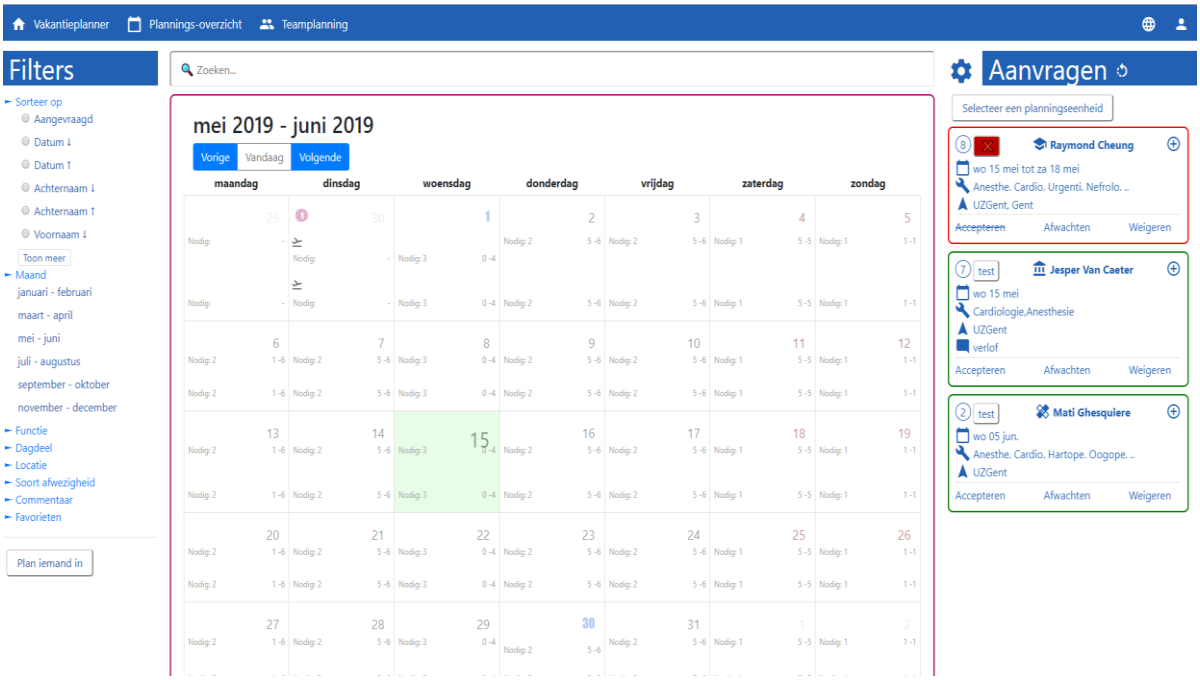
FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

MYSTAFF

VAKANTIEPLANNING EN REGELING

16/05/2019: EINDVERSLAG

Door Alexander Van Nevel, Jesper Van Caeter, Matisse Ghesquière, Raymond Cheung, Robin De Zwaef



http://bpvop5.ugent.be:8081

UNIVERSITEIT
GENT

DANKWOORD

Deze bachelorproef zou niet mogelijk zijn geweest zonder de hulp van een aantal personen.

Eerst en vooral willen we de coaches bedanken. Ze stonden ons bij wanneer de databank en andere componenten zoals docker ons in de steek lieten.
Zonder mevr. Ongenae, mr. Saelens en mr. Macoir zou de databank nog steeds niet operationeel zijn. Ook konden we bij hen terecht met onze moeilijke beslissingen.

Vervolgens willen we Eva Dauw en Frederic Pape van het bedrijf Axians bedanken. Zonder hen zou dit project uiteraard niet kunnen hebben plaatsvinden. Ook gaven ze steeds snel antwoord op onze vele mails in de late uurtjes.

Ten slotte bedanken we onze begeleidster Ann Van Overberge. Zij wees ons steeds de juiste richting doorheen het verloop van het project.

KORTE SAMENVATTING

Axians is een bedrijf dat actief is in de medische sector, meer bepaald in het ontwikkelen van software op maat voor ziekenhuizen en andere zorginstellingen. Voor hun planningstool myStaff is het nog op zoek naar een efficiënte module voor het verwerken van verlofaanvragen in de zorgcentra. Door de vele werknemers, de onderling verschillende competenties per werknemer en de nood aan permanente beschikbaarheid op sommige afdelingen, is het inplannen van vakantie een moeizaam proces. Dankzij onze implementatie zullen dokters op een intuïtieve manier verlofaanvragen kunnen reserveren, waarbij bv. de HR-manager deze m.b.v. een overzichtelijke kalender kan toewijzen. Dokters zullen op deze kalender kunnen zien wanneer ze aanvragen kunnen doen, zodat er geen overlappingen zijn met andere dokters. De planning-beheerder zal dan eenvoudig de meest optimale regeling kunnen vinden opdat alle dokters tevreden zijn. Onze opdracht bestaat eruit om deze tool op een performante, schaalbare en betrouwbare wijze te realiseren.

STRUCTUUROVERZICHT

Inhoudstafel

1 INLEIDING

1.1 Context

MyStaff, van het bedrijf Axians, is een planningstool op maat van noden van medische teams in zorginstellingen en bestaat uit verschillende modules, waaronder een organisatiebeheermodule, een wachtplanningmodule en een dagplanningmodule. Aangezien MyStaff nog geen vakantieplanningmodule beheert, bestond onze opdracht eruit om deze te ontwikkelen en te laten interageren met de bestaande modules.

Een ziekenhuis bestaat gewoonlijk uit een 70-tal dokters, waarbij elke dokter verschillende competenties heeft en op verschillende locaties kan werken. Als deze dokters verlof aanvragen, moet ervoor gezorgd worden dat er tijdens dit verlof op een bepaalde locatie nog dokters met dezelfde competenties aanwezig zijn. Een HR-medewerker kan dan een planning opstellen waarbij zo veel mogelijk verlofaanvragen worden goedgekeurd, rekening houdend met de nodige vereisten. De verantwoordelijke voor de planning zou hierbij handig gebruik kunnen maken van een digitale kalender. Dit rooster geeft een duidelijk overzicht weer van alle aangevraagde en goedgekeurde verlofaanvragen. Het bespaart de planningsverantwoordelijke veel werk en geeft de dokters een overzicht wanneer ze hun aanvragen kunnen plaatsen.

Via de andere modules die onder myStaff vallen, kan een organisatie ingesteld worden. Over de modules heen zijn er drie verschillende rollen, namelijk organisatiebeheerder, planner en teamlid. De organisatiebeheerder kan nieuwe teams aanmaken, de medewerkers beheren en de afgesproken feest -en brugdagen instellen. De planner zorgt voor het interne tijdschema van een team waaraan hij werd toegewezen. Dit houdt in dat deze persoon de structuur in een dienst zal opzetten en de aan -en afwezigheden zal beheren. Een teamlid is dan weer een arts of zorgkundige die deel uitmaakt van een bepaalde dienst. Deze kan zijn eigen planning verzorgen en controleren, maar kan ook de voorlopige kalender van zijn of haar team bekijken.

1.2 Probleemstelling

Momenteel heeft myStaff nog geen module voor het beheren van de individuele verlofdagen van zorgkundigen binnen een organisatie. Deze module zal moeten samenwerken met de andere modules van myStaff. Als een arts een aanvraag doet, dan zal de geldigheid van deze aanvraag getoetst worden d.m.v. een *constraint solver* van myStaff. Dit zijn beperkingen die opgelegd worden aan de aanvraag, zoals het feit dat shiften recupdagen kunnen hebben, waardoor een persoon enkele aansluitende dagen niet mag werken. Ook zijn er simultane shifts. Wanneer twee simultane shifts op dezelfde dag vallen, mogen personen niet in beide shifts tegelijk worden ingepland die dag. Zijn twee shiften niet simultaan, dan is dat wel oke.

De aanvraag van afwezigheden moet eveneens een bepaalde flow volgen. Een verzoek tot verlof heeft eerst de status ‘in overweging’ en wordt na evaluatie goedgekeurd of afgekeurd. De arts moet vervolgens verwittigd worden via mail om op de hoogte gesteld te worden over de beslissing van hun aanvraag. Eenmaal de aanvraag goedgekeurd is, kan deze natuurlijk ook weer ingetrokken worden door de arts.

De afwezigheden worden onderverdeeld in verschillende types, zoals jaarlijks verlof, part-time verlof of ziekte. Er wordt niet alleen onderscheid gemaakt tussen de types, maar ook de duur is belangrijk.

Gaat dit om een halve dag, een volledige dag of een reeks dagen? Aan iedere zorgkundige wordt eveneens een vakantiesaldo toegewezen. Deze teller houdt bij hoeveel verlofdagen nog opgenomen kunnen worden en moet gesynchroniseerd blijven met de voorlopige kalender.

1.3 Doelstelling

De opdracht bestaat eruit om al deze vereisten te implementeren in een webgebaseerde vakantieplanner. De website moet een intuïtieve user interface hebben voor de zorgkundigen, zodat aanvragen simpel doorgegeven kunnen worden. Voor de planner moet het geheel vooral overzichtelijk zijn, aangezien hij een globaal inzicht moet krijgen in een ideale ordening voor alle artsen.

De applicatie zal een goed samenwerkend geheel moeten vormen, waarbij alle componenten hun functie correct uitvoeren. De dokter zal zijn aanvraag moeten kunnen doorgeven aan een database, de planner zal deze aanvraag uit de database moeten halen en evalueren, de dokter moet hierna geïnformeerd worden over de beslissing en indien nodig zijn planning aanpassen... En dit alles moet gebeuren via een tussenliggende server zodat iedereen vanop afstand kan werken. Als er een fout voorkomt bij één van deze componenten, zal de applicatie niet werken. Dit vergt een grote nauwkeurigheid, waarbij het grondig testen van de individuele componenten noodzakelijk is.

Bij dit project zal ook gebruik worden gemaakt van de huidige standaarden in de informatica-wereld. Sommige van deze technologieën zullen ons nog onbekend zijn, maar het is dan ook de bedoeling dat hier veel over wordt bijgeleerd en deze na wat opzoekwerk kunnen worden geïmplementeerd in de applicatie.

De kalender zal sorteerbaar, betrouwbaar en inpasbaar moeten zijn. Sorteerbaar, zodat er geordend kan worden op competentie, locatie of personen die behoren tot een zekere categorie. Betrouwbaar, want als er een fout voorkomt bij het toekennen van verlof, zal er mogelijks geen arts aanwezig zijn waar nodig. En tot slot inpasbaar, aangezien de kalender ingevoegd moet worden in de bestaande myStaff website en zal moeten samen werken met de andere modules.

1.4 Structuur verslag

Eerst zal er een korte introductie zijn over de noodzakelijke functies van de applicatie. Wat moet de website allemaal kunnnen, welke knoppen zijn hier voor nodig, hoe zal dit allemaal worden voorgesteld, welke problemen zullen hier aan bod komen... Aangezien er veel beperkingen worden opgelegd aan de dagplanning, werd er gebruikt gemaakt van een backtracking algoritme. Dit wordt hier ook uitvoerig besproken. De high-level requirements worden verduidelijkt aan de hand van use case diagrammen en een tabel waarin alle features opgelijst staan. Voor elke sprint worden hier enkele onderwerpen uit gekozen die klaar moeten zijn tegen de deadline. Zo wordt het project stelselmatig opgebouwd en kan de klant zien of de website de juiste richting ingaat.

Vervolgens wordt er wat dieper ingegaan op de gebruikte technologieën. Er wordt beschreven welke programmeertalen er gebruikt werden, zowel voor de front end als de back end. Hoe worden de objecten weggeschreven naar de databank en hoe worden de gegevens eruit gehaald? Ook wordt er uitgelegd hoe de applicatie gestart kan worden. Deze zaken worden verduidelijkt a.d.h.v. een deployment diagram, een klassendiagram, een sequentiediagram en een database diagram.

Hierna komen de testen aan bod. Welke testen werden gebruikt, op welke componenten werden deze toegepast... Deze testen bespreken het gebruik van de web API, de connectie met de databank, de werking van de zelfgeschreven klassen en de werking van het algoritme.

Er is ook een evaluatie van het project voorzien. Waar vormden zich problemen, hoe hebben wij dit aangepakt en hoe kunnen wij dit in de toekomst voorkomen?

Ten slotte wordt uitgelegd hoe de klant de applicatie kan installeren en gebruiken. Hoe krijgt men de website draaiende, hoe wordt de connectie gemaakt... Er is ook een handleiding voorzien voor de gewone gebruiker, zodat zowel de dokter als de planner eenvoudig van de vakantieplanner gebruik kan maken.

2 GEBRUIKERSASPECTEN

2.1 High-level requirements

Om het de planner makkelijk te maken, werd gekozen voor een tweemaandelijks schema van alle ingenomen verlofaanvragen. De kalender kan worden ingesteld in een dag, week of maandoverzicht. Op iedere dag worden alle artsen afgebeeld die dan op verlof zijn. Hierbij worden enkele kenmerken van de arts meegegeven, zoals zijn competenties, de locaties waarop hij werkt, het type van zijn verlof... Op deze manier kan de planner eenvoudig zien of er op een bepaalde locatie een minimumaantal artsen is met een benodigde competentie. Het is niet de bedoeling dat elke dag vol staat met tekst, want dan is dit niet meer duidelijk voor de planner. Daarom zullen wij werken met symbolen in de kalender. Indien de planner meer uitleg wil, kan daarop gedrukt worden om een gedetailleerde representatie weer te geven. Het weergeven van vakanties en feestdagen is eveneens een belangrijke vereiste. Indien er een conflict zou zijn tussen planner en dokter, kunnen de vorige aanvragen geraadpleegd worden. De planner kan elke aanvraag openen, evalueren en hier een beslissing over nemen. Indien nodig is er ook een functie voorzien om manueel verlof in te plannen, zonder dat een arts hier aanvraag voor heeft moeten doen. Al deze aanvragen kunnen ook gefilterd worden op naam, datum, soort verlof...

Een zorgkundige ziet een soortgelijke kalender met een simpelere layout. Sommige functies zijn afgeschermd aangezien voor een zorgkundige niet dezelfde priviliges gelden als voor een planner. Een arts kan manueel via een knop zijn verlof inplannen of rechtstreeks op de kalender zijn gekozen dagen aanklikken. Naast de gekozen data wordt ook gevraagd naar het type afwezigheid (ziekte, jaarlijks verlof...), de periode en een mogelijke opmerking. Indien de dokter over zijn jaarlijks maximum aan verlofdagen zit, zal er een foutmelding volgen. Vervolgens wordt deze aanvraag weergegeven in de kalender en wordt de planner getriggerd via mail. Tijdens het verloop van dit verzoek tot verlof zal de arts de status ervan kunnen opvragen. Indien nodig zal de arts zijn aanvraag ook terug in kunnen trekken of wijzigen. Een zorgkundige zal eveneens de planning kunnen zien van de teams waartoe hij/zij behoort, zodat het zijn verlofaanvragen hieraan kan toetsen.

Bij elke aanvraag wordt het tijdstip meegegeven wanneer deze werd verstuurd. De planner kan dan het ‘first come, first serve’ principe toepassen. Alle zorgkundigen hebben een teller die bijhoudt hoeveel verlof nog kan worden opgenomen. Op deze manier kan de planner zien hoeveel verlofdagen een medewerker nog zal aanvragen.

De applicatie heeft een gedeelde databank nodig om de lopende aanvragen in te bewaren. Zo zullen de aanvragen van de artsen gesynchroniseerd zijn met de lopende aanvragen die de planner te zien zal krijgen.

2.1.1 Hoeveel mensen zijn er inzetbaar op een bepaald deel van de dag?

2.1.1.1 Probleemstelling

Opdat een planner zou kunnen bepalen of personen op verlof kunnen, dient hij te weten hoeveel mensen op een bepaalde dag inzetbaar zijn. Stel dat er bijvoorbeeld op een dag 5 mensen moeten aanwezig zijn met bepaalde competenties: wanneer de planner ziet dat er slechts 5 mensen beschikbaar zijn met de nodige skills en één iemand daarvan verlof vraagt, dan weet hij dat hij dat

moet afkeuren. Anders zou de shift natuurlijk niet meer op te vullen zijn. Zijn er een extra 7 mensen inzetbaar, dan kan de planner gerust het verlof goedkeuren.

Op zich lijkt dit een simpel te verwezenlijken probleem, maar er schuilen vele - soms subtiele - problemen achter.

Om het probleem op te lossen zijn voor elk dagdeel de gegevens over de shift nodig. Die worden aangeboden via de web API van Axians. Eén shift geldt gedurende een volledige dag. Elke weekdag krijgt een aparte shift. Er zijn meerdere shiften per dag.

Een shift heeft volgende gegevens:

- Number: Dit is een natuurlijk getal dat aangeeft hoeveel personen er exact nodig zijn op een dag. Dit is dus ook het aantal personen dat zal moeten worden ingepland, niet meer, niet minder.
- Required skills en alternative skills: dit zijn de competenties waarover dokters in deze shift moeten beschikken. Belangrijk is dat elke dokter over ofwel alle competenties die required zijn beschikt, ofwel over alle alternative skills. Zo mogen er bijvoorbeeld 2 dokters met de volledige required skillset en 3 dokters met de alternative skillset worden ingepland. Ook 5 dokters met de required skillset en 0 met de alternative skillset zijn in orde. Wat echter minder triviaal lijkt, is dat ook 0 mensen met de required en 5 mensen met de alternative skillset zijn toegelaten.
- Free period days: Dit is een natuurlijk getal dat aangeeft hoeveel recup een persoon krijgt wanneer hij die shift invult. Wanneer deze waarde bijvoorbeeld 3 is, en de persoon werkt op een zondag, dan zal hij maandag, dinsdag en woensdag niet mogen ingepland worden.

Bijkomend is het feit dat sommige shiften simultaan zijn en andere niet. Als twee shiften niet simultaan zijn, kunnen ze door dezelfde persoon op hetzelfde moment beide worden ingevuld. Dit geldt per dag. Als shift 1 en shift 2 simultaan zijn, mag een persoon natuurlijk wel bv. maandag in shift 1 worden ingepland en donderdag in shift 2.

Simultane shifts worden opgegeven in lijstjes. Bijvoorbeeld: shift 1 en shift 2 zijn simultaan, alsook shift 2 en 3. Belangrijk hierbij is dat er niet vanuit mag gegaan worden dat shift 1 en 3 ook simultaan zijn. Mensen kunnen dus zowel in shift 1 als in shift 3 tegelijk worden ingepland.

Wanneer mensen op verlof zijn, kunnen ze natuurlijk niet worden ingepland.

2.1.1.2 Oplossing

Wanneer shiften met elkaar geen rekening moeten houden, ligt de oplossing voor de hand. Dan moet er enkel geteld worden hoeveel mensen er zijn met de juiste skillset voor een dagdeel, en het probleem is opgelost.

Nu zorgt echter het inplannen van een persoon in sommige shifts dat deze persoon niet in andere shifts mag worden ingepland. Dit is het geval in niet-simultane shifts en shifts die voor recupdagen zorgen. Deze recupdagen zorgen voor scenario's die op het eerste zicht vergezocht lijken, maar weldegelijk kunnen voorkomen. Stel: shift 1 zorgt voor 2 dagen recup, en valt op maandag en vrijdag. Shift 2 zorgt ook voor 2 dagen recup en valt op woensdag. Wanneer een persoon nu ingepland wordt op maandag, dan kan hij woensdag niet aanwezig zijn, maar wel vrijdag, hij kan dus 2 shiften draaien. Wanneer hij echter woensdag wordt ingepland, vallen zowel maandag, als vrijdag uit de boot. Er moet dus niet enkel gekeken naar de dagen die wegvallen door de te bekijken shift, maar ook naar de dagen waardoor de te bekijken shift wegvalt.

Cruciaal is dus dat het feit dat shifts aan elkaar gelinkt zijn, ervoor zorgt dat men een systeem krijgt met geheugen. Inplannen van een persoon op shift x kan invloeden hebben op het inplannen ervan op shift y. De enige manier om een oplossing te vinden is dus door effectief mensen zogezegd in te plannen en te kijken welke resultaten dat levert.

Om te zorgen dat dit alles correct verloopt wordt het volgende gedaan om een deeloplossing te vinden. Het probleem is hiermee niet opgelost, maar het maakt het probleem al iets minder moeilijk.

1) Maak een tijdelijke lijst van werknemers die op een dag kunnen worden ingevuld, door enkel naar de skills te kijken, houd nog geen rekening met meer ingewikkelde beperkingen. Maak ook al een finale lijst die voorlopig leeg is.
2) Vul shifts op als het aantal mogelijke mensen erin gelijk is aan de bezetting. (m.a.w: er is geen reserve.) En voer stap 3 uit per persoon.
3) Verwijder deze mensen uit shifts die volgens de regels niet kunnen samengaan (door niet-simultaniteit of recupdagen) en uit de lijst van mogelijkheden waar ze al ingepland zijn.
4) Houd voor elke mogelijke persoon in elke shift een lijst bij van shiften die niet meer mogelijk worden indien de persoon hier wordt ingepland.
5) Is deze lijst leeg voor een bepaald persoon, en is het maximaal aantal personen nog niet bereikt, dan mag deze uiteraard worden ingevuld.

   Herhaal tot er geen effect meer is.

Opmerking: oorspronkelijk werd er nog een extra stap toegevoegd, namelijk:

- Wanneer mensen over enkel de nodige of alternatieve skillset bezitten (dus niets meer en niets minder) en de shift zit nog niet vol, dan mogen ze worden ingevuld op die datum in die shift. De logica hier is dat de shift toch moet worden volgepland, en er zo geen mensen verloren gaan met andere skills.

Dit is echter foutief. Het houdt geen rekening met recup of simultane shifts. Wanneer men ook hierop zou testen, zou er nog steeds clustering ontstaan, omdat eerst bepaalde data volledig worden opgevuld met deze mensen, en niet datum per datum.

Nu komt het moeilijke deel. De mensen die nog niet ingepland zijn, beïnvloeden steeds andere shiften. De enige manier om de oplossing te vinden is nu alle mogelijkheden afgaan, en kijken wat de beste oplossing biedt. Dit kan op een redelijk efficiënte manier gebeuren via backtracking.

Dit gebeurt als volgt:

Eerst wordt een shift genomen met het minst aantal personen mogelijk. Het is dus mogelijk dat er slechts één extra persoon nodig is in een shift en er nog 4 mogelijk zijn, toch zal dus de voorkeur uitgaan naar een shift waar 2 personen nodig zijn en er 3 mogelijk zijn. Dit omdat elke persoon moet worden getest. Deze personen worden één voor één getest. Dit noemen we personen van niveau 1. Vervolgens wordt gekeken of dit ergens anders personen in een shift verplicht, indien dit het geval is, worden deze personen ingepland. Vervolgens test men iemand anders van de shift die nu het minst aantal personen bevat (op die van niveau 1 na natuurlijk). Dit noemt men niveau 2. Zo vormt zich een soort boom.
Wanneer in bv. niveau 3 een foute planning bekomen wordt doordat een shift niet meer kan worden ingevuld, wordt de volgende persoon van niveau 2 getest. Wanneer alle personen van niveau 2 getest zijn, zal de volgende persoon van niveau 1 getest worden.

Nu zijn er twee manieren om een backtracking algoritme te implementeren. Men kan ervoor kiezen om de eerste oplossing die bekomen wordt te kiezen als juiste oplossing. Wat men ook kan doen is zoeken naar alle oplossingen en de “beste” bepalen. Deze tweede methode is de manier die hier zal verkozen worden.[1] Nu moet echter “beste” nog een betekenis krijgen in deze context.

Enerzijds kan gekozen worden om het aantal personen die nog niet zijn ingepland te tellen en de oplossing die zo in totaal het meeste personen mogelijk houdt, te kiezen. Langs de ene kant biedt deze manier een enigszins vervormde kijk op de zaken, doordat ze mensen zal weergeven op bv. zowel de dag van een shift met recup als de dag erna, terwijl dit niet kan voorvallen. Langs de andere kant ziet de planner wel als op één dag iemand nodig is, dat de persoon daar nog mogelijk is. Er wordt dus een berekende waarde getoond die steeds gelijk of hoger is dan de werkelijke waarde

Wanneer wordt gekozen om alle personen in te plannen, boven de maximumbezetting, en dat aantal extra ingeplande personen te tellen, krijgt men een realistischer beeld. Zo is men zeker dat de verkregen aantallen zeker kunnen en dat het resultaat 100% betrouwbaar is.Verkregen personen inplannen, verandert de waarden op andere dagen niet meer negatief. Anderzijds, wanneer de planner bijvoorbeeld maandag iemand verlof wilt geven, is het mogelijk dat er geen personen meer reserve zijn omdat deze zondag staan ingepland als reserve en die shift recup had. De berekende waarde die wordt weergegeven is dus steeds gelijk of lager dan de werkelijke waarde.

Hier zal gekozen worden voor de tweede methode.

[1] De applicatie bevat ook andere toepassingen waarvoor de andere methode zal worden gekozen, zoals bv bepalen of een HolidayMessage mag worden goedgekeurd of niet.

2.1.2 Gegeven een lijst met verlofaanvragen, welke mag ik goedkeuren?

2.1.2.1 Probleemstelling

In principe wil een planner in eerste instantie zien of hij een verlofaanvraag wel of niet mag goedkeuren. Het zou dus helpen moest hij voor elke verlofaanvraag kunnen zien of die voor problemen zorgt als hij ze zou goedkeuren of niet. Wanneer bv. een verlofaanvraag nooit kan worden goedgekeurd, zouden deze worden weergeven in het rood. Wanneer deze nooit voor problemen zorgt, worden deze in het groen gezet. Wanneer deze enkel in combinatie met andere verlofaanvragen voor problemen zorgt, zullen die worden weergeven in het oranje.

2.1.2.2 Simpele, maar onvoldoende oplossing

In een eerste implementatie doet men 1 of n+2 testen voor n HolidayMessages. Eerst test men of alle aanvragen die een planner kan goedkeuren, samen zorgen voor een planning die niet meer klopt. Indien dat lukt, kan men al die aanvragen in het groen weergeven en zeggen dat ze allen veilig in te plannen zijn.
Lukt dat niet, dan kan men voor elke aanvraag apart testen of ze zorgen dat de planning kapot gaat. Gebeurt dit, dan geeft men ze weer met een rood kleurtje. Die verloven mogen nooit worden ingepland.
Nu komt echter het tricky gedeelte. Men kan nu testen of de resterende verloven, zonder die dat foutlopen op zichzelf, samen foutlopen. Lopen ze niet fout, dan kan men ze wederom in het groen weergeven, want ze zullen nooit voor problemen zorgen. Is er echter wel een fout, dan betekent dit dat geen van deze verloven op zichzelf een probleem vormt, maar allemaal samen wel. Er is dus een combinatie van verloven die foutloopt. Omdat het onmogelijk is qua efficiëntie om alle combinaties af te gaan, worden ze in het oranje weergegeven, met de staat “unknown”.
De staat “unknown” zegt natuurlijk niet veel. Wat in principe nodig is, is een lijst met opgesomd: “deze en deze verlofaanvraag botsen met elkaar”. Het simpelste voorbeeld wanneer dat voorvalt is wanneer er op een dag één persoon nodig is. Twee personen hebben de kwaliteiten om dan te worden ingevuld, maar ze vragen beide verlof aan. Eén persoon kan verlof nemen, geen probleem. Beide kan echter niet. Ze vormen samen een combinatie die niet samen op verlof mag gaan.

2.1.2.3 Betere oplossing

Om het probleem voor te stellen, zullen we gebruik maken van een array met getalletjes. Elk getalletje staat voor een groep verlofaanvragen dat niet samen mag worden goedgekeurd. Belangrijk: de elementen in de array zijn niet gesorteerd op datum.

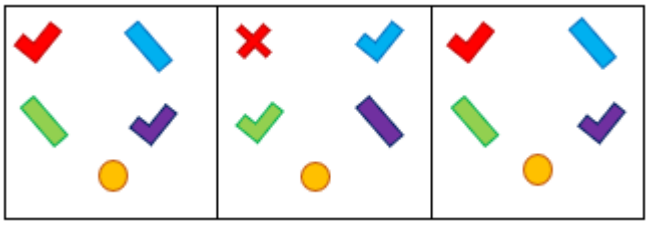
1,2	2		3	1,3	4	4,6	5,6	5		1
A	B	C	D	E	F	G	H	I	J	K

Elementen zonder getalletjes zorgen nooit voor problemen.
Constructies zoals FGHI lijken overdreven, maar ze zijn relatief makkelijk uit te leggen. Stel twee personen vragen op eenzelfde dag verlof en maar één van hen kan ook daadwerkelijk verlof nemen, dan hebben we groep FG.
Stel dat er nu ook drie dagen achter elkaar een shift zich voordoet met telkens één dag recup. Wanneer een persoon een dag werkt, mag hij de volgende dus niet werken. Er zijn telkens twee personen nodig en er zijn vijf mensen die deze shift kunnen doen. We nemen maandag t.e.m. woensdag als voorbeeld.
Dinsdag neemt persoon één verlof. Hij wordt maandag en woensdag ingepland.
Persoon twee en drie worden woensdag ingepland.

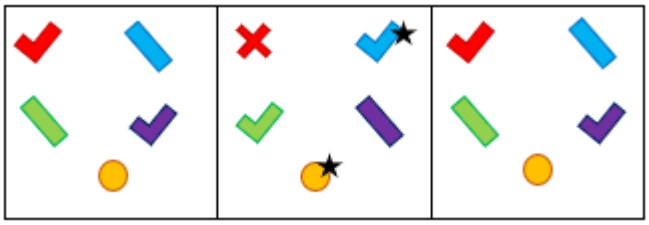
Persoon vier wordt net als persoon één op woensdag ingepland.
Persoon vijf wordt niet ingepland.
Als persoon twee nu op een bepaalde dag verlof aanvraagt, kan persoon vijf invallen. Wanneer echter persoon vijf en persoon twee op dezelfde dag verlof aanvragen, vormt dit een oranje groep.
Wanneer persoon één nu woensdag ook verlof zou aanvragen, en persoon twee vraagt dinsdag verlof aan, dan is dit ook een oranje groep.
Het wordt dus snel ingewikkeld. Hieronder enkele grafische weergaven. Elk kleurtje staat voor een persoon:

- Een vinkje betekent dat de persoon is ingepland
- Een kruisje betekent dat de persoon verlof neemt
- Een schuin streepje betekent dat de persoon niet kan worden ingepland door recup constraints
- Een bolletje betekent reserve
- Een sterretje bij het symbool staat voor een verlofaanvraag die nog niet is goedgekeurd

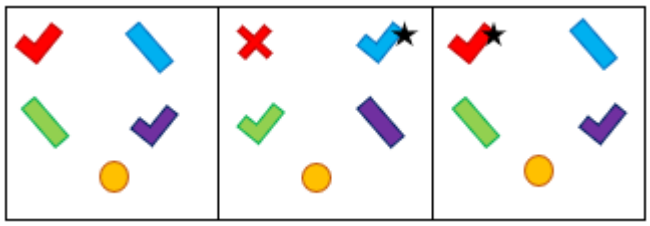
In Figuur 1 neemt enkel persoon 1 dinsdag verlof. In Figuur 2 vragen persoon twee en vijf beide verlof, maar door de recupregels kunnen ze niet beide worden goedgekeurd. In Figuur 3 vraagt persoon twee op dinsdag verlof aan en persoon een op woensdag (nadat hij op dinsdag al verlof kreeg). Ook deze twee vormen een oranje groep doordat persoon vijf slecht één keer kan invallen door de recupregels.



Figuur 1: enkel persoon 1 neemt dinsdag verlof.



Figuur 2: persoon twee en vijf vragen beide verlof



Figuur 3: persoon twee doet dinsdag een verlofaanvraag en persoon een woensdag

Door deze constructies met recupdagen en dan nog simultane shiften erbij, is het dus mogelijk dat zeer ingewikkelde groepen ontstaan. Belangrijk voor het algoritme is wel dat we ervan uitgaan dat een verlofaanvraag nooit tegelijk in drie groepen zit. Dit lijkt een correcte voorspelling.

Ook cruciaal aan het begrijpen van het algoritme is dat onze test neerkomt op: is deze planning nog steeds mogelijk indien we deze personen goedkeuren: ja of neen? We krijgen enkel een ja als

minstens één **volledige** groep voorkomt in de te testen groep. Analoog krijgen we enkel een nee als **geen enkele volledige** groep voorkomt in de te testen groep. Dus de hoofdvraag is: “Bevindt er zich geen volledige groep in de te testen subgroep”.

Nog een aspect dat belangrijk is, is dat er geen groepen meer voorkomen van grootte 1. Het onvolledige algoritme van 2.1.2.2 wordt eerst toegepast, zodat alle groepen van grootte 1 reeds gevonden zijn. Dit algoritme gaat dan verder op de rest van de HolidayMessages die de staat “UNKNOWN” hebben.

Het algoritme werkt als volgt:

1) Start met het testen van de eerste 2 elementen, indien niets gevonden, vergroot dan steeds met één extra element tot de groep crasht.
2) Indien de groep grootte 2 was, is de groep gevonden en kan men verder met stap 4.
3) Indien de groep groter was dan 2, willen we de kleinste groep vinden die crasht. Omdat de groep pas crashte wanneer het laatste element werd toegevoegd, in stap 1, weten we dat dit element zeker deel is van de groep. We testen nu de volledige gevonden groep ZONDER telkens elk element te testen. Stel we hebben een groep van 5 elementen, ABCDE, en de te vinden groep is DE. dan testen we in die volgorde: BCDE, ACDE, ABDE, ABCE en dus **niet ABCD**. Crasht de groep nog steeds voor een bepaalde subgroep, dan weten we dat het weggelaten element niet behoorde tot de te zoeken groep. Dit lijkt counter intuïtief. De planning zal nu nog mogelijk zijn in ABCE want de volledige groep DE komt niet voor. Dus kunnen we besluiten dat DE de groep is.
4) Wanneer nu de groep van stap 1 groter zou gemaakt worden, zou die altijd crashen omdat de gevonden groep er in aanwezig is. De oplossing is simpel. Test vanaf nu verder zonder de eerste verlofaanvraag van de gevonden groep.
5) Maak nu de groep weer groter tot hij crasht. Dit zal plaatsvinden in de voorbeeldarray bij BCDE (A werd eruit gelaten)
6) De weggelaten elementen blijven nog steeds weg. Laat nu in de groep opnieuw eens elk element weg. Waar de groep niet crasht, vindt men weer de nieuwe groep.
7) Herhaal stap 5 en 6 tot je op het einde bent van de array.
8) Nu zijn de meeste groepen gevonden (zijnde: AB, DE, FG, GH en HI). Groep 1, AEK werd echter niet gevonden. Dit omdat A al zeer snel werd uitgeschakeld in het zoekproces. Het probleem is dus te wijten aan het feit dat telkens het eerste element van elke groep werd uitgeschakeld. Er zit dus niets anders op dan elke permutatie af te gaan van de mogelijkheden van uitschakelen van een bepaalde dag van de groep.
   De gevonden groepen in dit voorbeeld zijn per toeval allemaal 2 elementen lang. We zullen de permutaties voorstellen met binaire getallen. Tot nu toe werd steeds het eerste element weggelaten uit elke groep, er zijn 5 groepen, dus kan dit voorgesteld worden als permutatie 00000.
   Belangrijk is dat de meeste groepen zijn gevonden. Er kan dus worden vanuit gegaan dat we verwachten dat we nog weinig groepen zullen vinden. Daarom zullen er minder testen worden uitgevoerd dan in het eerste deel van het algoritme.
   We testen nu de volgende permutatie: 00001 (hierbij zijn ABFGH uitgeschakeld). Dit doen we door de volledige overblijvende groep te testen (dus BCEIJK). Deze zal niet crashen, want er zit geen volledige groep in. Dus kan de volgende permutatie worden getest.
   Om de performantie te verhogen, wordt bijgehouden welke lijsten van verloven al zijn getest. Het is perfect mogelijk dat de permutatiecode van twee groepen verschilt, maar ze toch hetzelfde zijn omdat er overlap is, of dat de ene groep de andere omvat. Zo is bv na het testen van 01010 (AEFH) , de test op 010111 (AEFHI) overbodig, want de resterende groep

verloven in het tweede geval is een subgroep van dat bij het eerste geval. Het omgekeerde is niet waar.

9) Wanneer een crashende groep is gevonden, wordt terug de volledige groep eens getest zonder elk elementje. Hetzelfde principe als in stap 3 geldt.
10) Nu heeft men echter een nieuwe groep gevonden, dus gaan de te zoeken getallen nu van 000000 tot 211111. Omdat het bijkomen van een nieuwe groep het aantal overblijvende elementen kleiner maakt, zullen veel van de nieuwe groepen al onderzocht zijn.
11) Wanneer stap 8 t.e.m. 10 herhaald zijn en alle permutaties zijn afgegaan, is het zeker dat alle groepen gevonden zijn.

Opmerkingen

- Alle mogelijke combinaties afgaan zou 2^11 keer het berekenen van een planning vereisen, dat is onmogelijk, gezien dat deze functie telkens een seconde duurt.
- Deze methode voert 46 +- 10 (foutmarge door groepen over het hoofd te zien) keer de trage functie uit voor de laatste groep gevonden wordt. Het is zeer lastig om gewoon op het zicht de groepen te vinden die niet moeten worden getest doordat ze al getest zijn. Een overschatting van het probleem lijkt 100 keer te zijn in dit geval. (Groep 1 werd gevonden in permutatie 10000, dus 0 00000- 0 10000, 1 00000- 1 10000 en 2 00000-10000 moeten sowieso niet getest worden).
- Indien men te veel groepen vindt in fase één van het probleem (tot stap 7), kan men ervoor kiezen om het algoritme vroegtijdig te stoppen, de gevonden combinaties te geven en de overige aan te duiden als unknown.
- Het is zeer moeilijk om te testen onder welke randvoorwaarden dit algoritme goed genoeg presteert. Het creëren van deze situaties met de juiste personen die op de juiste dagen verlof nemen is zeer moeilijk om te zetten naar een simpele voorstelling met getallen. Het is dan ook nagenoeg onmogelijk tenzij in de praktijk om dit algoritme te testen.
- Zie bijlage 1 en 2 voor een visuele voorstelling van het algoritme.
- Het algoritme is ook in een animatie gegoten en in een PowerPoint geplaatst op gitHub.
- In praktijk gaat het natuurlijk niet gewoon om getalletjes, maar om verlofaanvragen. daarom wordt er nog een belangrijke optimalisatie gedaan. Aanvragen worden nog eens in groepjes verdeeld waartussen minstens het maximum aantal recupdagen van alle shiften zit. Deze groepen kunnen geen invloed op elkaar hebben. Ook begin -en einddatum van de te testen periode hangen af van het maximaal aantal recupdagen. Hierdoor verkleint de backstack van de backtracker die planningen maakt drastisch.

2.2 Use case diagram

Voor het eerste contact met de klant werden enkele functionele vragen opgesteld om te bespreken wat geïmplementeerd moet worden in de applicatie. De antwoorden op deze eerste functionele vragen alsook extra wensen van de klant, werden neergeschreven in een Use Case Diagram. Merk op dat een planner ook de rol dokter draagt in dit schema. Dit omdat hij ook zijn eigen verlof moet kunnen zien.

Figuur 4: Use Case Diagram

2.2.1 Detail-uitwerking

Naam: Verlof aanvraag
Doelstelling: Een dokter vraagt verlof aan gedurende bepaalde dagen. Vervolgens wordt dat verlof naar de planner gestuurd om het al dan niet goed te keuren. De dokter ontvangt een bevestigingsmail.
Primaire actor: Dokter
Secundaire actor: Planner
Precondities: De dokter is ingelogd.
Postcondities: De planner krijgt te zien dat de dokter een aanvraag heeft ingediend. De dokter ziet dat zijn verlofaanvraag verzonden is.

Succes scenario:

1. De dokter gaat naar de pagina /calendar (eventueel via de knop in de hoofdingsbalk)
2. De dokter klikt op de startdag van het verlof dat hij wenst aan te vragen.
3. De dokter geeft alle informatie in omtrent het verlof: einddatum, type (jaarlijks verlof, Europees verlof, …), de dagdelen die hij wenst op te nemen, en hij plaatst eventueel commentaar.
4. De aanvraag wordt nu in de database opgeslagen en de planner krijgt een mailtje dat hij kan beslissen over de nieuwe aanvraag. Ook de dokter krijgt nu een bevestigingsmail.
5. De dokter krijgt als status 'voorlopig' te zien op de dagen waarop hij verlof aanvroeg, dat de aanvraag nog niet bekeken is door de planner.
6. Wanneer de planner een beslissing maakt, krijgt de dokter een mailtje met het resultaat van de keuze van de planner.
7. De dokter ziet nu in zijn kalender en in zijn updates component onderaan de staat van het aangevraagde verlof.

Alternatief scenario:

2.a. De dokter klikt bovenaan op "Plan afwezigheid". De begindatum staat nu ingesteld op vandaag en moet nog aangepast worden door de dokter.

Naam: Planningsbeslissing
Doelstelling: De planner maakt een beslissing over de verlofaanvraag van een dokter.
Primaire actor: Planner
Secundaire actor: Dokter
Precondities: De planner is ingelogd. Hij is weldegelijk planner van een bepaalde planningseenheid. Er zijn al dokters die verlof aanvroegen.
Postcondities: De dokter die de aanvraag indiende, krijgt een mailtje. Hij ziet nu ook dat de aanvraag al dan niet is goedgekeurd.

Succes scenario:

1. De planner gaat naar de pagina /newsfeed. (eventueel via de knop in de navigatiebalk)
2. De planner ziet rechts dat er nieuwe aanvragen zijn.
3. De planner kan aanvraag per aanvraag bekijken en eventueel goedkeuren.
4. De planner kan meer info per aanvraag beschikbaar maken door op het plusje te klikken.
5. De planner controleert of de aanvraag mogelijk is:
    a. Via het kleurtje rond de aanvraag ziet de planner of er problemen zijn wanneer deze aanvraag wordt goedgekeurd. Is ze groen? Dan is er sowieso geen probleem. Is ze rood? Dan zorgt het goedkeuren van deze aanvraag altijd voor problemen, ze mag en kan niet worden goedgekeurd. Om ervoor te zorgen dat de planner deze aanvragen niet per ongeluk kan goedkeuren wordt de knop om de aanvraag goed te keuren uitgezet. Vormt ze samen met een andere aanvraag een oranje groep? Dan kan één van deze aanvragen niet worden goedgekeurd.
    b. Via de getalletjes die zich in de kalender bevinden. Het linkse getal op een dag geeft aan hoeveel mensen er nodig zijn die dag. Rechts staat dan x-y. x geeft aan hoeveel mensen er minimum zonder problemen kunnen worden op verlof gestuurd. Dit is 0

wanneer een bepaalde persoon sowieso niet op verlof mag. De y geeft dan aan hoeveel mensen maximum op verlof kunnen worden gestuurd.
    c. Via de filters die links beschikbaar zijn, kan gefilterd worden op skills, locaties, …
    d. Via favorieten te selecteren als mogelijke boosdoeners.
    e. Via de zoekbalk bovenaan die live wordt geüpdatet als je zoekt op welk attribuut dan ook
6. De planner beslist of de aanvraag mogelijk is en keurt ze al dan niet goed.
7. De dokter in kwestie krijgt nu een mailtje en ziet dat de staat van zijn aanvraag veranderd is.

Naam: Verlofwijziging
Doelstelling: Een reeds aangevraagde verlofaanvraag wijzigen
Primaire actor: Dokter
Precondities: De dokter vroeg reeds verlof aan en wil deze wijzigen.
Postcondities: De verlofaanvraag is gewijzigd en wordt doorgestuurd naar de planner om al dan niet goed te keuren.

Succes scenario:

1. De dokter surft naar de pagina /calendar (eventueel via de knop in de hoofdingsbalk)
2. De dokter klikt in de kalender op een dag waar een door hem aangevraagde aanvraag aanwezig is.
3. Er verschijnt nu een zwart balkje waarin hij op “Wijzig” kan klikken.
4. De dokter klikt op “Wijzig”, het formulier zal aan de rechterkant automatisch openklappen met een aanduiding dat de gebruiker zich in de ‘wijzig’ modus bevindt.
5. De dokter geeft rechts de nieuwe gegevens in en klikt op “Aanvragen”.
6. De dokter ziet nu onderaan bij “Updates” dat er een historiek is toegevoegd aan zijn aanvraag. Hierin kan hij zien wat de aanvraag vroeger was. Ook heeft hij nu een bevestigingsmail ontvangen.

2.3 Feature list

2.3.1 Aangevraagd door Axians

Tabel 1: Features Axians

Als …	wil ik …	zodat …	Prioriteit
Planner/ medewerker	Voor een periode van twee maanden alle afwezigheden in het team in één overzicht gevisualiseerd hebben	Ik op een gefundeerde manier aangevraagde afwezigheden kan goedkeuren of afkeuren of on hold zetten	Vereist
Planner/ medewerker	Bij het beoordelen van afwezigheden en in de planning zien wanneer schoolvakanties plaats vinden	ik op een gefundeerde manier aangevraagde afwezigheden kan goedkeuren of afkeuren of on hold zetten	Optioneel
Planner/ medewerker	Bij het beoordelen van afwezigheden en in de planning zien wanneer feestdagen vallen	ik geen overbodig verlof aanvraag of goedkeur	Vereist
Medewerker	Een verlofaanvraag indienen	de planner de verlofaanvraag kan behandelen: bekijken, goedkeuren, afkeuren	Vereist
Medewerker	De status van mijn verlofaanvraag opvolgen	ik weet of mijn verlofaanvraag gezien/in overweging, goedgekeurd of afgewezen is	Vereist
Planner	Zien op welk tijdstip een verlofaanvraag binnengekomen is	ik het principe 'first come, first served' kan toepassen	Vereist
Planner/ medewerker	Voor een periode van een jaar alle afwezigheden en status voor het team in één overzicht gevisualiseerd hebben	ik een duidelijk overzicht heb en kan inschatten of iets al dan niet goedgekeurd zal worden	Vereist
Planner	Een verlofaanvraag goedkeuren, afkeuren of in behandeling zetten	de medewerker feedback krijgt over de status van zijn verlof en de agenda visueel aangepast wordt	Vereist
Medewerker	Mijn eigen verlofteller opvragen	ik weet hoeveel verlof ik nog kan aanvragen	Vereist
Planner	Verloftellers aanpassen (bv. Parttimers of in functie van overzetten van saldo's naar het volgend jaar)	planners en medewerkers weten hoeveel verlof nog opstaat	Optioneel
Planner	De verloftellers van medewerkers opvragen	ik kan opvolgen hoeveel verlof de medewerkers nog zullen aanvragen	Vereist

Planner	Een gefilterd overzicht van alle medewerkers en hun kalender	ik een gesorteerd overzicht kan krijgen van de verlofdagen onder bepaalde voorwaarden	Optioneel
Planner	De lopende verlofaanvragen bekijken	ik een beslissing kan nemen voor de vakantieregeling van een zorgkundige	Vereist
Planner	Manueel verlof inplannen voor een bepaalde zorgkundige	ik zelf verlof kan inplannen zonder dat de arts hier onnodig aanvraag voor moet doen	Vereist

2.4 Geselecteerde features per sprint

2.4.1 Sprint 1

Tabel 2: Features sprint 1

To Do	Complexiteit
Basisflow vastleggen voor het aanvragen van verlof • Dokter kiest zijn verlofdata • Planner bekijkt de aanvraag • Planner neemt een beslissing	Middelmatig
Basis GUI van de kalender ter ondersteuning • Dokter ziet zijn vrije verlofdagen • Planner ziet welke dagen al ingenomen zijn • Vakanties en feestdagen worden weergegeven	Middelmatig
Filteren en sorteren van werknemers op skills, locatie, competenties...	Middelmatig

2.4.2 Sprint 2

Tabel 3: Features sprint 2

To Do	Complexiteit
Onderscheid maken tussen voormiddag en namiddag	Middelmatig
Filters verder uitwerken • Clear knop • Aanvragen ook sorteren en filteren	Middelmatig
Dokter kan planning van zijn eigen team bekijken	Middelmatig
Visuele voorstelling aanpassen met symbolen	Eenvoudig
Per kalenderdag een cijfer • Hoeveel zijn er in verlof • Hoeveel dokters zijn er nodig	Moeilijk

Historiek van aanvragen	Middelmatig
Mogelijkheid om lopende of goedgekeurde aanvragen te wijzigen	Middelmatig
Pop-ups opschonen	Eenvoudig
Planner moet voor iemand anders verlof kunnen aanvragen	Middelmatig - Moeilijk
Aantal vakantiedagen kunnen worden geraadpleegd	Eenvoudig - Middelmatig
Kleuren bij aanvragen duiden aan in hoeverre het mogelijk is om het goed te keuren	Zeer moeilijk
Gridtable: alternatieve weergave van de afwezigheden	Moeilijk

2.4.3 Sprint 3

Tabel 4: Features sprint 3

To Do	Complexiteit
Vertaling voorzien voor Nederlands en Engels. Dit eveneens open zetten voor uitbreiding	Middelmatig
Symbolen toevoegen naast de kleuren (voor kleurenblindheid)	Makkelijk
Alternatieve view in newsfeed perfectioneren	Moeilijk
Filters en zoekbalk in newsfeed verder uitwerken	Middelmatig
Optimalisaties bij historiek i.v.m. laden	Makkelijk
Bij aanvraag tonen hoe vaak die persoon al geweigerd is geweest	Middelmatig
CSS op punt zetten (bv. bij navigatiebalk)	Meer problemen dan verwacht
Mails versturen bij elke wijziging	Makkelijk

3 SYSTEEMARCHITECTUUR

In dit project werden meerdere technologieën samengebracht tot een harmonisch geheel. Zo werd gebruik gemaakt van Angular[1], Java Play[2], Ebean[3] en Docker[4].

Angular is een technologie die wordt gebruikt aan de front end. Met Angular is het mogelijk om een website op te splitsen in meerdere kleine componenten. Zo'n component bevat dan een html-, een CSS- en een ts-bestand. Het html-bestand legt de structuur van een component vast, het CSS-bestand legt de stijl (kleuren, lettertypes, …) vast en het ts-bestand voegt functionaliteit onder de vorm van code toe. TS staat hierin voor typescript. Dit is een taal die eigenlijk een superset vormt op javascript, met meer restricties. Daarnaast voegt Angular ook performantieverbeteringen toe. Wanneer twee verschillende pagina's dezelfde componenten gebruiken, worden die hergebruikt. Zo moeten ze niet opnieuw worden gedownload. Dit is een verbetering op vlak van bandbreedte en snelheid.

Java Play wordt dan weer gebruikt aan de back end. Een back end zorgt ervoor dat de front end wordt voorzien van de correcte data. Zo wordt eerst de connectie gemaakt met een databank. Wanneer de front end om data vraagt worden dan de juiste gegevens uit de databank gehaald. Vervolgens wordt die data geformatteerd op een manier waarop de front end er mee kan omgaan. Tot slot wordt de data verzonden.
Het Play-framework maakt normaliter gebruik van het Model-View-Controller (MVC) principe. Hierin wordt het model vorm gegeven door de standaardklassen die verbinding maken met de data, die data omzetten en vervolgens aanbieden aan de controller. De view is dan het deel dat je effectief kan zien. Normaal gezien biedt Java Play hiervoor zelf methodes. Omdat er een eerder ingewikkelde structuur aan de front end is, werd dit deel echter vervangen door Angular in dit project. De controller zorgt er dan voor dat via een url de juiste data wordt teruggegeven.
Omdat er niet echt gebruik wordt gemaakt van de views in het Play-framework, spreken we eerder over een REST principe dan een MVC principe. Dit omdat er enkel JSON wordt teruggegeven bij het aanvragen van een URL en dus geen klassieke website.

Ebean zorgt ervoor dat het object relationeel mappen correct gebeurt. In een databank wordt informatie opgeslagen in tabelvorm. Hierin stelt elke kolom een parameter van een object voor. Voor elk object wordt er dan een nieuwe rij aangemaakt waarbij elke kolom netjes wordt ingevuld. Een simpel voorbeeld is bijvoorbeeld een klasse Persoon. Deze bevat een naam en een leeftijd. De tabel van dit object heeft dan twee kolommen: "Naam" en "Leeftijd". Voor elke persoon wordt er dan een nieuwe rij aangemaakt bv. "Bart", "35".
Nu worden objecten op een andere manier opgeslagen in een programmeertaal. Zo worden er verwijzingen gelegd tussen objecten terwijl dit niet mogelijk is in een tabel. In code kan men makkelijk een attribuut "Kinderen" bijmaken, en kan dit een lijstje bevatten met attributen naar andere personen. In tegenstelling tot de databank waarin tabellen moeten worden gemaakt om deze linken correct op te slaan. Ebean zorgt ervoor dat dit correct gebeurt doormiddel van extra hulptabellen.

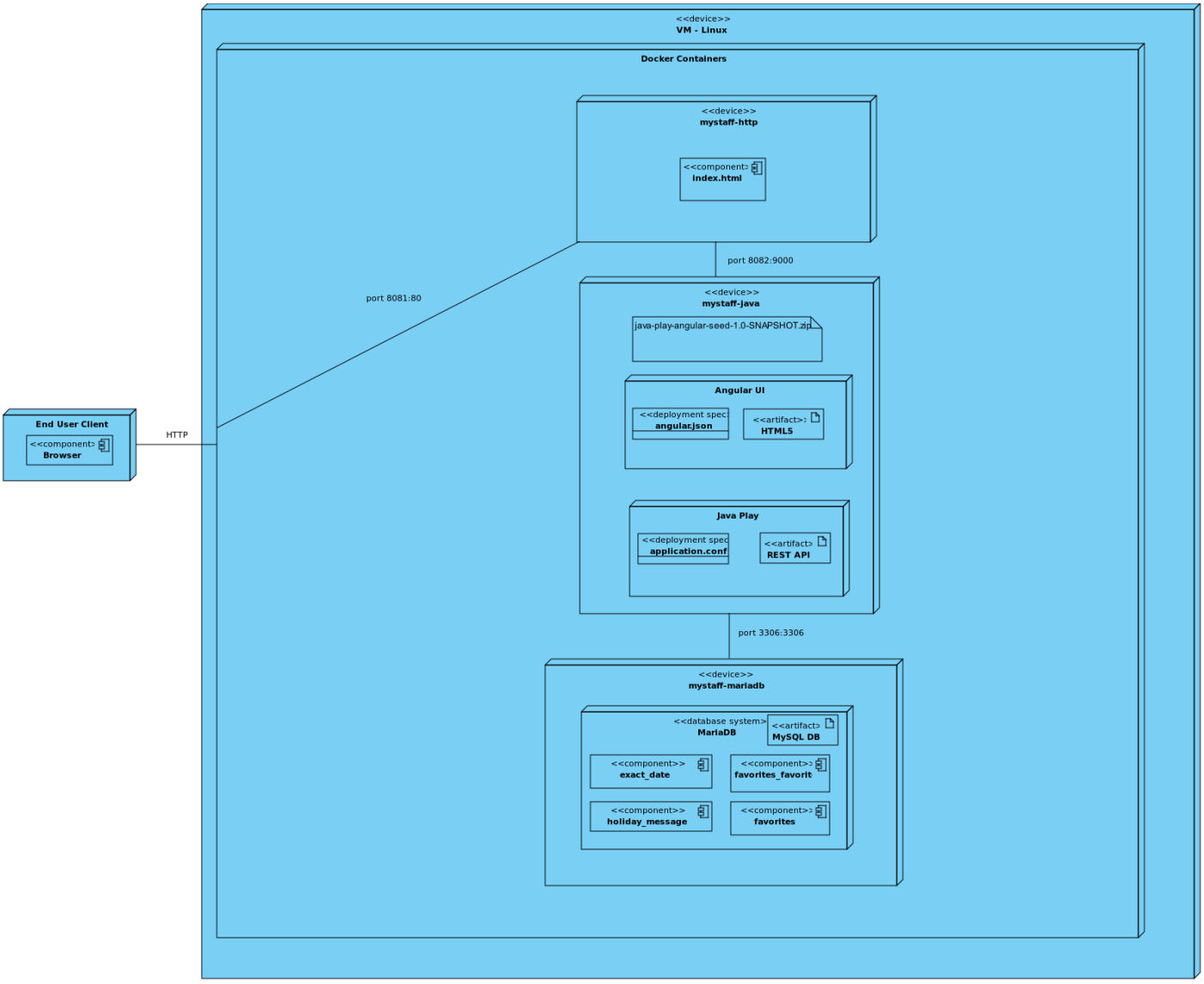
Docker is een soort virtuele testomgeving waarin de front end, de back end en de databank samen kunnen lopen. De databank waarvoor werd gekozen is MariaDB. Dit is een uitbreiding van een MySQL databank. De Docker waarin het project loopt, kan worden bereikt via http://bpvop5.ugent.be.

[1] https://angular.io/
[2] https://www.playframework.com/
[3] https://www.playframework.com/documentation/2.6.x/JavaEbean
[4] https://www.docker.com/

De front-end loopt op poort 8081 (via http://bpvop5.ugent.be:8081) . De back-end kan bereikt worden via poort 8082. Hiervoor dient wel verbinding te worden gemaakt met de Ugent VPN.

3.1 Deployment diagram

In het deployment diagram is te zien hoe alle componenten samenwerken. De user wordt via het HTTP protocol doorverbonden naar de HTTP-module van de virtuele machine. De website wordt gerenderd door het Angular framework aan de frontend en Java Play aan de backend. Beide onderdelen hebben hun eigen configuratiebestanden, waarmee eenvoudig de instellingen aan te passen zijn zoals het versienummer en de nodige dependencies. Onze website moet regelmatig data ophalen via de database. Via de interne poort kan de Java-module de MariaDB-module bereiken. Deze bevat enkele tabellen, waarvan de belangrijkste in het diagram worden weergegeven. De tabellen worden aangemaakt via een script en vervolgens worden de Java klassen gemapt met behulp van het Ebean framework.



Figuur 5: Deployment diagram

3.2 Klassendiagram

3.2.1 Zelfgemaakte klassen in de back end

In het klassengram zijn verschillende deelgroepen te vinden.

Ongeveer centraal staat de HolidayMessage klasse. Deze bevat alle informatie die nodig is om verlof aan te vragen. Meer hierover in 3.2.2.

Een uiterst belangrijke rol wordt gespeeld door de klassen wiens naam eindigt op “Controller”. Zij bevatten de functies die ervoor zorgen dat de juiste informatie wordt teruggegeven wanneer een bepaalde “GET” of “POST” aanvraag wordt gesteld. Via een speciaal routebestand worden die functies dan gelinkt aan url’s. Zo zorgt de AxiansController ervoor dat de gegevens uit de web api van Axians kunnen worden opgevraagd.
De DatabaseController zorgt er dan weer voor dat gegevens uit de eigen databank worden teruggegeven. Dit zijn dan HolidayMessages, Settings of AbsenceCounters. Het maakt dan ook gebruik van de HolidayMessage- en SettingsController.
De TranslateController wordt dan weer gebruikt om id’s van zaken zoals personen, skills, locaties en planningunits te vertalen naar hun leesbare naam.
De FormatController zorgt voor formattering van data. Vaak bevatten de objecten van Axians zeer veel informatie waar men aan de front end niet echt iets mee is. Daarom worden deze objecten gestript tot de kern die nodig is, om niet te veel netwerkverkeer te veroorzaken. Ook voor zaken uit de database zijn soms speciale formatteringen gewenst en ook daarvoor zorgt deze controller.
Niet onbelangrijk te vermelden bij deze controllers is dat bij sommigen onder hen een GET aanvraag opgesplitst is in twee functies. Eén van de twee functies zet de gevraagde data om naar de juiste klasse. Het is makkelijker om met een klasse te werken, dan met JSON die bijvoorbeeld ontvangen werd van de Axians web api. De tweede functie kapselt dan die functie in en zorgt ervoor dat het bestand als JSON wordt teruggezonden in een Result object. Een Result object zorgt ervoor dat de juiste code wordt meegezonden naar de ontvanger, zoals “200 OK”.
Door deze dualiteit kunnen ook andere klassen makkelijk zaken aan de controller vragen door enkel gebruik te maken van de binnenste functie die een klasse teruggeeft. Wanneer alles in één functie zou worden geplaatst die een Result object teruggeeft, zou dit onmogelijk zijn.

In een eerste stadium van de applicatie, hield de back end ook een groep bij van personen en hun gegevens. Ook werd de op dat moment ingelogde persoon bijgehouden op de back end. Dit zorgt natuurlijk voor zeer vreemde situaties. Wanneer er één persoon tegelijk inlogt, is er geen probleem. Wanneer echter persoon 1 inlogt, ingelogd blijft en persoon 2 inlogt, staat de ingelogde gebruiker op persoon 2 en telt nu alles wat persoon 1 doet als persoon 2. Uiteraard is dit niet de bedoeling. Daarom ook dat het bijhouden van een groep personen uit de applicatie is gehaald. De groep personen gerelateerd aan persoon 1 hoeven niet dezelfde te zijn aan die van persoon 2, dus hoeft deze ook niet te worden bijgehouden.

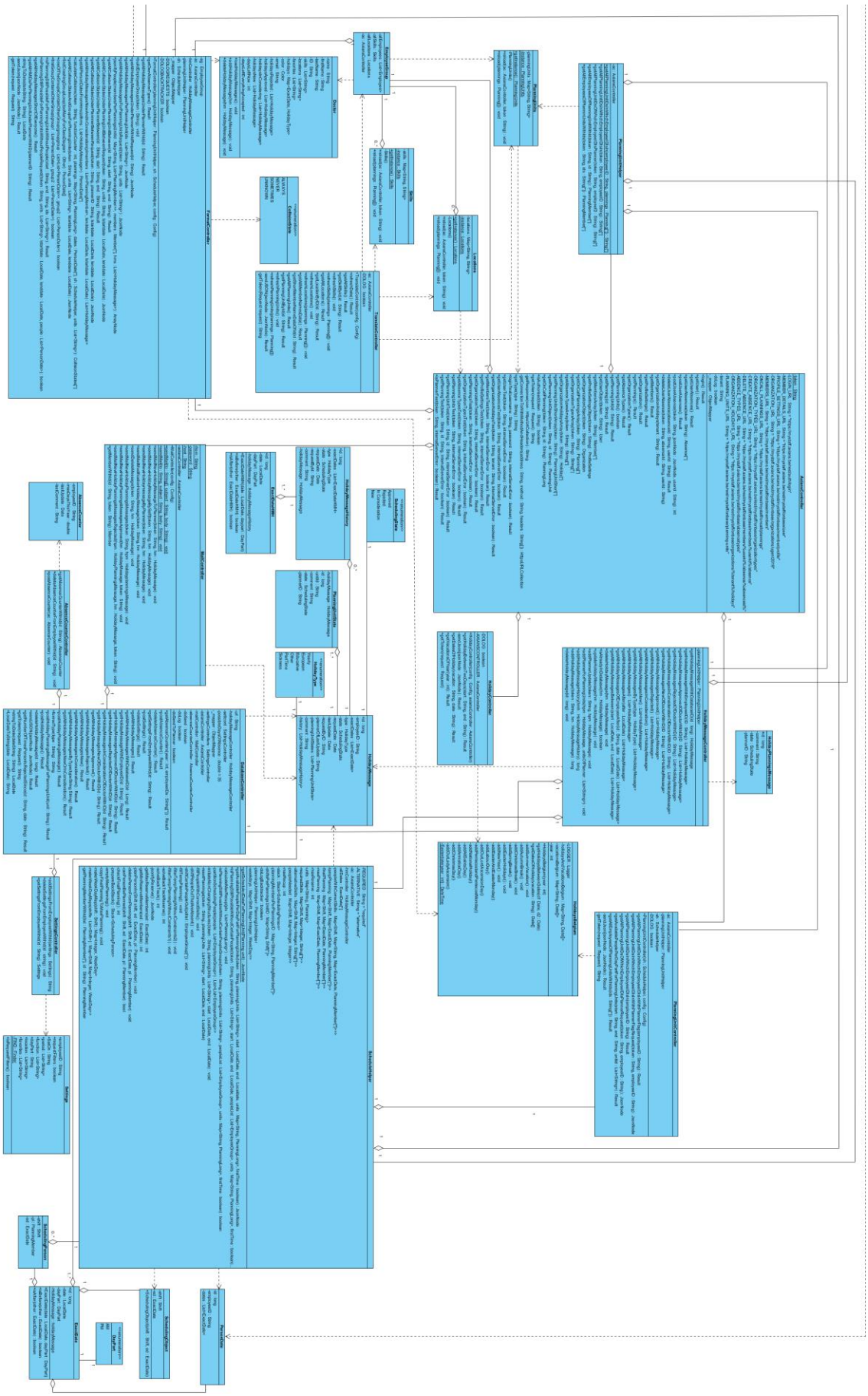
Enkele simpele klassen zijn Skills en Locations. Ze bevatten een simpele map van strings op strings. In een eerdere implementatie werden enumeraties gebruikt. Dit bleek echter snel zijn gebreken te hebben. Zo is het niet praktisch om aan een enumeratie nieuwe waarden toe te voegen. Belangrijker echter is het feit dat Axians skills en locaties adresseert met een id. Zo dus is het logischer om een implementatie te gebruiken die een id op de leesbare versie mapt.
Bij opsommingen die absoluut vast staan zoals de planningsstaat van een verlofaanvraag (goedgekeurd, afgekeurd, …) of een dagdeel (AM, PM), werd dan logischerwijs wel gebruik gemaakt van enumeraties.

Een andere belangrijke klasse is MailController. MailController zal ervoor zorgen dat mails worden verstuurd bij elke belangrijke gebeurtenis dat zich voordoet. Het al dan niet versturen van mails is configureerbaar. Dit omdat wanneer men zou zien bij het in gebruik nemen van het programma, dat er een waterval aan mails wordt gegenereerd, dat men dit toch nog kan uitschakelen.
Op zich is MailController niet afgeleid van Controller. Toch werd de naamgeving gekozen omdat de klasse via mails toch contact heeft met gebruikers.

HolidaysBelgium heeft één voornaam doel: alle Belgische vakantiedagen weergeven in een array. Voor bv. het berekenen van de paasvakantie heeft deze vrij ingewikkelde algoritmen nodig.

ScheduleHelper zorgt voor de functionaliteit die de planner zal helpen met het beslissen of een verlof al dan niet mag worden goedgekeurd. De basis van veel van deze functies is een backtrackingsalgoritme dat een mogelijke planning opstelt.
Zo wordt berekend welke verlofaanvragen voor problemen zorgen door mogelijke planningen op te stellen. Zo kan men via het beschreven algoritme in 2.1.2 verlofaanvragen een bepaalde kleurcode toekennen om aan te geven of deze al dan niet zonder problemen kunnen worden ingepland.
Ook zorgt deze klasse bv. voor het berekenen hoeveel mensen minstens er maximaal op verlof kunnen worden gestuurd op een bepaalde dag. Het is nodig om twee getallen i.p.v. één getal te geven omdat bv. door bepaalde constraints een bepaald persoon niet mag op verlof gaan, omdat hij onmisbaar is, en andere personen wel. Om het minimum te berekenen worden personen uit een geslaagde planning gehaald op een manier dat de planning zo snel mogelijk kapot gaat. Om het maximum te vinden, worden personen uit een geslaagde planning gehaald op een manier die zo weinig mogelijk invloed heeft op andere dagen en personen met de gegeven constraints. Wanneer er in totaal 7 personen aanwezig zijn en 1 persoon nodig, kan het voorvallen dat men 0 als minimum en 6 als maximum vindt. Slechts één persoon heeft de nodige skills voor die dag en is dus onmisbaar.
Deze klasse maakt gebruik van SchedulingPerson- en SchedulingObject-objecten.

SchedulingPerson bevat een shift, een dagdeel en een werknemer. Een lijst van deze objecten kan dus een planning voorstellen. Wanneer men in het proces van het opstellen van de planning zit en zaken bekijkt voor één enkele persoon, hoeft die persoon niet mee opgeslagen te worden en is dus het SchedulingObject dat enkel een shift en dagdeel bevat, voldoende.



Figuur 6: Klassendiagram back end

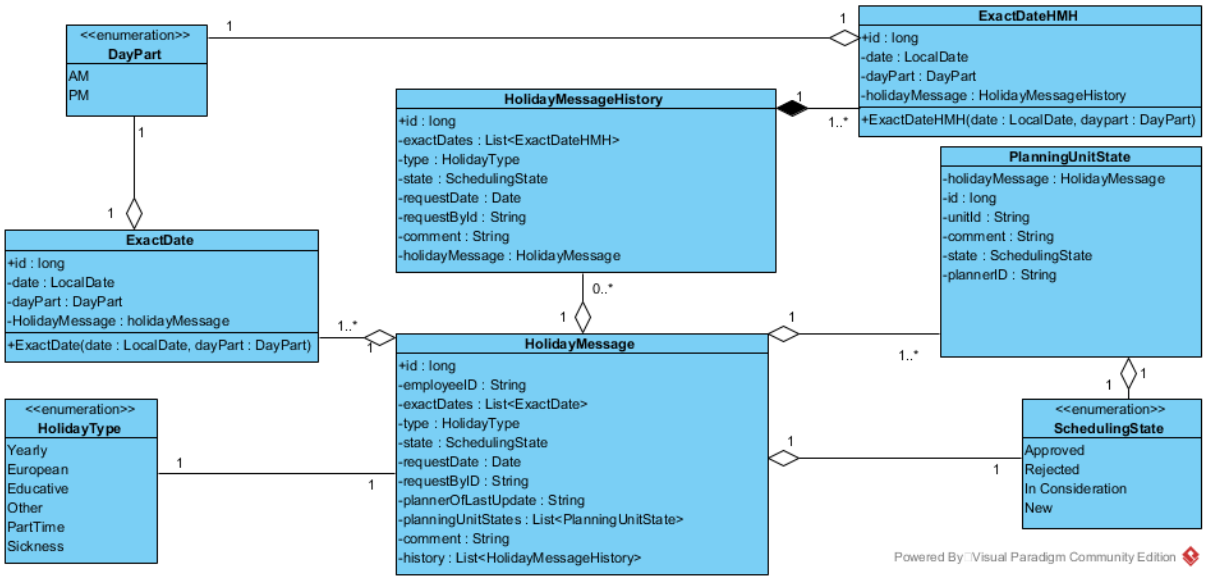
3.2.2 Detailweergave: HolidayMessage

De HolidayMessage klasse zorgt ervoor dat front en back end op een eenduidige manier met elkaar kunnen communiceren over hoe verlof wordt aangevraagd. Het is een object waarin opgeslagen staat wie verlof neemt, welk soort verlof hij neemt, of het al goedgekeurd is door de planner, wanneer het werd aangevraagd, wanneer de planner de laatste update deed, eventueel commentaar die de persoon zelf gaf bij het indienen en commentaar die de planner eventueel gaf bij het al dan niet goedkeuren van de verlofaanvraag.

Ook bevat zo'n object natuurlijk de dagen waarover het gaat. Omdat een werknemer enkel voormiddagen, enkel namiddagen of mixen ervan kan aanvragen, kon niet gewoon gewerkt worden met een begin en einddatum. Daarom bevat elke HolidayMessage een lijst van halve dagen die zo samen het totaalbeeld vormen. Er werd gekozen voor LocalDate objecten in plaats van gewone Date objecten, omdat op deze manier enkel de dag wordt bijgehouden, en niet het uur en de minuten, die hier niet belangrijk zijn.

Verder bevat een HolidayMessage mogelijk een lijst van HolidayMessageHistory (HMH) objecten. Het is mogelijk dat een dokter een verlofaanvraag toch nog verandert nadat hij ze indiende, omdat hij bijvoorbeeld toch een dag extra wilt, of hij per ongeluk enkel voormiddagen had aangeklikt. Deze HMH objecten bevatten net deze informatie van vorige versies van de HolidayMessage. Opmerkelijk misschien is de klasse ExactDateHMH. Deze is nagenoeg hetzelfde als de gewone ExactDate, enkel verwijst ze naar HMH objecten. Deze manier werd gekozen omdat de ORM-software (die ervoor zorgt dat objecten naar de database worden geschreven) niet goed om kan met interfaces of abstracte klassen. Er werd bijvoorbeeld geprobeerd om zowel HM als HMH objecten onder een zelfde interface "ExactDateHolders" te steken, maar dit gaf niet het gewenste resultaat.

Als laatste bevat elke HolidayMessage ook steeds een lijst van PlanningUnitStates. Een verlofaanvraag moet steeds goedgekeurd worden door minstens één planner van elke planningseenheid waar de persoon lid van is. Daarom moet een HolidayMessage dus bijhouden welke planners van welke planningseenheden een bepaalde staat gaven aan zijn bericht, en in welke planningseenheden hij nog steeds de staat "nieuw" draagt. Hiervoor zorgt de PlanningUnitState.



Figuur 7: Klassendiagram HolidayMessage

3.2.3 Web API Axians

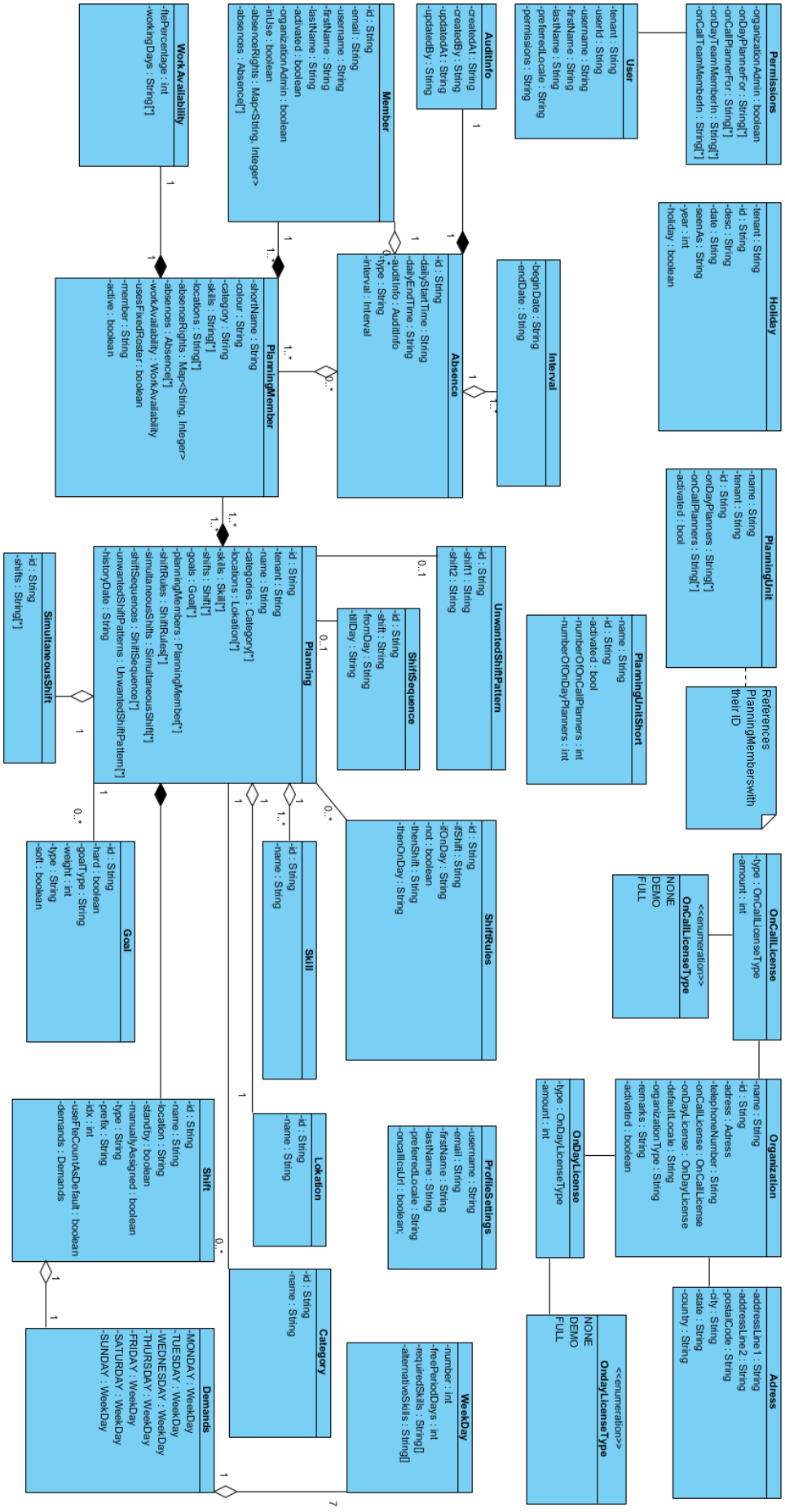
Om een overzicht te krijgen van de data die verkregen werd van Axians, werd ook een klassendiagram van die data opgesteld. De manier waarop is vrij eenvoudig.
Eerst werden alle aanvragen naar Axians toe uitgevoerd, zodat JSON-strings verkregen werden. Vervolgens werden deze JSON-strings in een tool geplakt die JSON-bestanden omzet naar Java klassen[1].

De belangrijkste klasse die nu in het oog springt is Planning. Zij bevat alle informatie over de shiften die mogelijk zijn in een bepaalde planningseenheid. Ze bevat alle regels waaraan shiften moeten voldoen. Zo kan men bijvoorbeeld instellen dat als een persoon een shift X doet, dat hij de dag erna shift Y niet mag doen, of juist moet doen.
Merk op dat er dubbels lijken te zijn van sommige klassen.
Zo zijn er bv. Member en PlanningsMember. Bij beide worden ze member genoemd door Axians, maar ze bevatten niet dezelfde attributen. Een member bevat alle gegevens die echt bij de persoon horen: voornaam, achternaam, email, … Een planningsmember daarentegen stelt meer de gegevens voor die echt bij een planning horen. Zo zijn er voor dezelfde persoon meerdere planningsmembers mogelijk. Eentje voor elke planningseenheid. Deze geeft dan aan welke skills deze persoon heeft en op welke locaties deze persoon werkt in die planningseenheid. Een persoon kan bv. cardioloog zijn in de ene planningseenheid en enkel anesthesist in de andere.
Ook zijn er klassen die eindigen op Short zoals PlanningUnit een variant PlanningUnitShort heeft. Het verschil is subtieler. Wanneer een lijst van alle PlanningUnits wordt opgevraagd, wordt de verkorte vorm doorgestuurd. Wanneer de details van één enkele PlanningUnit wordt opgevraagd via haar id, dan worden er meer details vrijgegeven.

Dat de klassen gegenereerd werden, is te zien aan de klassen die de weekdagen voorstellen. In het bericht van Axians werd per shift een overzicht teruggestuurd met er in : MONDAY { number: x, shifts:[…,…]}. Zo ook voor de andere weekdagen. De klassengenerator nam dus aan dat elke weekdag een eigen klasse moest zijn, terwijl ze eigenlijk dezelfde soort data bevatten. Omdat echter de JSON-strings in de back end ook worden vertaald naar klassen door een automatische mapper, is er gekozen om de weekdagen zo te houden. Om echter plaats te besparen in het schema, werd er gekozen om deze met een enkele klasse WeekDay voor te stellen.

Een voorbeeld van het gebruik van deze klassen is het opstellen van een voorbeeldplanning die volledig moet zijn ingevuld. Dit gebeurt meerdere keren in de applicatie, om bv. te berekenen hoeveel personen op een bepaalde dag op verlof kunnen gaan. Om dit te realiseren wordt een lijst van alle Plannings opgevraagd, wordt vervolgens geïtereerd over alle planningsmembers daarin, om daarna bij te houden welke skills en locaties deze persoon bevat. Hierna wordt gekeken naar al de shifts van die plannings, om erna te kijken naar de demands en om vervolgens te kijken naar al de weekdagen om te ontdekken welke skills er op die dag nodig zijn en hoeveel mensen deze moeten invullen.
Opmerking: Een weekdag bevat zowel required skills als alternative skills. Enkel personen die de volledige set van één van deze twee soorten skills bevatten, mogen worden ingepland op deze dagen.

[1] http://www.jsonschema2pojo.org



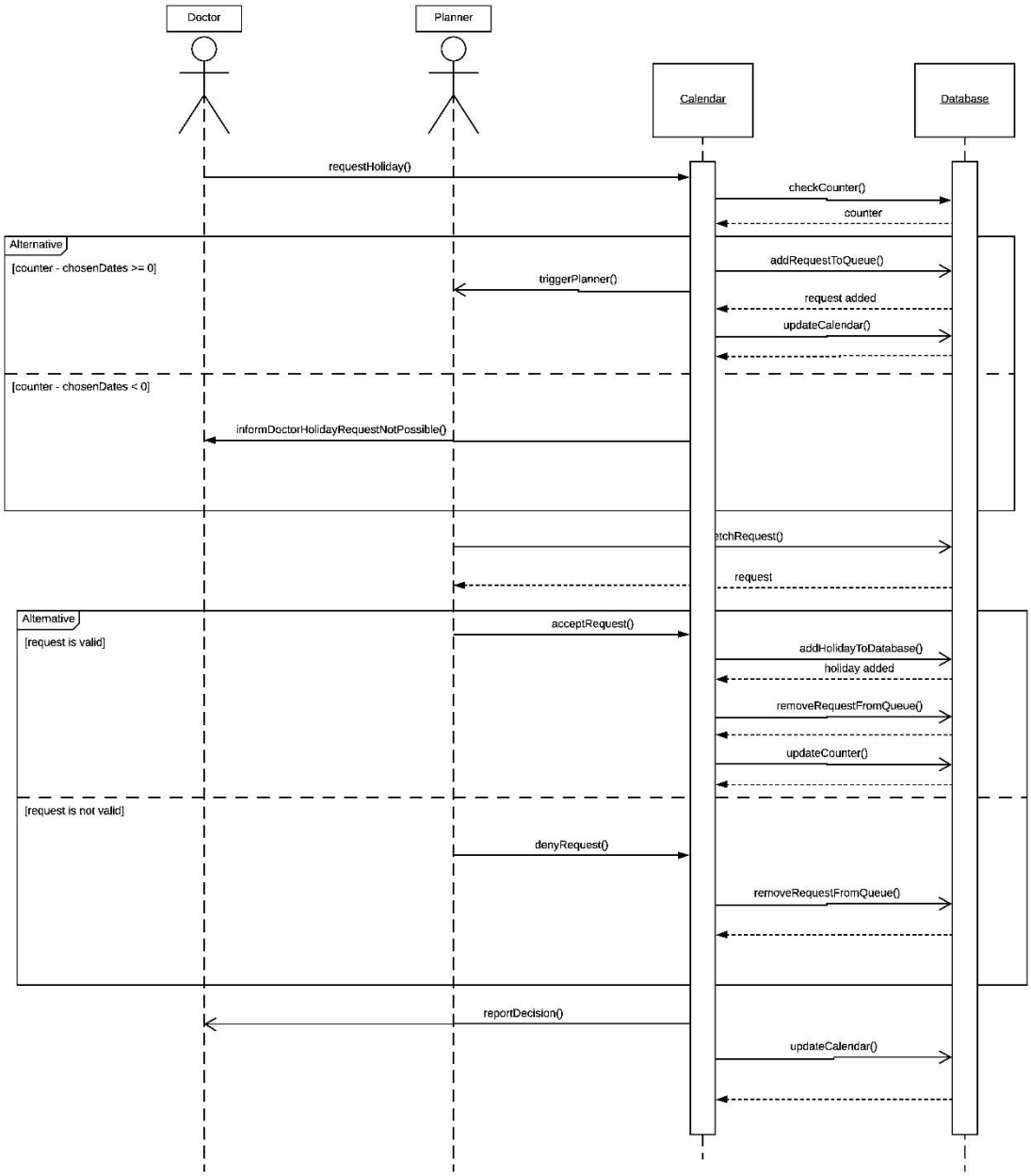
Figuur 8: Klassendiagram web API

3.3 Sequentiediagram

Om het verloop van een verlofaanvraag te visualiseren, werd er gebruikt gemaakt van een sequentiediagram. In het diagram zijn twee actoren te zien: een dokter en een planner. Deze interageren met twee componenten: de kalender en de database. De kalender is ook nog onderverdeeld in verschillende componenten, maar voor de eenvoud stellen we dit als één geheel voor.

Nadat de dokter zijn verlofdagen gekozen heeft in de kalender, kan hij/zij op de submit knop drukken. Vervolgens wordt de methode requestHoliday() aangeroepen op de kalender component. Deze zal eerst checken in de database of de dokter nog verlof kan opnemen. Indien dit niet het geval is, zal de dokter een foutmelding te zien krijgen. Als er wel nog genoeg verlofdagen over zijn, zal de aanvraag in de wachtrij geplaatst worden. De planner zal dan worden getriggerd d.m.v. een e-mail. Ook de dokter krijgt nu een bevestigingsmail. De lopende verlofaanvraag zal hierna getoond worden in de kalender van de dokter.

De volgende stap in de levenscyclus van een aanvraag, is de evaluatie door de planner die verantwoordelijk is voor deze dokter. De planner kan gebruik maken van enkele hulpmiddelen om een aanvraag grondig te evalueren, zoals een kleurtje dat aangeeft of de aanvraag mogelijk is, meerdere kalender lay-outs, een overzicht van de verlofdagen per team, filteren op naam, skills, locatie... Op deze manier kunnen zo veel mogelijk verlofaanvragen optimaal worden toegewezen. De planner heeft een overzicht van alle lopende aanvragen en kan een aanvraag goedkeuren, afwijzen of in beraad houden. Indien de aanvraag in beraad wordt gehouden, wordt er geen actie uitgevoerd en blijft de aanvraag bestaan. Als de aanvraag wordt goedgekeurd, zal deze worden toegevoegd aan de database en wordt het uit de wachtrij verwijderd. Het aantal opgenomen verlofdagen van de dokter wordt hierna ook aangepast. Bij een afgekeurde aanvraag, wordt deze gewoon verwijderd uit de wachtrij in de database. Na evaluatie van elke aanvraag zal er een gepaste mededeling aan de dokter worden getoond waarom de planner de aanvraag heeft goedgekeurd/afgekeurd en zal hen een mailtje worden gestuurd. Als laatste stap zal de kalender zichzelf updaten om de meest recente beslissingen weer te geven.



Figuur 9: Sequentiediagram

3.4 Database diagram

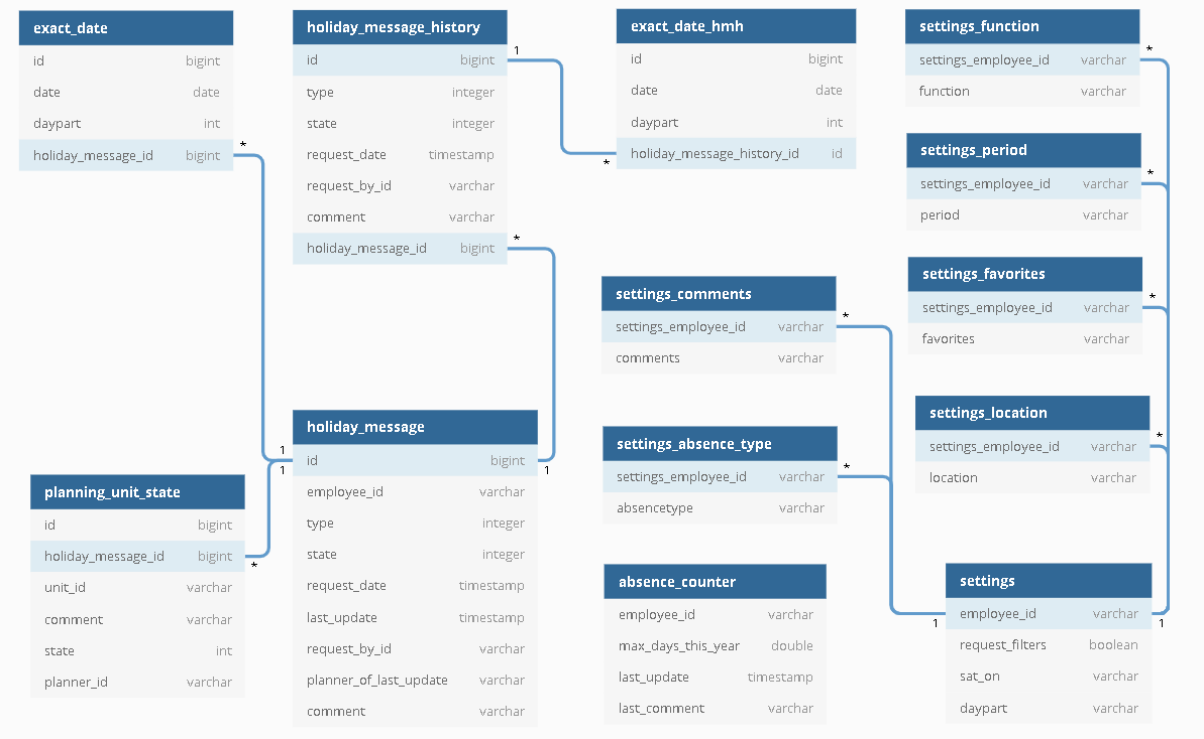
De database bewaart twee belangrijke zaken. HolidayMessages en Settings.

Voor meer info over het nut van HolidayMessages, zie sectie 3.2.2. Zoals daar beschreven staat, bevat een HolidayMessage meerdere ExactDates. Deze één op veel relatie wordt verwezenlijkt door het ID van de HolidayMessage op te slaan bij elke ExactDate. Op die manier weet elke ExactDate bij welke HolidayMessage ze hoort. Zo ontstaat een eenduidige waarheid. Hetzelfde gebeurt voor HolidayMessageHistory- en PlanningUnitStateobjecten.

Merk op dat hoewel een HolidayMessage een referentie heeft naar het ID van een werknemer, werknemers niet in deze databank worden opgeslagen. Werknemers worden immers opgehaald uit de databank van Axians, via de web api.

Settingsobjecten zorgen ervoor dat de staat van gekozen instellingen van een persoon wordt opgeslagen. Zo hoeft men niet steeds dezelfde instellingen opnieuw aan te vinken. Zo bevat een Settingsobject bv. favorieten. Die worden gebruikt aan de front end bij het overzicht van verlofdagen van collega's. Het is dan mogelijk om enkel deze favorieten te tonen of ze alleszins toch bovenaan te laten staan.
Ook dit is een één op veel relatie. De tabel Settings bevat de werknemers die favorieten hebben. De tabel SettingsFavorites bevat dan als favorites het ID van een werknemer die als favoriet bevonden wordt, en linkt deze dan aan via SettingsEmployeeID aan het ID van de werknemer van wie hij als favoriet wordt bevonden. De andere instellingen werken op een analoge manier.

AbsenceCounterobjecten zorgen ervoor dat de verloftellers van een bepaalde persoon worden opgeslagen. Ook de laatste update van het object en eventuele commentaar over de laatste wijziging zijn erbij opgeslagen.



Figuur 10: Database diagram

4 TESTPLAN

4.1 Communicatie met MyStaff API

Om na te gaan of de applicatie correct gebruik maakt van de MyStaff API werden er een aantal unit tests voorzien. Deze zullen alle methodes van de AxiansController, die requests sturen naar de MyStaff API, testen. De methodes worden één voor één getest op verschillende manieren, nl. door gebruik te maken van correcte login gegevens, tenant (groep) en IDs van planningunits. Vervolgens worden er correcte data gemixt met valse data, zo wordt er nagegaan of de AxiansController correct kan omgaan met zowel correcte als valse gegevens. Dit werd nagegaan door Robin De Zwaef

4.2 Communicatie met MariaDB

Naast de unit tests voor de AxiansController zijn er ook unit tests die de communicatie met de gebruikte MariaDB zullen testen. Vooraleer er een test uitgevoerd wordt, zullen de tabellen die gebruikt worden door de unit tests leeggemaakt worden, zodat er geen conflicten zouden ontstaan door eventuele reeds bestaande gegevens. In de unit tests die aanwezig zijn, is er vooral een focus op de HolidayMessages en ExactDates die gelinkt zijn aan deze HolidayMessages. Zo zijn er tests voorzien op het gewoon toevoegen en opnieuw opvragen van één of meerdere HolidayMessages, toevoegen van een HolidayMessage en enkele ExactDates en vervolgens nagaan of deze een correcte relatie hebben in de database. Ook wordt er getest of HolidayMessages kunnen worden opgevraagd aan de hand van hun type en status. Als laatste test voor de HolidayMessages wordt er nagegaan of het updaten van bepaalde kolommen van een HolidayMessage, en vervolgens deze geüpdatete kolommen opnieuw in de database plaatsen, de gewenste effecten heeft. Dit werd eveneens getest door Robin De Zwaef.

4.3 Opvragen van vakanties

Aangezien de meeste vakanties in België berekend worden a.d.h.v. feestdagen, werd ook een klasse geschreven om de vakanties in een bepaald jaar terug te geven. Ook hier werden unit tests gebruikt om de vele randgevallen te testen en na te gaan of begin -en einddatum overeen komen met de officiële vakantiedata opgelegd door de overheid. Het bepalen van meerdere vakanties en feestdagen steunt regelmatig op het berekenen van de dag waarop Pasen valt. Deze berekening is redelijk ingewikkeld en werd dus ook grondig getest. Dit testen zijn uitgevoerd door Matisse Ghesquiere.

4.4 CollisionState algoritme (kleurtjesalgoritme)

Het kleurtjes algoritme geeft bepaalde aanvragen bepaalde kleuren om te tonen of ze al dan niet zorgen voor fouten indien ze worden ingepland (zie 2.1.2).
Om te kijken of het kleurtjesalgoritme correcte resultaten levert, volstaan simpele testen niet. Dit mede doordat we niet beschikken over de database van Axians waarin shiften en dokters worden opgeslagen.
Daarom moeten handmatig via de website van Axians[1] nieuwe planningseenheden met nieuwe shifts worden aangemaakt met bepaalde eisen. Ook moeten extra dummy-accounts worden bijgemaakt.

1 https://mystaff.axians.be/

Vervolgens moeten deze dummy-accounts worden gekoppeld aan deze shifts op manieren waarbij het voorspelbaar is welke kleuren het algoritme zal leveren.
In de volgende voorbeelden wordt gebruik gemaakt van planningseenheden Demo1, Demo2 en Demo3 en accounts demo1, demo2 en demo3.
Voorbeelden:

- In Demo1 zijn twee shiften simultaan ze hebben elk één persoon nodig, demo1 en demo2 vragen verlof aan, demo3 werkt gewoon. Ze hebben allen de nodige skills. Accounts demo1 en demo2 vormen nu een conflict en staan in het oranje.
- In Demo2 is er een shift met één persoon nodig. Account demo2 is de enige persoon die deze shift kan invullen. Wanneer hij op een dag van die shift verlof aanvraagt, verschijnt deze aanvraag in het rood.
- Om de werking van de recup te testen wordt in Demo2 vrijdag een shift toegevoegd die twee dagen recup heeft en zondag een shift wordt toegevoegd die ook één persoon vergt. We zorgen dat slechts 2 personen de nodige skills hebben voor deze twee shifts. Wanneer één van de twee personen op één van deze twee dagen verlof neemt, zal deze rood komen te staan.

Aangezien een groene kleur de standaardsituatie is, wordt deze kleur niet uitgebreid getest. In principe zal wanneer rood en oranje correct werken, ook groen naar behoren functioneren. Deze testen werden geschreven door Jesper Van Caeter.

5 EVALUATIE EN DISCUSSIES

5.1 Performantie

5.1.1 Databankoperaties

In de back end wordt Java Play gebruikt, deze technologie gebruikt standaard de ORM software Ebean. Ebean blijkt nu niet de beste ORM software beschikbaar te zijn. Dit werd duidelijk wanneer bleek dat Ebean niet toelaat om embeddable objecten op te slaan. Dit zijn objecten die eigenlijk volledig deel zijn van een ander object. Ook met een object dat lijsten van andere objecten bevat, kan het niet goed overweg.

Daarom lukte het in eerste instantie niet om een HolidayMessage (een verlofaanvraag) op te slaan, omdat die lijsten bevat van ExactDates (de data van die aanvraag) en PlanningUnitStates (bevat data i.v.m. de beslissing van de planner over de aanvraag). Het probleem had te maken met het feit dat ID's worden toegekend aan een object als het wordt opgeslaan in de databank, maar niet aan het object zelf. Daarom worden nu eerst de data aan de HM gelinkt, wordt die HM dan opgeslagen in de databank, wordt hij er dan weer uitgehaald, wordt er nog eens alles aan gelinkt, en dan nog eens opgeslagen.

Het spreekt voor zich dat dat niet zeer performant is, maar het lijkt de enige manier te zijn die werkt in Ebean.

5.1.2 Zoekfuncties

In de newsfeed-component zijn er meerdere zoekfuncties aanwezig. In de balk bovenaan kan je zoeken naar personen, skills, locaties, … Links kan je als planner een andere persoon inplannen en ook daar kan je zoeken naar een persoon.
Nu zijn er twee mogelijkheden om alle data aan die zoekfuncties te bieden. Ofwel wordt alle data op voorhand gestuurd naar de front end. Dit zorgt ervoor dat de zoekfunctie uiterst snel werkt, maar dat er mogelijks data wordt opgestuurd die nooit wordt opgevraagd. Ofwel wordt steeds wanneer iets gezocht wordt, de data voor één zoekmethode opgestuurd. Dit zorgt dan dat er mogelijks minder data wordt opgestuurd. Het feit dat er dan meer calls gebeuren en dat het resultaat trager is, zorgde ervoor dat wij de eerste methode verkozen.

5.1.3 Stabiliteit

Door de vele calls die nodig zijn van front end naar back end, ontstond er veel asynchroniteit.
De meeste functies waar dit voor problemen kan zorgen, zijn correct geconfigureerd om daarmee om te gaan. Enkel wanneer de gebruiker als planner naar het overzicht gaat voor de eerste keer in zijn sessie, kan het gebeuren dat zijn kleurtjes niet laden. Een oplossing hiervoor is nog niet gevonden. De gebruiker kan dan refreshen of beter, enkel de kleurtjes herladen door op het refresh-icoontje rechts van aanvragen te klikken.

5.1.4 Rode aanvragen

Om de planner kleurtjes aan te bieden i.v.m. aanvragen en om hem te zeggen hoeveel personen elke dag op verlof mogen worden gestuurd, steunen de algoritmes vaak op het maken van een geldige planning. Wanneer er geen planning kan gemaakt worden, geven deze algoritmes geen geldig

resultaat en levert dat veel fouten op aan de front end. Het is dus uiterst belangrijk dat de planner nooit een rode aanvraag goedkeurt.
Normaal gezien wordt voor een rode aanvraag de accepteer-knop uitgegrijsd en kan deze niet worden aangeklikt. Ook wanneer er een element van een oranje groep wordt goedgekeurd, wordt de staat van de andere elementen in die groep snel berekend en worden de nodige oranje aanvragen rood indien nodig. Zo is het dus in principe nooit mogelijk om een onmogelijke aanvraag goed te keuren.
Wanneer de planner te snel is voor de controle, of een hacker slaagt er toch in om een rode aanvraag goed te keuren, dan is het dus helaas onmogelijk om aanvragen te testen in de periode van de gefaalde aanvraag. Ook zullen verschillende andere componenten uiterst onvoorspelbare resultaten geven.
Ook belangrijk is dat het niet mogelijk is om wanneer een rode aanvraag toch op één of andere manier wordt goedgekeurd, die achteraf terug te vinden. De aanvraag is nu goedgekeurd, dus alles crasht.
Het enige wat de planner kan doen is manueel kijken naar alle aanvragen die zijn goedgekeurd in de teamplanning en te vragen aan de persoon van wie de aanvraag was, of die persoon die aanvraag wil verwijderen. Een planner mag immers nooit goedgekeurd verlof kunnen opnieuw verwijderen. Dit toestaan zou sowieso wangebruik veroorzaken.
Alhoewel het dus in principe onmogelijk is voor een planner om een rode aanvraag goed te keuren, dient hij toch altijd te dubbelchecken bij het goedkeuren van oranje aanvragen.

5.2 Beveiliging

5.2.1 Authenticatie

Axians, het bedrijf die de opdracht gaf, zorgde zelf voor een inlogsysteem. Het doorgeven van een usernaam en een wachtwoord naar hun back end, levert een soort token die hoort bij die persoon en dat elke keer dat hij inlogt, verandert. Bij het doen van elke call naar hun back end, moet dat token worden meegegeven. Indien het token verkeerd is, wordt er een fout opgeroepen. Daardoor is het systeem automatisch beveiligd voor calls die naar Axians gaan.

Dit systeem is ook voortgezet in functies die in principe niet op de back end van Axians steunen, maar zich bv. enkel op de database baseren. Dat wordt gerealiseerd door eerst een kleine call te doen naar Axians, die dus een token vergt, en te kijken of die methode al dan niet een fout oproept. Hierdoor wordt het onmogelijk om eerder welke call te doen naar de back end wanneer de gebruiker niet ingelogd is.

De usernaam en het wachtwoord in de login-call zijn helaas slechts geëncrypteerd in base64. Afluisteren van de calls kan dus snel het wachtwoord van de gebruiker opleveren.

5.3 Schaalbaarheid

5.3.1 Meerdere bedrijven

Axians heeft zelf een systeem ingebouwd waardoor met elke call enkel informatie kan verkregen worden voor één bepaald bedrijf. Dit gebeurt door een “tenant” mee te geven aan die call. Zo werd om dit project te ontwikkelen de tenant “ugent2019” toegewezen.

Die tenant is opgenomen in de configuratiebestanden van het project, namelijk in application.conf. Hierdoor kan Axians makkelijk de applicatie toewijzen aan meerdere bedrijven, zonder dat ze aan elkaars informatie kunnen.

5.3.2 Trage algoritmes

Voor dit project werden slechts zes accounts toegewezen. Enkele dubbelaccounts konden worden gemaakt, maar lang niet genoeg om te kijken hoe de applicatie in een realistische omgeving zal reageren. Een meer realistische testomgeving met meer gebruikers bleek niet mogelijk te zijn.

Het kleurtjesalgoritme dat aan de planner verteld of een aanvraag al dan niet mag worden goedgekeurd, is van veel dingen afhankelijk. Zo is ze sneller wanneer er minder mensen aanwezig zijn onder een bepaalde planner, wanneer de aanvragen ver uit elkaar liggen, wanneer er minder shifts zijn onder die planner, hoeveel groepen van aanvragen er zijn die samen voor problemen zorgen etc.
Het is dus nooit getest in een realistische omgeving en enkel de tijd zal vertellen of het performant genoeg is.

5.4 Problemen en geleerde lessen

5.4.1 Asynchroniteit

De applicatie was redelijk lang instabiel. De back end voorziet steeds personen, locaties, skills, etc. aan de back end onder de vorm van id’s. Daarom is er natuurlijk een service nodig aan de front end die de id’s kan omzetten naar hun leesbare naam. Deze service moet echter eerst ingeladen worden met alle id’s en hun passende leesbare namen. Hierdoor ontstaat asynchroniteit.

Vaak werden dus vertalingen aangevraagd terwijl de vertaling nog niet geladen was, waardoor overal fouten werden gegooid en de applicatie vreemd gedrag vertoonde.

Dit werd opgelost door een event te triggeren wanneer alles is geladen en dus alle functies te laten wachten tot het event voltooid is.

5.4.2 Planningseenheden

In de eerste sprint, was er enkel een basisflow nodig. De persoon die verlof aanvroeg moest kunnen worden weergegeven, samen met diens vaardigheden.
In sprint twee zou dan meer complexe informatie worden gegeven.

De complexere informatie bestond uit planningseenheden. Een planningseenheid is een soort van collectie aan personen, shiften, skills en locaties. Personen kunnen in meerdere planningseenheden zitten en een planner kan planner zijn van meerdere planningseenheden. Elke planningseenheid heeft dus zijn eigen shiften, skills en locaties, maar deelt personen met andere planningseenheden.

In sprint één werden skills en dergelijke gekoppeld aan een persoon.

Voor sprint twee klopte dit eigenlijk niet meer en was er een veel ingewikkeldere structuur nodig. De planner moest namelijk kunnen kijken per planningseenheid welke skills etc. bij een bepaalde persoon horen. Dit zorgde er echter voor dat veel functies aan de front end praktisch volledig moesten worden weggegooid en opnieuw geschreven.

In de toekomst is het dus misschien beter om vanaf het begin uit te zoeken hoe het eindproduct er zal moeten uitzien, en ofwel meteen de structuur correct te maken, ofwel de structuur op uitbreidingen te voorzien.

5.4.3 JSON to POJO

Om de calls van Axians goed op te vangen, was er een model nodig die die calls omzette naar klassen. In een eerste benadering werden deze klassen met de hand opgesteld door naar elke call apart te kijken. Dit was een zeer tijdrovend en foutgevoelig proces.

Snel googelen leverde echter een makkelijke tool[1] op die gegeven de JSON-string van de call een kant en klare Java code opleverde die meteen kon werden geïmplementeerd door de back-end.

5.4.4 Proxy naar Axians

Tijdens het begin van het project was er het probleem dat de applicatie geen requests kon sturen naar de myStaff API omdat deze geblokkeerd werden door de browser, omwille van het feit dat deze geblokkeerd werden door het Cross-Origin Resource Sharing (CORS) mechanisme. De requests werden niet voorzien van een Access-Control-Allow-Origin HTTP header met als gevolg dat er geen requests konden gestuurd worden naar een server dat zich niet op hetzelfde domein bevond als de applicatie. Dit probleem wordt uiteindelijk opgelost door een proxy aan te maken naar de URL van de myStaff API. Deze proxy kan geconfigureerd worden in het proxy.conf.json bestand.

5.4.5 Klassendiagram

Enkele dagen voor de deadlline werd ook het klassendiagram bijgewerkt. Voor de finale versie zouden pijlen moeten worden getrokken tussen elke klasse die elke klasse gebruikt, zoals het hoort. Dus tussen elke klasse die ooit een bepaalde klasse gebruikt, werd een pijltje getrokken.
Dit resulteerde in een immens spinnenweb aan lijntjes waardoor het gebruikte programma, Visual Paradigm, crashte.
De fout in de gedachtegang is eenvoudig. Stel dat een gebruiker een auto heeft en die auto heeft wielen en een stuur. Dan werd in het klassendiagram de gebruiker verbonden met zowel auto, wielen als stuur. Dit klopt natuurlijk niet.
Daarom werden er in het klassendiagram enkel “hoofdklassen” met elkaar verbonden. Ook al zou een persoon bijvoorbeeld zijn banden oppompen, hij zou nog steeds niet worden verbonden met de wielen en enkel met de auto. Dit omdat het dan onhoudbaar is om de pijlen netjes te structureren.

5.4.6 Internationalisatie

Er werd gevraagd om zowel een Nederlandse als Engelse versie (met mogelijkheid tot uitbreiding) te voorzien in de applicatie. Aangezien dit pas in sprint 3 werd geïmplementeerd was het grootste deel van de applicatie al uitgewerkt, met een niet zo goede ondersteuning voor internationalisatie.
Daardoor moest er soms creatief omgegaan worden met de manier waarop sommige elementen vertaald werden weergegeven, zonder dat dit een nadelig effect zou hebben op de werking van de applicatie.

[1] http://www.jsonschema2pojo.org/

6 HANDLEIDINGEN

6.1 Aanpassen van de code

6.1.1 Installatie project

Er zijn zip's beschikbaar waarin de volledige applicatie zit. Deze kunnen op een server worden geplaatst, waarover later meer. Indien echter nog modificaties nodig zijn, kan men deze doen in een IDE naar keuze, hier zal IntelliJ IDEA gebruikt worden. Deze IDE kan dienen als server in ontwikkelingsfase. Als startproject dient het project van op https://github.ugent.be/bp-vop-2019/mystaff te worden gecloned.

1. Download & installeer IntelliJ Ultimate(everything)
2. Copy de link via de repository "https://github.ugent.be/bp-vop-2019/mystaff"
3. Importeer project in IntelliJ, niet openen met version control!
4. Selecteer sbt als build tool (standaard voor Java Play 2 projecten), druk meerdere keren op'next' en ten slotte op finish
5. Hierna zal het project een tijdje compileren. Als het geladen is, dient nog in 'File->Settings->Languages & Frameworks->Play2->Use Play2 Compiler for this project' geselecteerd te worden en vervolgens IntelliJ te herstarten.

6.1.2 Database

De applicatie heeft een database nodig om te kunnen functioneren. In ons geval gebruiken we MariaDB. Elke andere SQL-gebaseerde databank volstaat ook. Wanneer een SQL-databank naar keuze is opgestart, kan men de juiste tabellen laten genereren met behulp van het opstartscript dat zich bevindt op de gitHub-repo onder de naam: "mystaff/MystaffApp/conf/evolutions/default/1.sql". De bovenste helft van die file bevindt zich onder de naam "ups" en vormt de reeks commando's die dienen te worden ingegeven om de nieuwe tabellen te generen. Onderaan, onder de naam "downs" zijn de tegengestelde commando's te vinden, die de databank opnieuw leegmaken. Wanneer het deel onder ups gebeurd is, is de database klaar voor gebruik. Alternatief kan men ook "mystaff/docker/mystaff-mariadb/init.SQL" runnen. Dit script zorgt ervoor dat de naam van de gebruikte database "iiidb" wordt.

Nu moet er nog communicatie kunnen gebeuren tussen de database en de back-end. Daarom moet men bezitten over de nodige drivers van de gebruikte databank. Deze vindt men snel op Google. Wanneer een JAR-bestand met de driver erin is gevonden, kan men deze plaatsen in de map "mystaff/MystaffApp/lib/". Wanneer het bestand "mariadb-java-client-2.4.0.jar" nog aanwezig is in deze map, en geen mariadb-databank wordt gebruikt, mag dit verwijderd worden.

Nu moet de back-end nog weten wat de inloggegevens zijn van de databank. Hiertoe moet "mystaff/MystaffApp/conf/application.conf" worden aangepast. Control-F'en naar "default.driver" brengt je snel naar de nodige plaats. Hier moeten de volgende zaken worden aangepast:

- default.driver: dit heeft te maken met de gebruikte JAR. De naam die hier moet gebruikt worden, vindt men normaal op de plaats waar men de JAR vond. Vb: org.mariadb.jdbc.Driver
- default.url: de connectie url van de database. Deze is afhankelijk van de soort database die men gebruikt, van het ip-adres en de poort waarop ze draait en de naam van de gebruikte databasegroep. Vb: "jdbc:mysql://192.168.99.100:3306/iiidb"

- default.username en default.password: de gebruikersnaam en het wachtwoord van de gebruikte database

6.1.3 Front end

De communicatie url naar de back end wordt gegeven in het configuratiebestand “mystaff/MystaffApp/ui/src/assets /config.dev.json”. De url naar de back end wordt voorgesteld door de variabele “backend.url”. Vb: “http://localhost:”. Merk hierbij de aanwezigheid van het protocol (http) en de afsluitende dubbelpunt op. De poort van de back end wordt dan weer voorgesteld door de variabele “backend.port” en is in ons voorbeeld 9000.

De front end kan makkelijk worden gebuild door in IntelliJ in de terminal onderaan zich te begeven naar Mystaffapp/ui en daar het commando “ng build –prod” in te geven.

Nu bevindt zich een compacte html- en js-voorstelling van de frontend in de map “GITMAP\MystaffApp\ui\dist\java-play-angular-seed”. Deze kan dan worden gedraaid op een webserver naar keuze.

6.1.4 Lopen in testfase

Wanneer dit allemaal gebeurd is, kan je de server starten door eenvoudigweg onderaan in IntelliJ de sbt shell aan te klikken en wanneer die opgestart is, het commando ‘run’ in te voeren. De sbt shell wordt standaard toegevoegd aan IntelliJ bij een Java Play project. Bij het runnen zal ook de back end worden geladen vanaf de eerste call van front naar back end.

6.1.5 Compileren voor productiefase

Ook de back end kan worden gebuild naar een compact archief. Dit doet men door in de sbt terminal onderaan het commando “dist” in te voeren. Na enkele minuten wachten bevat de map “MystaffApp\target\universal” nu het bestand “java-play-angular-seed-1.0-SNAPSHOT”. Dit bestand kan worden geladen op een java-server naar keuze. Belangrijk is dat minstens de JVM arguments “-Xms1G -Xmx2G -Xss4M” worden meegegeven. Zonder Xss zal de recursie niet diep genoeg kunnen gaan en wordt er steeds een stackOverflowException gegooid wanneer de gebruiker newsfeed opent.

6.1.6 Versies

De minimale vereiste versie die compatibel is voor Angular in dit project is Angular 5, maar de aanbevolen versie is Angular 6 en hoger. De reden dat Angular 5 ook werkt voor een Angular 6 project ligt aan het feit dat beide versies bijna identiek aan elkaar zijn waardoor versie 6 projecten achterwaarts compatibel zijn met versie 5.
Voor de backend is er gebruik gemaakt van de Java Development Kit versie 1.8.0_211.

6.2 Productiefase zonder Docker

6.2.1 Korte beschrijving

Wanneer het eindproject van op github gebruikt, of men enkele aanpassingen heeft gedaan en gecompileerd via IntelliJ, kan men de http- en de java- files nu deployen op een server.
De http-files bevinden zich onder “GITMAP\MystaffApp\ui\dist\java-play-angular-seed”.
De java-zip is “GITMAP\MystaffApp\target\universal\java-play-angular-seed-1.0-SNAPSHOT”.
De myStaff-applicatie kan draaien op elk besturingssysteem.

Een databank dient men te deployen zoals in 5.1.2.

6.2.2 Configuratie van front end

Het configuratiebestand van de front end vindt men onder “assets/config/config.dev.json”. Er zijn slechts twee parameters aanwezig.

- backend.url: de url naar de back end. Vb. “http://localhost:”. Merk hierop de “http://” vooraan en de “:” achteraan op.
- backend.port: de poort van de back end. Vb. “9000”

Wanneer dit ingesteld is, kunnen de files gewoon worden geplaatst op een http-server.

6.2.3 Configuratie van de back end

Het configuratiebestand van de back end vindt men in de zip als “conf/application.conf”.
Hierin zijn zowel parameters die product specifiek zijn, als parameters die niet moeten worden aangepast. Enkel de product specifieke worden overlopen.

6.2.3.1 Parameters ivm databank

Zie hiervoor 6.1.2

6.2.3.2 Parameters in verband met mails

- mails.server: de gebruikte mailserver bv “smtp.gmail.com”
- mails.serverPort: de gebruikte poort van de server bv “587”
- mails.account: het e-mailadres vanwaar de mails worden gestuurd bv “mystaffexample@gmail.com”
- mails.password: het wachtwoord van het bovenstaande account “vakantie@5”
- mails.doSend: moeten er mails worden gestuurd?
- mails.doSendToPlanner: moeten er mails worden gestuurd naar de planner bij elke nieuwe aanvraag?
- mails.LogSentMails: moet er gelogd worden als er een mail wordt gestuurd onderaan?

6.2.3.3 Parameters in verband met het gebruikte bedrijf

- axians.tenant: de aangeboden tenant-code door Axians bv “ugent2019”
- axians.standardMaxAbsenceDays: hoeveel dagen een persoon mag verlof nemen per jaar bv “35”

6.2.3.4 Andere parameters

- mystaff.doLogRequests: moeten de gedane GET en POST requests worden gelogd?

6.2.4 Opzetten van de back end

Om de java back end op te zetten, dient men enkel de zip uit te pakken op de java server. Starten van de server kan dan met het script dat zich bevindt onder de naam: “bin/ java-play-angular-seed.bat”.

6.3 Productiefase met Docker

6.3.1 Korte beschrijving

Om geen last te hebben van alle verschillende servers, en deze te concentreren op één plaats, kan men gebruikmaken van een Docker container.

Om deze Docker container werkende te krijgen, dient men het project van op GitHub te clonen.
De bestanden die hier nodig zullen zijn vindt men in de “/docker” map.
Er zal worden gebruikgemaakt van een Linux terminal voor bepaalde commando’s. Makkelijkst is het wanneer men hiervoor de Docker Quickstart Terminal[1] gebruikt. Deze zal ook gebruikt worden voor het compileren van de finale containers.
Er werd gebruik gemaakt van de Docker client versie 18.03.0-ce en de Docker server versie 18.09.2. Deze versies kan men bekomen via het commando “docker version” in de Docker Quickstart Terminal.

6.3.2 Front end builden

De front end kan worden gebuild met bv. IntelliJ zoals beschreven in 6.1.3. Wanneer dat gebeurd is, kan men de files van “GITMAP\MystaffApp\ui\dist\java-play-angular-seed” plaatsen in “GITMAP\docker\mystaff-http\src”.
De laatste versie van het project staat daar met de standaardparameters klaar. Indien nodig kunnen enkel de parameters er analoog aan 6.2.2 worden aangepast.
Zo dient de url in het configuratiebestand “http://192.168.99.100:” te zijn wanneer de front end gelinkt is aan een back end op het zelfde Windowssysteem, dit gebeurt dus enkel in testfase. Bij Linux kan dit worden ingesteld met als ip-adres “localhost”. Wanneer de back end draait op een andere computer, dient de juiste url naar die webserver te worden gebruikt, bv. “http://bpvop5.ugent.be:”.

6.3.3 Back end builden

De back end kan gebuild worden met bv. IntelliJ zoals beschreven in 6.1.5. Merk op dat de JVM argumenten niet ingesteld moeten worden wanneer men een docker container gebruikt. Nu dient men de zip bekomen in “GITMAP\MystaffApp\target\universal” te kopiëren en te plaatsen in “GITMAP\docker\mystaff-java". Indien de map src er al bestaat, dient deze te worden verwijderd. Nu dient men in een Linux terminal zich te begeven naar de map “GITMAP\docker\mystaff-java” en er het commando “bash unzipJarScript” uit te voeren. Nu bevindt zich in die map in de map “src” de inhoud van de zip, op een speciale manier uitgepakt.
De laatste versie van het project staat daar met de standaardparameters klaar. Indien nodig kunnen enkel de parameters er analoog aan 6.2.3 worden aangepast.
In application.conf hoort voor een Docker container een url analoog aan “jdbc:mysql://192.168.99.100:3306/iiidb” te worden ingegeven indien men werkt in een Windows omgeving. In een Linux omgeving dient men een url zoals “jdbc:mysql://localhost:3306/iiidb” in te geven.

6.3.4 Algemene Docker opties

De algemene Docker opties kunnen worden gevonden in “GITMAP \docker\project.yml”.
Hierin kan men de poorten aanpassen waarop bepaalde containers worden geëxporteerd. Ook de inloginstellingen van de databank kunnen worden ingesteld. De naam van de database kan ook worden aangepast (standaard “iiidb”) maar dient hetzelfde te zijn als de naam vermeld in het bovenste lijntje van “GITMAP \docker\mystaff-mariadb\init.sql” alsook in eldersvermelde url’s die het woord “iiidb” bevatten. Verder vindt men ook de eerder vermelde JVM argumenten in deze file.

[1] https://docs.docker.com/toolbox/toolbox_install_windows/

6.3.5 Starten van de Docker container

Dit is redelijk simpel. Men dient enkel via de Docker Quickstart Terminal zich te begeven naar "GITMAP\docker" en er het commando "bash buildandrun.sh"[1] uit te voeren. Wanneer het eerder genoemde script geen fouten opgooit, er dus vier keer DONE staat na het uitvoeren van het script, dan is alles normaal gezien in orde.
Merk op dat het opstarten van de Java container iets langer kan duren dan de andere containers. Dit omdat deze moet wachten tot de database is opgestart.

6.3.6 Onderhoud van de Docker container

Men kan testen of effectief alle containers lopen door het commando "docker container ls" uit te voeren. Logs van een bepaalde container kan men bekomen door "docker logs <containernaam>" uit te voeren. De containernamen ziet men via het ls commando. Bv "docker logs docker_mystaff-java_1". Indien een container crashte, kan men de naam ervan terugvinden via "docker container ls -a". Dit commando geeft alle, en dus ook de niet-lopende, logs.

6.4 Software op clientside

Aangezien het hier om een website gaat, heeft de client enkel een browser naar keuze nodig.
Inloggen kan met accounts aangeboden door Axians.
Indien geen toegang werd gegeven door Axians, is het mogelijk om eens een kijkje te nemen op de applicatie met het test-account met gebruikersnaam "**demo1**" en wachtwoord "**vakantie**".

6.5 Gebruikershandleiding.

6.5.1 Dokter

6.5.1.1 Verlof aanvragen

Om verlof aan te vragen, kan een dokter surfen naar /calendar en op "Plan afwezigheid" klikken. Rechts kan hij dan de gegevens van zijn aan te vragen verlof ingeven. Alternatief kan men in de kalender op een dag klikken en worden deze gegevens rechts ingevuld.

6.5.1.2 Aangevraagde verloven bekijken

Om aangevraagde verloven te bekijken, kan een dokter surfen naar /calendar. Nu ziet hij in de kalender op welke dagen reeds verlof werd aangevraagd. Aan de hand van het icoontje bij een dag (vinkje, zandloper of kruisje) weet een dokter of de dag goedgekeurd, in beraad of afgekeurd is.
Een aanvraag zal slechts aan de dokter worden voorgesteld als goedgekeurd wanneer een planner van elke planningseenheid waar hij lid van is, heeft goedgekeurd.
Wanneer de dokter hovert over het icoontje, krijgt hij informatie over het aantal planningunits waarin hij al is goedgekeurd.
Onderaan de pagina staat ook een handig tabeloverzicht waarin ook alle aangevraagde verloven staan.

[1] Dit opstartscript werd samen met het unzip-script voorzien door de begeleidende coaches.

6.5.1.3 Verlof wijzigen

Om een aangevraagde verlofperiode te wijzigen, kan een dokter surfen naar /calendar. Hier kan hij de aanvraag in kwestie aanklikken. Vervolgens komt er een zwart balkje onder de dag waarin de dokter op “Wijzigen” kan klikken. Rechts vindt hij nu de gegevens terug van die bepaalde verlofperiode. Wanneer hij die aanpast en op “Aanvragen” klikt, wordt de nieuwe aanvraag terug naar de planner gestuurd om al dan niet goed te keuren.

6.5.1.4 Verlofteller bekijken

Om de eigen verlofteller te bekijken, kan een dokter surfen naar /calendar. Wanneer hij klikt op een willekeurige dag, ziet hij onder de invulvakjes om verlof aan te vragen enkele tellers. Hij ziet hier het aantal dagen dat hij maximum mag opnemen, alsook het aantal goedgekeurde verlofdagen. Verder ziet hij ook hoeveel dagen hij heeft verbruikt indien al zijn verloven die nog in aanvraag staan zouden worden goedgekeurd.

6.5.1.5 Verlof bekijken van teamgenoten

Om de planning te bekijken van teamgenoten, kan de dokter surfen naar /teamplanning. Hier kiest hij eerst bovenaan het team waarvan hij graag de planning wil zien. Nu krijgt hij de goedgekeurde aanvragen te zien die net als bij zijn eigen planning in kalender- en feedvorm worden gepresenteerd.

6.5.2 Planner

Alles wat volgt, speelt zich af in /newsfeed.

6.5.2.1 Aanvragen bekijken

De planner ziet rechts een lijst van alle lopende aanvragen die hij nog niet verwerkt heeft. Bovenaan de lijst kan hij de aanvragen kiezen binnen een planningseenheid.
Binnen een aanvraag staan de naam, de skills en de locaties waar de dokter werkt, het tijdstip van de aanvraag, id van de aanvraag en het type verlof dat een dokter heeft aangevraagd met eventueel commentaar van de dokter zelf.
Klikken op het plusje levert alle informatie die eerst niet binnen de lijntjes kon, net als de aanvraagdatum en de conflicten met de andere aanvragen binnen een planningseenheid.

6.5.2.2 Reageren op een aanvraag

Na het bekijken of de aanvraag mag worden goedgekeurd (zie volgende alinea’s), kan de planner een beslissing nemen. Een aanvraag kan worden goedgekeurd, in wacht worden gezet of afgekeurd worden. Bij dat laatste verschijnt er eerst een box met een tekstveld waar de planner nog commentaar kan bijgeven waarom hij die beslissing heeft gemaakt. Als iemand meer dan 10 keer heeft voorgehad (binnen één jaar) dat zijn verlof werd afgekeurd, verschijnt er hiervan een melding.

6.5.2.3 Bekijken of een aanvraag botst met andere aanvragen

In de rechtse lijst staat elke aanvraag omlijnd door een kleurtje.
Is de kleur paars, dan is de berekening nog bezig, deze kan enkele seconden duren. Er is ook een draaiend laad-symbool te zien gedurende deze periode.
Is de kleur groen, dan zorgt de aanvraag nooit voor problemen. Deze kan worden goedgekeurd.
Is de kleur rood, dan zorgt de aanvraag altijd voor problemen. Deze mag en kan niet worden goedgekeurd.

Is de kleur oranje, dan zorgt de combinatie van enkele aanvragen voor een probleem. Wanneer bv. één persoon nodig is op een dag en de twee mogelijke personen die dag beide verlof vragen, dan vormt dit een probleem. De planner ziet dat de botsing tussen deze twee aanvragen gebeurt. De planner kan nu kiezen welke van de twee hij zal goedkeuren. Na goedkeuren van de ene oranje aanvraag, zal de aanvraag die samen met deze aanvraag oranje was, rood worden, en niet meer kunnen worden goedgekeurd.

6.5.2.4 Bekijken hoeveel mensen op een dag nodig zijn en hoeveel er op verlof kunnen

In de kalender ziet de planner telkens enkele getallen per dag.
Rechtsboven staat de dag van de maand, dit heeft niets te maken met het probleem.

Voor wat volgt werd een planning gegenereerd die voldoet aan alle constraints. Omdat er meerdere planningen mogelijk zijn, werd de planning gekozen met het meest aantal reserve personen. Dit zijn personen die naast het benodigde aantal ook zouden kunnen worden ingepland. De berekening hiervan duurt normaal ongeveer een seconde.

Verder staan er twee rijen van cijfers per dag, één rij voor elk dagdeel, voor- en namiddag.
Deze rijen zijn gestructureerd als volgt: “x y-z”.
Het getal x stelt voor hoeveel personen in totaal nodig zijn tijdens dat deel van de dag. Hierover hoveren met de muis toont welke personen werden ingepland in de berekende planning in elke shift.
Het getal y toont aan wat het minimum aantal personen zijn dat op verlof kunnen worden gestuurd. Dit kan nul zijn als er een persoon is die niet op verlof mag worden gestuurd. Dit valt voor wanneer er bv. één persoon nodig is voor een shift en slechts één persoon over de nodige skills bezit.
Het getal z toont aan hoeveel mensen maximum op verlof kunnen worden gestuurd.

Wanneer men bijvoorbeeld “3 1-6” ziet staan, dan kan men het volgende concluderen:

- Er zijn dat dagdeel drie mensen nodig.
- Wanneer één iemand verlof vraagt die dag, kan dat altijd worden goedgekeurd
- Wanneer twee mensen verlof aanvragen, is het mogelijk dat het kan, maar het kan ook zijn dat het niet mogelijk is. Verder onderzoek is nodig.
- Wanneer 7 mensen verlof aanvragen, kunnen ze sowieso niet allemaal worden goedgekeurd.

6.5.2.5 Gebruik van de sorteren, filters, favorieten en zoeken

Voor een beter overzicht kan de aanvraaglijst worden gesorteerd op verschillende criteria: volgens aanvraag datum, op voornaam en achternaam, naam van skill, datum van en locaties (zowel oplopend als aflopend).

Via de filters kan de planner aanduiden welke aanvragen hij wil zien in de kalender. Hij kan skills, dagdelen, locaties, types verlof en wel of geen commentaar selecteren. Enkel wat hij aanduidt, zal worden getoond in de kalender.
Deze filters kunnen eventueel ook worden toegepast op de aanvragen (verder meer).

Ook is er de favorieten component. Een gekozen favoriet zal zich steeds bovenaan in de lijst van de aanvragen plaatsen, dit ongeacht welke filter geselcteerd is.

Als laatste is er de zoekfunctionaliteit, hierin kan je o.a. een naam ingeven en dan zullen enkel goedgekeurde aanvragen worden getoond met die naam. Hiernaast kan ook op locatie, skill en dagdeel worden gezocht. De zoekfunctionaliteit is niet hoofdlettergevoelig.

6.5.2.6 Goedkeuren van aanvragen

Het goedkeuren van aanvragen loopt niet altijd even evident, zelfs niet met filters. Daarom is er een extra functionaliteit ingebouwd om voor de aanvragen met conflicten een keuze te maken die de planner het best ligt. Deze functionaliteit heet ‘test’, bij elke aanvraag zal er een ‘test’-knopje komen te staan. Deze stelt de planner in staat om na te gaan wat het effect zou zijn, mocht hij verschillende aanvragen goedkeuren.

Een voorbeeld: drie oranje omringde aanvragen veroorzaken samen een conflict, de planner drukt voor de eerste twee aanvragen op test-knop en de test geeft een groen vinkje terug, wat wijst op geen conflict. Stel de derde aanvraag ook wordt meegetest, en alle vinkjes springen op een rood kruis, wijst dit op een onmogelijke combinatie.

6.5.2.7 Indienen van aanvragen door een planner

Een planner kan ook aanvragen indienen voor dokters. Onder Favorieten staar er een knop ‘Plan iemand in’. De werking is hetzelfde als in /calendar met het verschil dat er uiteraard een naam moet opgegeven worden.

6.5.2.8 Instellingen

De planner wordt nog voorzien van extra functionaliteit met deze knop. Met instellingen kan hij bv. kiezen of de aanvragen worden meegefilterd op de kalender. Of hij kan wisselen van overzicht en uit de planner-modus gaan.
Mee filteren met de aanvragen is evident: hij klikt op de schakelaar en de aanvragen zullen synchroon meegefilterd worden met de kalender. Wisselen van overzicht, zal de kalender wisselen met een overzicht van alle personen (links) en de kalenderdagen (bovenaan). Een gekleurd vakje wijst op een aanvraag die al dan niet is goedgekeurd. Een laatste instelling is de planner modus, hiermee gaat de planner een overzicht krijgen zoals een niet-planner die krijgt.

Stel dat er problemen zijn in de omgeving en je wilt terug naar de fabrieksinstellingen. Dan druk je op ‘Herstel Instellingen’, onderaan in de instellingen pop-up.

6.5.2.9 Planner modus

Zoals de naam doet vermoeden is er een tweede versie van de /newsfeed omgeving. Deze is de omgeving van de dokter. Enkel een planner kan hier tussen switchen.

- De ‘plan iemand in’-knop is nu een ‘plan jezelf in’-knop geworden.
- Rechts worden geen aanvragen meer weergegeven maar wel de updates van je eigen aanvragen.

  De updates component toont een lijst met lopende, goedgekeurde en geweigerde aanvragen van de dokter zelf die gesorteerd staan op de laatste wijziging. Dit kan handig zijn voor de dokter om te weten wie wanneer op verlof gaat, hoe het met jouw aanvragen staat, en als hij wil kan hij zichzelf direct inplannen met de ‘plan jezelf in’-knop.

6.5.2.10 Help

Indien er onduidelijkheden zijn, bevindt er zich onderaan (rechts) een knop met een vraagteken waarin informatie staat over de aanvragen en de kalender.

7 BESLUIT

7.1 Realisaties

Toen wij eerst de opdracht van Axians te horen kregen, dachten wij dat dit nog een redelijk simpel project te zijn, waar wij niet zo veel tijd in konden steken. Maar na de eerste meeting bleek al snel dat er zeer veel onder de motorkap zat. Axians wou een module waar dokters verlof konden aanvragen, waarna de planner deze aanvragen kon bekijken en mogelijks kon goedkeuren. Hierbij moet hij kunnen gebruiken van allerlei hulpmiddelen om dit zo makkelijk en intuïtief mogelijk te maken. Alle features die Axians heeft gevraagd, hebben wij correct kunnen implementeren. Wij hebben geprobeerd om nog enkele handige extra functies in de applicatie te steken, zoals het tonen van de historiek van de aanvragen, of het opstellen van de ideale planning via een backtracking algoritme. Deze zaken namen veel tijd in beslag, maar wij zijn ervan overtuigd dat dit zeer handige extra features zijn zodat dokters en planners met plezier gebruik kunnen maken van de website. Van Axians kregen wij ook geen template hoe het eruit moest zien, aangezien de vorige versies niet echt in orde waren. Hier hebben wij alle vrijheid in gekregen en volgens ons is er geen eenvoudigere en functionelere weergave mogelijk. Tijdens het project konden wij erg veel bijleren over allerlei technologiëen en hulptools waar wij nog geen ervaring mee hadden. Dit alles zorgt ervoor dat wij zeer tevreden zijn over ons project en het met trots kunnen afsluiten.

7.2 Referenties

DanielDaniel. (n.d.). Why Play Framework 2.5 doesn't respect JVM memory settings in sbt? Retrieved from https://stackoverflow.com/questions/41116318/why-play-framework-2-5-doesnt-respect-jvm-memory-settings-in-sbt

DrYapDrYap 4, & Adrian ShumAdrian Shum 29.1k763107. (n.d.). Variable Number of Nested For Loops. Retrieved from https://stackoverflow.com/questions/18165937/variable-number-of-nested-for-loops

Finding all subsets of a given set in Java. (2018, May 30). Retrieved from https://www.geeksforgeeks.org/finding-all-subsets-of-a-given-set-in-java/

Home. (n.d.). Retrieved from https://www.playframework.com/documentation/2.6.x/JavaJsonActions

How to send HTTP request GET/POST in Java. (n.d.). Retrieved from https://www.mkyong.com/java/how-to-send-http-request-getpost-in-java/

Lizard, B. T. (n.d.). How can I send an email by Java application using GMail, Yahoo, or Hotmail? Retrieved from https://stackoverflow.com/questions/46663/how-can-i-send-an-email-by-java-application-using-gmail-yahoo-or-hotmail

Rob. (n.d.). Android HTTPUrlConnection : How to set post data in http body? Retrieved from https://stackoverflow.com/questions/20020902/android-httpurlconnection-how-to-set-post-data-in-http-body

Suishsuish 2, & Planetenkillerplanetenkiller 729515. (n.d.). How to increase stack size of play framework activator? Retrieved from https://stackoverflow.com/questions/33295415/how-to-increase-stack-size-of-play-framework-activator

Yohangz. (2019, March 27). Yohangz/java-play-angular-seed. Retrieved from https://github.com/yohangz/java-play-angular-seed

Principes van de vakantieregeling in het onderwijs. (n.d.). Retrieved from https://onderwijs.vlaanderen.be/nl/principes-vakantieregeling

Wesolowski, D. W. (n.d.). Calculate the date of Easter Sunday. Retrieved from https://stackoverflow.com/questions/26022233/calculate-the-date-of-easter-sunday/26022891

Angular Material: Table (n.d.). Retrieved from https://material.angular.io/components/table/overview

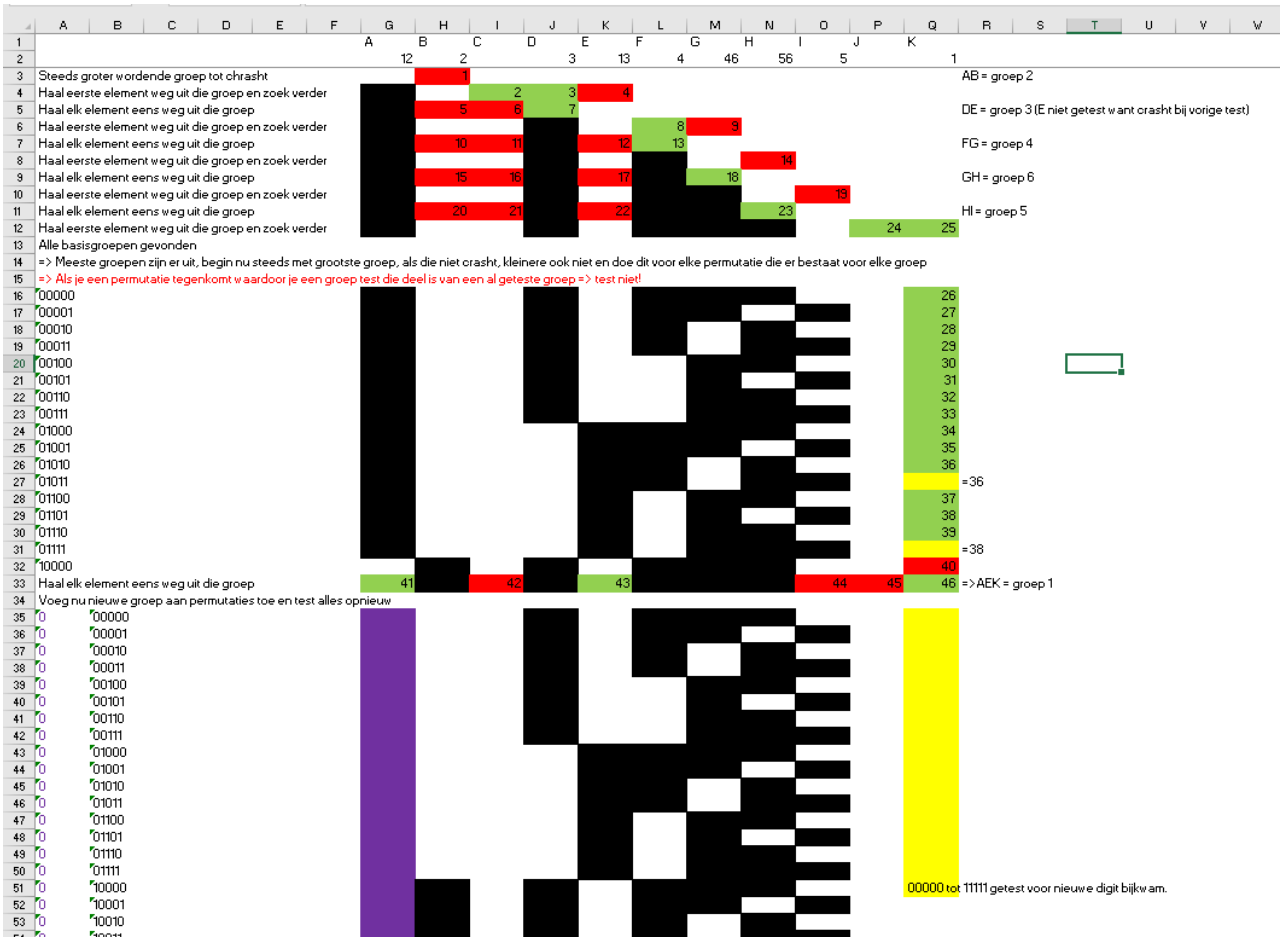
Angular Material: Icons (n.d.). Retrieved from https://material.io/tools/icons/?style=baseline

Angular Quickstart (n.d.). Retrieved from https://angular.io/guide/quickstart

Mattlewis92. (2019, April 27). Mattlewis92/angular-calendar. Retrieved from https://github.com/mattlewis92/angular-calendar#getting-started

8 BIJLAGEN

8.1 Bijlage 1: Visuele voorstelling algoritme 2.1.2.3



Legende:
Rood: algoritme crasht, er bevindt zich een volledige foute groep in de geteste subgroep.
Groen: algoritme crasht niet, er bevindt zich geen volledige foute groep in de geteste subgroep.
Zwart en paars: Element werd uit de te testen groep gehaald.
Geel: Groep wordt niet getest, is subgroep van andere groep.
De nummertjes tellen hoeveel keer een test is gebeurd.
Op rij 3 start het algoritme en worden A en B samen getest, het resultaat is een fout. A en B vormen dus een groep. Vanaf nu wordt A dus uit de te testen groep gehaald.
Op rij 4 worden ABC, ABCD en ABCDE respectievelijk getest, pas bij ABCDE crasht de groep
Op rij 5 wordt de groep ABCDE getest zonder telkens elk element, enkel als D eruit gelaten wordt, crasht de groep niet, DE vormt dus een groep.
En zo verder.

De Excelfile is ook te vinden in de root van de repo onder de naam “uitwerking algoritme 2.1.2.3”
Een PowerPoint met daarin een animatie van dit algoritme is ook te vinden op gitHub.

8.2 Bijlage 2: Uitgezoomde voorstelling algoritme 2.1.2.3