is bedoeld voor de weergave van de naam van de patiëntengroep waartoe de betreffende patiënt behoort. Deze informatie wordt dynamisch samengesteld door de patientTree. Deze boomstructuur ordent de patiënten hiërarchisch op basis van hun groep. Met behulp van de extractPatientMetadata-functie wordt deze boom doorlopen, waarbij elke patiënt en diens groep worden geïdentificeerd. Vervolgens wordt, binnen de findClassificationsForPatients-functie, elk patiëntobject opgehaald en verrijkt met deze metadata, zodat het direct in de spreadsheet zichtbaar is.

```
this.patientsData = this.patientsData.map(patient => {
  const patientId = patient.patientId;
  const metadata = this.patientMetadata.get(patientId);
  return {
    ...patient,
    responderStatus: metadata ? metadata.status : 'Unknown',
    patientPath: metadata ? metadata.path : patientId
  };
});
```

Figuur 45: Spreadsheet map functie

Deze map-functie voegt eigenschappen zoals de responderstatus en de patientPath toe aan elk patiëntobject. Deze eigenschappen worden dynamisch afgeleid uit de bijbehorende metadata, waarbij standaardwaarden worden toegekend indien de metadata ontbreekt. Hierdoor zijn ze beschikbaar voor de weergave in de spreadsheet.

Het is mogelijk gemaakt om de kolombreedte van elke kolom in de spreadsheet zelf aan te passen. Deze functionaliteit wordt ingebouwd geleverd door de p-table-component van PrimeNG. Op de p-table-component wordt het attribuut 'resizableColumns' ingesteld op 'true', wat aangeeft dat de tabel de mogelijkheid krijgt om de kolommen breder of smaller te maken. Daarnaast is een specifiek gedrag vereist bij het aanpassen van de kolombreedte, hiervoor kan men het attribuut 'columnResizeMode' met de waarde 'expand' toepassen op de p-table-component. Dit resulteert erin dat wanneer een kolom wordt vergroot of verkleind, de totale breedte van de tabel meeverandert, in plaats van dat andere kolommen automatisch krimpen of uitzetten. Deze logica wordt volledig toegepast in het HTML-bestand.

```
<p-table [value]="patientsData"
  [tableStyle]="{{ 'min-width: 70rem' }}"
  [loading]="loading" [scrollable]="true" scrollHeight="{{scrollHeight}}"
  [resizableColumns]="true" columnResizeMode="expand" [virtualScroll]="true"
  styleClass="p-databutable-gridlines custom-grid-table"
  (click)="!$event.target === $event.currentTarget && clearSelection()>
```

Figuur 46: PrimeNG kolom vergroten

Hierna diende de PrimeNG-directive 'pResizableColumn' te worden toegevoegd aan elk th-element. Deze zorgt voor de weergave van een handvat. Dit is een visueel element dat aangeeft dat de kolom in breedte aanpasbaar is.

```
<th class="table-header" pResizableColumn pSortableColumn="patientId">
  Patient ID
  <p-sortIcon field="patientId"></p-sortIcon>
</th>
```

Figuur 47: PrimeNG kolom vergroten visueel element

Verder is er CSS toegevoegd ter verbetering van de gebruikerservaring. Het belangrijkste aspect hiervan is dat, wanneer men over de kolomlijn beweegt, de muisaanwijzer verandert in een 'dragable' (versleepbaar) element. Dit visualiseert dat de kolom in breedte kan worden aangepast.

Bovendien is een sorteefunctie toegevoegd per kolom. Dit verklaart de aanwezigheid van de uiterst linkse selectiekolom (met een vinkje). Deze functionaliteit vergemakkelijkt het direct inzien van geselecteerde patiënten, aangezien handmatig doorzoeken van de gehele spreadsheet inefficiënt is.

De sorteermethode voor de overige kolommen hanteert een vergelijkbaar principe: voor numerieke waarden is dit van klein naar groot of omgekeerd, terwijl voor metadata (patiëntengroepen) en Patiënt-ID de sortering alfabetisch (A-Z of Z-A) zal zijn.

Dit is gerealiseerd door aan elk th-element de directive 'pSortableColumn' toe te voegen. Dit informeert PrimeNG dat de betreffende kolom sorteerbaar is en koppelt deze aan het specifieke veld.

```
<th class="table-header" pResizableColumn pSortableColumn="patientId">
  Patient ID
  <p-sortIcon field="patientId"></p-sortIcon>
</th>
```

Figuur 48: Sorteer icon spreadsheet

Voor de kolom Patient ID zal het veld patientId worden.



Om de sorteerbaarheid van een kolom te visualiseren, wordt er ook een 'p-sortIcon'-component toegevoegd binnen elk th-element. Deze component toont een icoon (zoals een pijl omhoog of omlaag) dat de sorteerrichting (oplopend, aflopend, of ongesorteerd) aangeeft, zoals hiernaast te zien is.

Voor elke kolom wordt het sorteren standaard door PrimeNG beheerd, met uitzondering van de selectiekolom (de kolom met het vinkje). Aan deze kolom is het 'field'-attribuut 'selected' toegewezen, wat verwijst naar de 'selected'-eigenschap in de patientsData-array. Wanneer op deze kolom wordt geklikt, wordt de data gesorteerd op basis van deze eigenschap.

De functie 'updateSelectedProperty()' speelt hierin een belangrijke rol. Deze methode wordt telkens aangeroepen wanneer patiënten worden geselecteerd of gedeselecteerd, hierdoor wordt gegarandeerd dat de 'selected'-eigenschap altijd up-to-date is met de huidige selectiestatus.

Daarnaast is de spreadsheet uitgebreid met Excel-achtige selectiefunctionaliteit. Hoewel een Shift-selectie-optie al beschikbaar was, diende deze verder uitgebreid te worden. Het is nu mogelijk om door de Ctrl-toets ingedrukt te houden, meerdere elementen afzonderlijk te selecteren. Vergelijkbaar met Excel-gedrag, zal bij een klik zonder de Ctrl-toets de huidige selectie worden opgeheven, waarna enkel het specifiek aangeklikte veld geselecteerd blijft. Deselectie kan ook plaatsvinden door een nieuwe lege rij te selecteren, of door specifiek op de selectiekolom (het vinkje) te klikken.

```
public selectRow(userId: string, event?: MouseEvent): void {
  if (event) {
    event.preventDefault();
  }

  if (event && (event.ctrlKey || event.metaKey)) {
    if (this.selectedUsers.includes(userId)) {
      this.hoverService.removeSelectedUser(userId);
    } else {
      this.hoverService.addSelectedUser(userId);
    }
    this.lastSelectedRowId = userId;
  }
  else if (event && event.shiftKey && this.lastSelectedRowId) {
    this.selectRowRange(this.lastSelectedRowId, userId);
  }
  else {
    this.lastSelectedRowId = userId;
    this.hoverService.setSelectedUser(userId);
  }
}
```

Figuur 49: Spreadsheet selectRow

Eerst worden alle standaard browseracties onderdrukt. Dit zorgt ervoor dat uitsluitend de eigen gedefinieerde logica wordt toegepast en ongewenst gedrag wordt voorkomen. Wanneer de Ctrl-toets (of command op Mac) wordt ingedrukt, wordt gecontroleerd of de aangeklikte rij al deel uitmaakt van de selectie. Als de rij al geselecteerd is terwijl de Ctrl-toets is ingedrukt, zal deze worden gedeselecteerd. Als de rij nog niet geselecteerd is, wordt deze juist aan de selectie toegevoegd. De 'userId' van de laatst geklikte rij wordt ook opgeslagen, aangezien dit noodzakelijk is voor de Shift-functionaliteit, die een startpunt vereist. Als de Shift-toets is ingedrukt (en er een startpunt aanwezig is), worden alle rijen geselecteerd die zich bevinden tussen de 'lastSelectedRowId'

en de 'userId' van de zojuist aangeklikte rij. Bij een normale klik (zonder ingedrukte toetsen) wordt bij het selecteren van een nieuwe rij de voorgaande selectie opgeheven. De focus ligt dan enkel op die ene, nieuw geselecteerde rij. Het deselecteren van een reeds geselecteerde rij is alleen mogelijk via de selectiekolom (het vinkje).

Tevens is de kleurfunctionaliteit geïmplementeerd. Deze werkt via de functie 'getPatientGroupColor', die vanuit de [ColorService](#) wordt aangeroepen. De logica hiervan is bekend, aangezien dezelfde kleurtoewijzing al werd toegepast in de Parallel Coordinates- en boxplot-visualisaties. Dit zorgt voor een consistente visuele weergave van patiëntengroepen door de hele applicatie heen.

Patient groups	SRR14418862	Non_responder	0.00035	0.08966	0.00869	0.90101
<input checked="" type="checkbox"/> Non_responder	SRR14418863	Non_responder	0.00150	0.24920	0.00308	0.74542
<input checked="" type="checkbox"/> Non_responder	SRR14418864	Non_responder	0.00036	0.12926	0.01066	0.85951
<input checked="" type="checkbox"/> Responder	SRR14418865	Responder	0.00040	0.09618	0.00894	0.89422

Figuur 50: Kleur selectie spreadsheet rij

De binnenrand van elke rij is voorzien van een specifieke kleur. Diverse kleurtoepassingen, zoals het kleuren van de gehele rij of enkel het lettertype, zijn geprobeerd. Deze specifieke aanpak resulteerde echter in het meest esthetische, duidelijke en niet te opvallende resultaat.

3.9. ColorService

Deze service biedt een centrale functionaliteit voor de componenten. Door de implementatie hiervan wordt kleurconsistentie binnen de applicatie gewaarborgd. Dit stelt verschillende componenten in staat dezelfde kleurenpaletten te hanteren, wat resulteert in een consistent en esthetisch geheel.

3.9.1. Mijn bijdrage aan de colorservice

Zoals eerder in het document is besproken, is een vergelijking gemaakt tussen HSV- en HCL-kleuren ([hierover meer](#)). Voor de ColorService is gekozen voor HSV, aangezien de specifieke context geen hiërarchische niveaus omvatte. Er diende wel duidelijk onderscheid te worden gemaakt tussen de verschillende patiëntengroepen. Om deze reden bleek het HSV-kleurenschema geschikter. De gedetailleerde vergelijking kan hier nogmaals worden geraadpleegd.

De gehele logica voor kleurconversie is geïmplementeerd in de ColorService. Hierin zet de `hsvToRgb`-functie ([hierover meer](#)) de HSV-waarden om naar RGB-waarden, waarna de return-statement zorgt voor de transformatie naar een hexadecimale code.

Voor het toewijzen van de HSV-kleuren is vervolgens de functie `assignColorsToGroups()` aangemaakt. Deze is verantwoordelijk voor het dynamisch toewijzen van de kleuren aan de patiëntengroepen.

```
public assignColorsToGroups(groups: string[], baseHueRange: [number, number] = [0, 360]): void {
    const saturation = 0.7;
    const brightnessScale = 0.8;
    const value = brightnessScale * 0.65;
    const numGroups = groups.length;

    this.groupColorMap.clear();

    groups.forEach((group, index) => {
        const hueRange = baseHueRange[1] - baseHueRange[0];
        const segmentWidth = hueRange / numGroups;
        const starthue = baseHueRange[0] + index * segmentWidth;
        const endHue = starthue + segmentWidth;
        const hue = (starthue + endHue) / 2;

        const color = this.hsvToRgb(hue / 360, saturation, value);
        this.groupColorMap.set(group, color);
    });
}
```

Figuur 51: assingColorsToGroups functie colorservice

In deze functie wordt het hue-bereik (baseHueRange) ingesteld op 0 tot 360 graden, wat inhoudt dat het volledige kleurenspectrum kan worden gebruikt. Vervolgens worden de variabelen voor het HSV-kleurenschema gedefinieerd: de saturation (verzadiging) en de value (helderheid). Daarna wordt het aantal groepen bepaald. De interne groupColorMap wordt leeggemaakt om te voorkomen dat eerdere kleuraanduidingen

blijven bestaan. Vervolgens wordt het bereik van 360 graden opgedeeld door het aantal groepen. Bijvoorbeeld, indien er twee groepen zijn, wordt het bereik verdeeld in segmenten van 0-180 graden en 180-360 graden. De uiteindelijke hue (tint) voor elke groep wordt bepaald door het midden van dit segment te nemen. Dit garandeert een groter onderscheid tussen de verschillende groepen. Tot slot wordt de conversie naar RGB uitgevoerd door de `hsvToRgb()`-functie.

Daarnaast is er een functie getColorForGroup(). Deze functie is verantwoordelijk voor het retourneren van de correcte kleur voor een opgegeven patiëntengroep. Hierbij wordt rekening gehouden met zowel de standaard automatisch toegewezen kleuren als de handmatig ingestelde kleuren.

Een andere belangrijke functie is de setCustomColorForGroup(). Via deze functie wordt het mogelijk om handmatig een specifieke kleur in te stellen voor een bepaalde groep.

```
public setCustomColorForGroup(groupName: string, color: string): void {
    this.customColors.set(groupName, color);
    this.colorChanged.next(groupName);
}
```

Figuur 52: setCustomColorForGroup

Wanneer een kleur is ingesteld en opgeslagen in de customColors, stuurt deze functie een notificatie via colorChanged. Hierdoor kan de rest van de applicatie direct reageren op veranderingen.

4. Besluit

Hiermee komt een einde aan mijn realisatiedocument over mijn stage rond de visualisatie van biologische data. Ik heb geprobeerd de applicatie zo goed mogelijk uit te leggen, met daarbij een focus op de onderdelen waar ik zelf het meest aan heb bijgedragen.

Zoals eerder vermeld, werd dit project in duo uitgevoerd. Ik durf met zekerheid te zeggen dat we beiden evenveel inspanning hebben geleverd.

Bij de zaken die ik hierboven heb toegelicht, kwam ik af en toe ook een bug tegen. Door deze op te lossen heb ik geleerd dat het normaal is dat er bugs zijn, zeker in een interactieve visualisatie. Het is dan ook belangrijk om de applicatie veel te testen en zoveel mogelijk uit te proberen, waardoor bugs vanzelf worden ontdekt.

Daarnaast heb ik ook veel tijd besteed aan opzoekwerk en het testen van verschillende mogelijkheden om tot het best mogelijke eindresultaat te komen.

Het project bouwde week na week verder op wat we eerder hadden ontwikkeld. We hadden wekelijks een meeting met onze stagebegeleiders, waarin we onze voortgang presenteerden. Op basis daarvan kregen we telkens nieuwe functies die toegepast moesten worden in de applicatie, waardoor het project voortdurend werd uitgebreid.

Natuurlijk zijn er nog mogelijkheden tot uitbreiding. Zo zouden er extra interactieve elementen kunnen worden toegevoegd, zoals een aparte patiëntenweergave of andere functionaliteiten die men in de toekomst nog kan bedenken. Binnen de gegeven tijdspanne van drie maanden is echter een stabiel en mooi eindproduct opgeleverd.

Tijdens deze stage heb ik ook mijn technische vaardigheden sterk kunnen verbeteren. Vooral mijn kennis van Angular en D3.js is aanzienlijk gegroeid. Vooraf had ik geen ervaring met D3.js, maar door er dagelijks mee te werken heb ik het onder de knie gekregen en voel ik me nu veel comfortabeler met deze visualisatielool.

Ik vond het erg boeiend om stage te lopen in een andere omgeving dan ik gewend ben. Het was een interessante en leuke ervaring om aan de onderzoeksrand van de informatica te werken, in plaats van in een meer productiegerichte context.

Tot slot wil ik mijn stagebegeleiders in Bordeaux van harte bedanken voor hun tijd, hun duidelijke en constructieve feedback, en hun bereidheid om steeds op onze vragen in te gaan.

LITERATUURLIJST

- D3 by Observable | The JavaScript library for bespoke data visualization. (z.d.). <https://d3js.org/>
- Angular. (z.d.). Angular. <https://angular.dev/>
- Wikipedia-bijdragers. (2025, 23 april). HSV (kleurruimte). Wikipedia. [https://nl.wikipedia.org/wikiHSV_\(kleurruimte\)](https://nl.wikipedia.org/wikiHSV_(kleurruimte))
- HCL Wizard. (z.d.). HCL Wizard - Somewhere Over The Rainbow. <https://hclwizard.org/>
- Icicle plots. (z.d.). https://www.cs.middlebury.edu/~candrews/showcase/infovis_techniques_s16/icicle_plots/icicleplots.html
- Brweb. (z.d.). BT.709 Parameter values for the HDTV standards for production and international programme exchange. <https://www.itu.int/rec/r-rec-bt.709>
- Wikipedia contributors. (2025, 16 mei). REc. 709. Wikipedia. https://en.wikipedia.org/wiki/Rec._709

BIJLAGEN

Bijlage 1 – Tree Colors: Color Schemes for Tree-Structured Data

Tennekes, M., De Jonge, E., & Statistics Netherlands. (z.d.). Tree Colors: Color Schemes for Tree-Structured Data. https://10mapz.com/downloads/publications/TreeColors_manuscript.pdf