

# Eindverslag

Mobi02

22 mei 2017

## 1 Korte samenvatting

Reeds sinds 2014 is het Mobiliteitsbedrijf van stad Gent bezig met het opzetten van een regionaal verkeerscentrum. Het heeft als doel om het verkeer in de regio op verschillende vlakken continu te monitoren, en deze data vervolgens voor de bevolking ter beschikking te stellen. Echter is het zo dat op dit moment enkel Twitter gebruikt wordt om verkeersmeldingen te sturen, waardoor het publiek sterk beperkt wordt.

Deze bachelorproef heeft als doel dit gemis in te vullen, met name door het maken van een mobiliteitsdashboard. Alle informatie wordt hierbij verzameld en op een overzichtelijke manier weergegeven op een webpagina, toegankelijk voor elke internetgebruiker. Op dit ogenblik werd reeds een webpagina opgebouwd met een hele hoop informatie die in het verkeer van belang kan zijn, gaande van files en wegenwerken tot openbaar vervoer. In ontwikkeling is ook een persoonlijke pagina, waarbij de gebruiker kan inloggen en zijn voorkeuren kan doorgeven omtrent welke informatie hij wenst te zien.

## 2 Structuuroverzicht

### Inhoud

1 Korte samenvatting	2
2 Structuuroverzicht	3
3 Hoofdstuk 1: Inleiding	8
3.1 Context . . . . .	8
3.2 Probleemstelling . . . . .	8
3.3 Doelstelling . . . . .	9
3.4 Structuur van het verslag . . . . .	9
4 Hoofdstuk 2: Gebruikersaspecten	10
4.1 High-level requirements . . . . .	10
4.1.1 Usability & Humanity requirements . . . . .	10
4.1.2 Operationele & Omgevingsrequirements . . . . .	11
4.1.3 Look & Feel Requirements . . . . .	11
4.1.4 Wettelijke Requirements . . . . .	12
4.1.5 Gebruikersbevraging . . . . .	12
4.2 Mock-ups . . . . .	13
4.3 Use case diagrammen . . . . .	16
4.4 Use cases tekstueel . . . . .	18
4.5 Backlog . . . . .	27
4.5.1 Sprint 1 . . . . .	27
4.5.2 Sprint 2 . . . . .	28
4.5.3 Sprint 3 . . . . .	30

5	Hoofdstuk 3: Systeemarchitectuur	32
5.1	High-level systeem model . . . . .	32
5.2	Klassendiagrammen . . . . .	35
5.2.1	Package diagram backend . . . . .	35
5.2.2	Frontend . . . . .	37
5.3	Sequentiediagrammen . . . . .	38
5.4	Activiteitendiagram . . . . .	46
5.5	Databank diagram . . . . .	48
6	Hoofdstuk 4: Testplan	50
6.1	Testplan frontend . . . . .	50
6.1.1	Integration testen . . . . .	50
6.1.2	Usability testen . . . . .	54
6.2	Testplan backend . . . . .	54
6.2.1	Automatische unit testen . . . . .	55
6.2.2	Integration testen . . . . .	56
7	Hoofdstuk 5: Evaluatie en discussies	57
7.1	Performantie . . . . .	57
7.1.1	Op component niveau . . . . .	58
7.1.2	Op systeem niveau . . . . .	58
7.2	Security . . . . .	59
7.3	Problemen en geleerde lessen . . . . .	59
8	Hoofdstuk 6: Handleidingen	62
8.1	Installatiehandleiding . . . . .	62
8.1.1	Beschrijving software . . . . .	62

8.1.2	Benodigdheden . . . . .	62
8.1.3	Installatie en configuratie van de database . . . . .	63
8.1.4	Stored Procedures . . . . .	63
8.1.5	Installatie en configuratie van de Glassfish-webcontainer . . . . .	64
8.1.6	Installatie van de applicatie . . . . .	66
8.2	Gebruikershandleiding frontend . . . . .	67
8.2.1	Gebruikershandleiding hoofdpagina . . . . .	67
8.2.2	Gebruikershandleiding persoonlijke pagina . . . . .	68
9	Hoofdstuk 7: Besluit	70
10	Referentielijst	71
11	Bijlagen	71

## Figuren

1	Mock-up hoofdpagina . . . . .	13
2	Mock-up instellingen widgets . . . . .	14
3	Mock-up persoonlijke pagina . . . . .	14
4	Mock-up persoonlijke pagina: keuzemogelijkheden . . . . .	15
5	Mock-up meldingen via Messenger . . . . .	15
6	Mock-up meldingen via mail . . . . .	16
7	Use case diagram sprint 1 . . . . .	17
8	Use case diagram sprint 2 . . . . .	17
9	Use case diagram sprint 3 (einde) . . . . .	18
10	Use case 1 . . . . .	19
11	Use case 2 . . . . .	20
12	Use case 3 . . . . .	21
13	Use case 4 . . . . .	22
14	Use case 5 . . . . .	23
15	Use case 6 . . . . .	24
16	Use case 7 . . . . .	25
17	Use case 8 . . . . .	26
18	Structuur applicatie . . . . .	32
19	Deployment diagram . . . . .	33
20	Overzichtsdiagram . . . . .	34
21	Package diagram . . . . .	36
22	Sequentiediagram cachingmechanisme . . . . .	39
23	Sequentiediagram De Lijn . . . . .	40
24	Sequentiediagram globaal . . . . .	42

25	Sequentiediagram notifications (vereenvoudigde versie) . . . . .	43
26	Sequentiediagram gedeelte Messenger notifications . . . . .	45
27	Sequentiediagram gedeelte mail notifications . . . . .	46
28	De Lijn widget bij aanvang . . . . .	47
29	Activiteitendiagram De Lijn widget . . . . .	47
30	Database tabellen . . . . .	49
31	Frontend zonder backend . . . . .	51
32	Online framework Assertible . . . . .	56
33	Monitoring VisualVM . . . . .	61
34	Glassfish-adminconsole . . . . .	64
35	Geslaagde ping . . . . .	66

## Tabellen

1	Backlog sprint 1 . . . . .	27
3	Backlog sprint 2 . . . . .	28
5	Backlog sprint 3 . . . . .	30
7	Overzicht frontend testen . . . . .	50
9	Overzicht backend testen . . . . .	55

## 3 Hoofdstuk 1: Inleiding

### 3.1 Context

Informatie omrent de verkeerssituatie in een stad is tegenwoordig niet meer weg te denken uit de maatschappij. De laatste jaren steeg het aantal voertuigen en weggebruikers in dergelijke mate, dat de nood er kwam om verkeerscentra op te richten. Er kan geconstateerd worden dat er dagelijks heel wat kilometers file op onze Vlaamse wegen staan, waardoor verplaatsingen voor vele mensen te veel tijd in beslag nemen.

Ook in Gent wou men niet achterblijven, en dus startte men in 2014 met het opzetten van een regionaal verkeerscentrum. Met de invoering van het nieuwe mobiliteitsplan heeft men hier des te meer nood aan. Het is dan ook van uiterst belang dat de bevolking geïnformeerd wordt over de verkeerssituatie in Gent. Vermits de technologie bij aanvang reeds heel wat beter was dan bij oprichting van het Vlaamse verkeerscentrum, opteren zij in tegenstelling tot het Vlaamse verkeerscentrum, voor het oprichten van een zoveel mogelijk geautomatiseerd centrum. Hierbij is het de bedoeling dat het verkeer in bepaalde regio's continu gemonitord wordt. Dit dient te gebeuren op semiautomatische basis op normale werkdagen, en kan extra bemand worden tijdens piekmomenten of bepaalde grote evenementen. Tijdens de week probeert men een automatische signalisatie te bekomen van onverwachte incidenten, calamiteiten of significante verhogingen van de reistijden.

Real time zou deze informatie aan iedereen die dit wenst ter beschikking gesteld moeten worden. Dit kan op verscheidene manieren gebeuren, met name via een website, sociale media, open data portalen en zelfs individuele berichten. Hiermee wil men de gebruiker de kans geven om steeds de beste route door Gent te nemen, alsook indien mogelijk het beste moment te kiezen om hun verplaatsingen te maken in en rond Gent.

### 3.2 Probleemstelling

Momenteel heeft men in het verkeerscentrum in Gent reeds heel wat gegevens ter beschikking, die automatisch via verscheidene wegen opgehaald worden en op een cloud-platform worden opgeslagen. Deze gegevens worden vervolgens verwerkt en uitgestuurd via het Twitterplatform @VerkeerGent. Bovendien wordt de data ook ter beschikking gesteld in ruwe vorm via het open dataportaal op de website van Stad Gent.

Dit alles heeft echter twee grote nadelen. Enerzijds moet de gebruiker die de informatie wenst te zien de Twitterpagina vaak raadplegen, en bijgevolg Twitter gebruiken. Anderzijds kan men wel de website raadplegen, maar deze data is vrij ruw en niet overzichtelijk. De aanwezige informatie bevat bovendien ook nog een hele hoop informatie die niet relevant is voor de gebruiker.

Kortom, er is nood aan een overzichtelijker manier van dataweergave waarbij niet relevante data gefilterd kan worden. Daarnaast is het ook van belang dat een groter publiek gebruik kan maken van de toepassing, waardoor procentueel meer mensen van de bevolking de kans krijgen de verkeersinformatie op te volgen.

### 3.3 Doelstelling

Zoals bij de probleemstelling aangehaald werd, is het nodig dat er een betere manier komt van dataweergave. In het algemeen zou men dit kunnen omschrijven als het weergeven van informatie op zo'n danige manier dat je in één oogopslag kunt zien hoe het met de mobiliteit gesteld is op dat ogenblik. Om dat basisprincipe waar te kunnen maken, moet bijvoorbeeld gedacht worden aan overzichtelijke grafieken die tonen hoe de situatie op dat moment afwijkt van de ‘ideale situatie’. Hierdoor kan de gebruiker onmiddellijk zien of de huidige situatie uitzonderlijk is of niet.

Een ander iets dat ook niet mag ontbreken, is het real time weergeven van de mogelijkheden tot openbaar vervoer in Gent. Zo kan de gebruiker meteen zien wat de meest aangewezen vervoersmethode is om zich te verplaatsen.

Een stap verder in de doelstelling, is dat de mogelijkheid geboden wordt om de informatie op maat weer te geven. Een gebruiker kan hierbij een account aanmaken, en keuzes maken omtrent welke informatie hen specifiek interesseert. Als de gebruiker vervolgens gaat inloggen, krijgt hij enkel en alleen deze informatiewidgets te zien die voor hem relevant zijn.

Tot slot kan ook nog uitgebreid worden op vlak van het op de hoogte brengen van de gebruiker. Er kan eventueel gedacht worden aan het toevoegen van een mogelijkheid om de gebruiker te laten kiezen om meldingen te ontvangen via sms of sociale media over de informatie die hij verkiest. Op die manier dient te gebruiker niet meer zelf te kijken bij het overwegen van een vervoersmiddel of route, maar wordt hij automatisch op de hoogte gebracht als iets anders loopt dan verwacht op zijn traject.

### 3.4 Structuur van het verslag

De corpus van dit verslag is opgebouwd uit meerdere onderdelen, allen deelaspecten van de omschrijving en analyse van het project – een mobiliteitsdashboard op maat van stad Gent – dat centraal stond in deze bachelorproef.

Eerst en vooral komen hierbij de gebruikersaspecten naar voor. Hierin vindt men de use cases, de high-level requirements alsook het backlog van het project. Vervolgens komt de systeemarchitectuur naar voren, waarbij onder meer het high-level systeemmodel geïllustreerd wordt met behulp van een deployment diagram. In dit gedeelte komen ook

de klassen- en sequentiediagrammen aan bod, die een inzicht geven over hoe het systeem opgebouwd werd. Nadien komt het testplan, waarbij een overzicht wordt gegeven van de voorziene testplannen. Dit wordt gevolgd met een stukje over evaluaties op vlak van performantie, security, schaalbaarheid en de geleerde lessen en problemen. Het laatste stukje van de corpus bevat een installatie- en gebruikershandleiding, om de distributie naar de klant op een eenvoudige manier te kunnen realiseren.

Tot slot volgt een kort besluit, gevolgd door een referentielijst van de geraadpleegde bronnen.

## 4 Hoofdstuk 2: Gebruikersaspecten

### 4.1 High-level requirements

De high-level requirements geven aan welke vereisten de klant had voor het project op verscheidene vlakken. Eerst wordt per type requirement de indicator, het meetvoorschrift en de norm gegeven, en vervolgens een korte beschrijving.

#### 4.1.1 Usability & Humanity requirements

- **Indicator** : Responsive voor verscheidene apparaten.
- **Meetvoorschrift** : Applicatie moet beschikbaar zijn op alle relevante (moderne) toestellen.
- **Norm** : Elke feature moet beschikbaar en bruikbaar zijn op elk toestel.

Vermits stad Gent een breed publiek wenst aan te spreken, is het uitermate van belang dat wanneer een website ter beschikking gesteld wordt, deze ook benut kan worden. De vereiste is uiteraard wel dat er een internetverbinding mogelijk is, maar de webpagina door meerdere browsers ondersteund wordt. Tijdens de ontwikkeling wordt voornamelijk met Google Chrome gewerkt, maar er wordt ook rekening gehouden dat bepaalde zaken op bijvoorbeeld Internet Explorer niet altijd draaien, en dan wordt een gepaste melding weergegeven.

Ook is het van belang dat verschillende types toestellen, zoals een laptop, smartphone, tablet of desktop, een aan het scherm aangepast formaat te zien krijgen. Kortom een responsive design is een must.

#### 4.1.2 Operationele & Omgevingsrequirements

- **Indicator** : Productieomgeving.
- **Meetvoorschrift** : Webapplicatie moet vlot draaien.
- **Norm** : De operaties moeten uitgevoerd kunnen worden zonder te grote vertragingen.

Het is van belang dat informatie die op het mobiliteitsdashboard weergegeven moet worden, binnen een relatief korte tijd op het scherm verschijnt. Bij een applicatie waar de gebruiker te lang dient te wachten vooraleer de informatie binnenkomt, zal de gebruiker snel afhaken.

Op het dashboard wordt de informatie weergegeven in verschillende widgets, waarin de informatie van een bepaalde API overzichtelijk weergegeven wordt. Om deze informatie in te laden, moet een API call gemaakt worden, wat in dit geval via een REST call gebeurd. Om te vermijden dat de webapplicatie te lang zou moeten wachten op bepaalde API's, worden alle widgets via een afzonderlijke call ingevuld. Er kon in testfase namelijk vastgesteld worden dat bepaalde API calls, zoals De Lijn, bijzonder lang duren en dit niet de ganse webapplicatie mocht vertragen.

Bovendien wordt de data ook gecachet van zodra de server opgestart wordt. Hierdoor is er heel snel data beschikbaar wanneer de gebruiker de webpagina opent. De gecachte data wordt met behulp van een timer op regelmatige tijdstippen bijgewerkt, zodat de informatie up to date blijft. De timeoutwaarde is specifiek ingesteld voor elke API, afhankelijk van de noden en de variabiliteit van de data.

#### 4.1.3 Look & Feel Requirements

- **Indicator** : Visueel overzichtelijk.
- **Meetvoorschrift** : Men moet duidelijk kunnen afleiden waarvoor alles dient.
- **Norm** : De stijl moet overzichtelijk zijn en alle informatie moet gestructureerd zijn.

Zoals in de doelstelling reeds beschreven werd, heeft de klant gevraagd om een weergave van data op een manier dat in één oogopslag duidelijk is hoe het gesteld is met het verkeer in Gent op dat moment. Het is dus van uiterst belang dat de structuur op een logische manier opgebouwd wordt, en de meest relevante data het duidelijkst naar voren komt. Zoals verder in dit verslag nog aan bod zal komen, werd een kleinschalige enquête gehouden, en op basis daarvan werd een basisstructuur van de widgets opgebouwd.

Hiervoor werd een modulaire structuur opgebouwd, waarbij voor elk type informatie een widget wordt weergegeven. Deze widgets staan standaard verdeeld in drie kolommen, maar de gebruiker krijgt de mogelijkheid om dit in meer of minder kolommen weer te geven. Bovendien kan de gebruiker een keuze maken in welke widgets voor hem van belang zijn. De optie bestaat om bepaalde widgets uit te zetten, en andere widgets dan weer meer naar boven of onderaan de pagina te plaatsen. Zo krijgt iedereen de kans om zijn eigen lay-out te bepalen.

Daarnaast wordt ook een kaartje getoond, zodat de gebruiker onmiddellijk ook op een meer grafische manier de data te zien krijgt. De gebruiker kan hierbij zelf kiezen welke informatie al dan niet op de kaart aangegeven wordt. Dit mede om te vermijden dat de kaart te druk en ongestructureerd zou worden.

#### 4.1.4 Wettelijke Requirements

- **Indicator** : Strikte afscherming.
- **Meetvoorschrift** : Alle data moet een geldige licentie hebben.
- **Norm** : Concrete afspraken met providers moeten gevuld worden.

Alle data die op de webpagina getoond wordt, wordt gehaald uit een bepaalde databron. Elke hoeveelheid data wordt via verschillende API calls opgehaald vanuit de verschillende databronnen. De databronnen zelf hebben bepaalde vereisten, bijvoorbeeld een maximaal aantal calls per uur of dag. Bovendien zijn bepaalde databronnen enkel ter beschikking tegen betaling, en heeft stad Gent met deze bedrijven een overeenkomst. Voor deze API's wordt door de server dan ook eerst een login call gedaan.

#### 4.1.5 Gebruikersbevraging

Naar aanleiding van de doelstelling van het project, het voorzien van informatie voor de inwoners en bezoekers van Gent, werd de vraag gesteld van wat willen de gebruikers eigenlijk zien. Hiervoor werd een kleinschalige enquête uitgevoerd, waarvan de resultaten in bijlage 1 worden weergegeven. Op basis hiervan kan een beter beeld gevormd worden van welke informatie de gebruiker het belangrijkste vindt, maar ook op welke manier ze graag informatie krijgen.

Er bleek uit de enquête dat de gevormde probleemstelling van dit project zeker en vast gegronsd is, namelijk dat de Twitteraccount waarop nu de meldingen worden gegeven niet echt gekend is. De enquête werd ingevuld door 47 personen, met leeftijden tussen de 15 en 57 jaar, en zowel inwoners van Gent als niet-inwoners.

Uit de enquête kon geconcludeerd worden dat het merendeel wel geïnteresseerd is in het krijgen van informatie omtrent het verkeer in Gent, en dat vooral de vertragingen van het openbaar vervoer, wegenwerken en parkeerplaatsen belangrijk zijn. Ook werd duidelijk dat veel gebruikers een mobiele applicatie zouden appreciëren, echter is dit geen onderdeel van de opdracht, en werd dit bijgevolg niet geïmplementeerd.

## 4.2 Mock-ups

In figuur 1 wordt er getoond wat de gebruiker bij navigatie op het eerste moment te zien krijgt. Van zodra de webpagina geopend wordt, worden alle beschikbare gegevens reeds ingeladen, zodat zoals bij de requirements beschreven, de gebruiker niet te lang dient te wachten. Om de gebruiker de mogelijkheid te geven deze hoofdpagina aan te passen, is er een instellingenknop, zichtbaar in figuur 2. Met de widgets zelf kan ook 'drag and drop' toegepast worden.

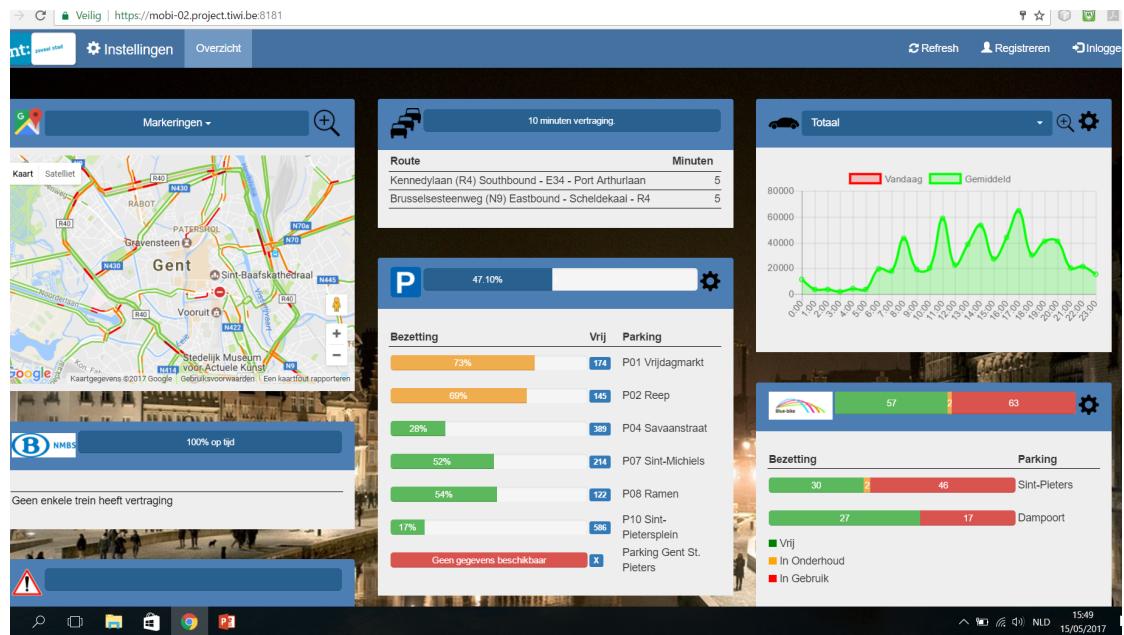


Fig. 1: Mock-up hoofdpagina

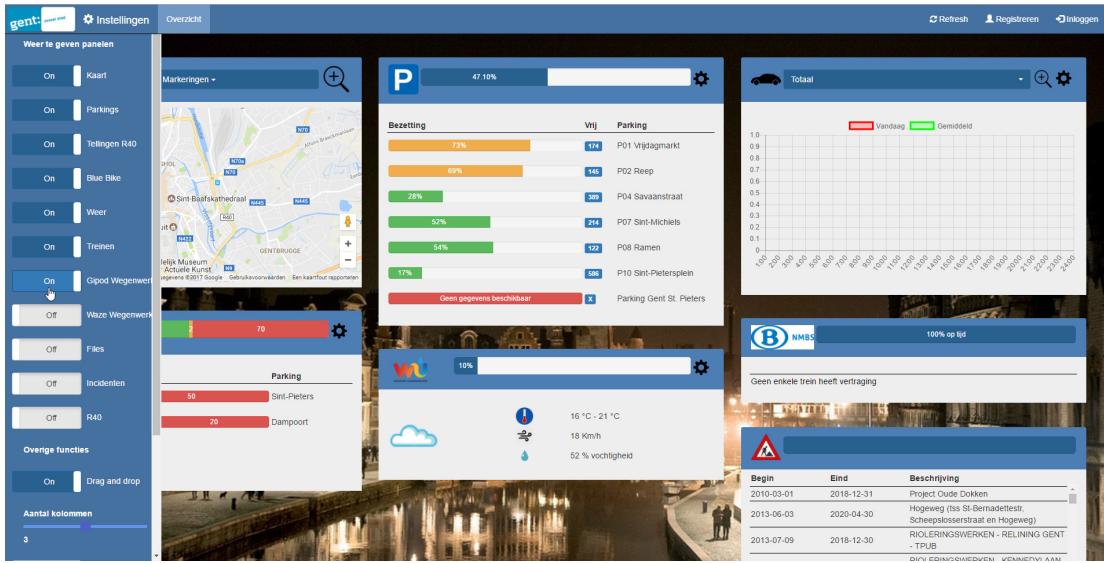


Fig. 2: Mock-up instellingen widgets

Vervolgens kan de gebruiker kiezen om te registreren, met behulp van een e-mailadres. Eens deze registratie doorlopen werd, kan de gebruiker inloggen en komt hij op zijn persoonlijke pagina terecht, zichtbaar in figuur 3. Een andere optie bestaat eruit om in te loggen met behulp van een Facebookaccount, waarbij dan automatisch de naam en de foto opgehaald worden.

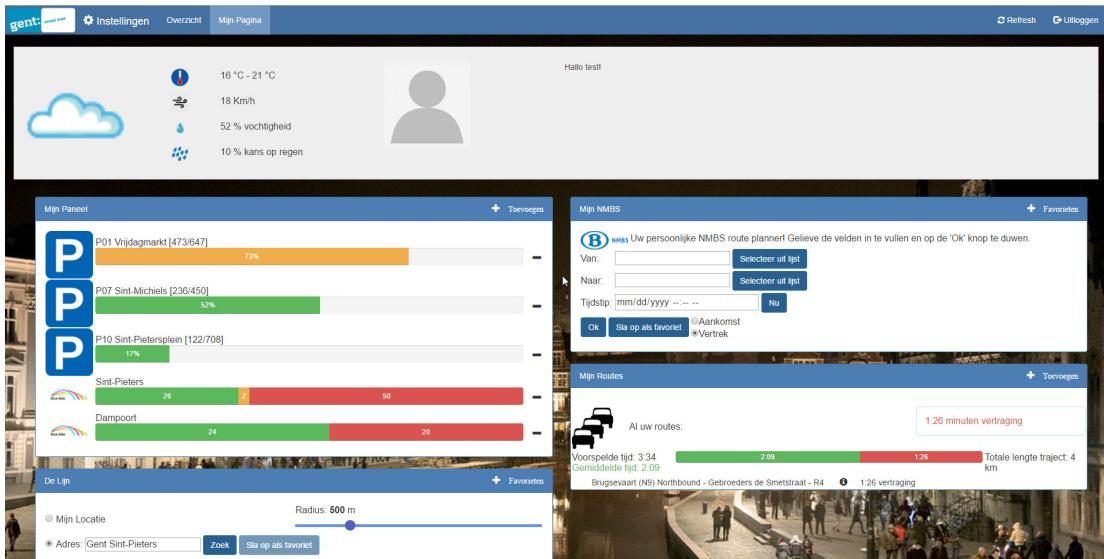


Fig. 3: Mock-up persoonlijke pagina

Indien ingelogd werd via Facebook, kan de gebruiker eventuele notifications instellen, zie figuur 4, die hij vervolgens zal ontvangen via Messengerberichten op de gewenste tijdstippen, geïllustreerd in figuur 5. Wanneer daarentegen ingelogd werd via gewone registratie, kan men deze notifications eventueel via mail opvragen. Dan zal de gebruiker, op de gewenste tijdstippen een mailtje krijgen met de gevraagde informatie, zoals het voorbeeld in figuur 6.

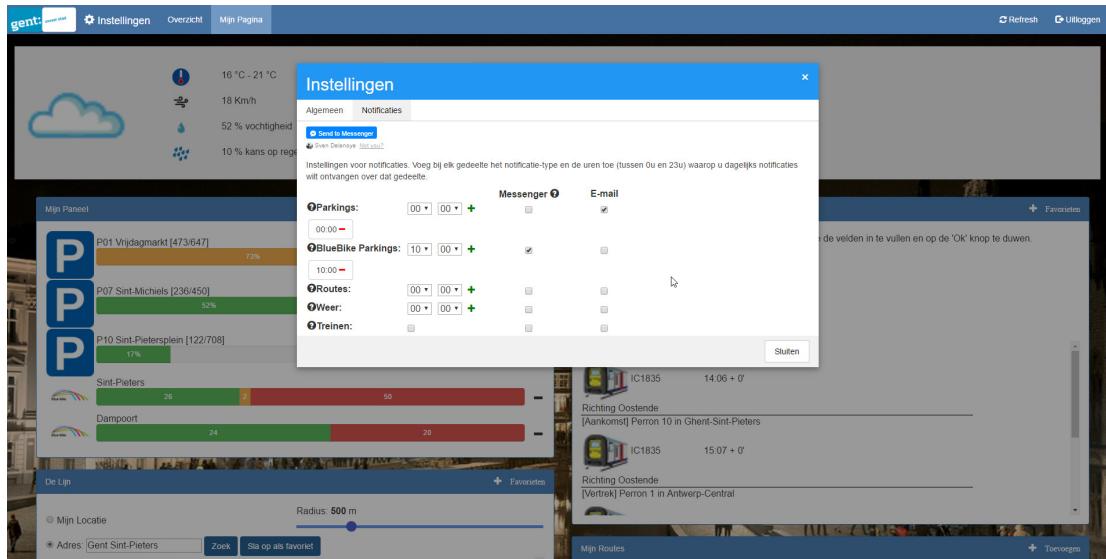


Fig. 4: Mock-up persoonlijke pagina: keuzemogelijkheden

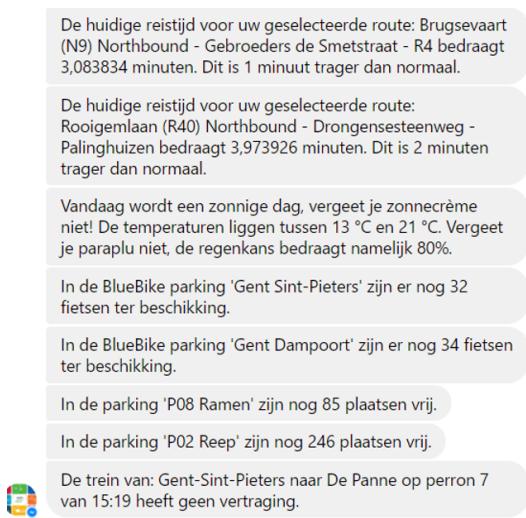


Fig. 5: Mock-up meldingen via Messenger

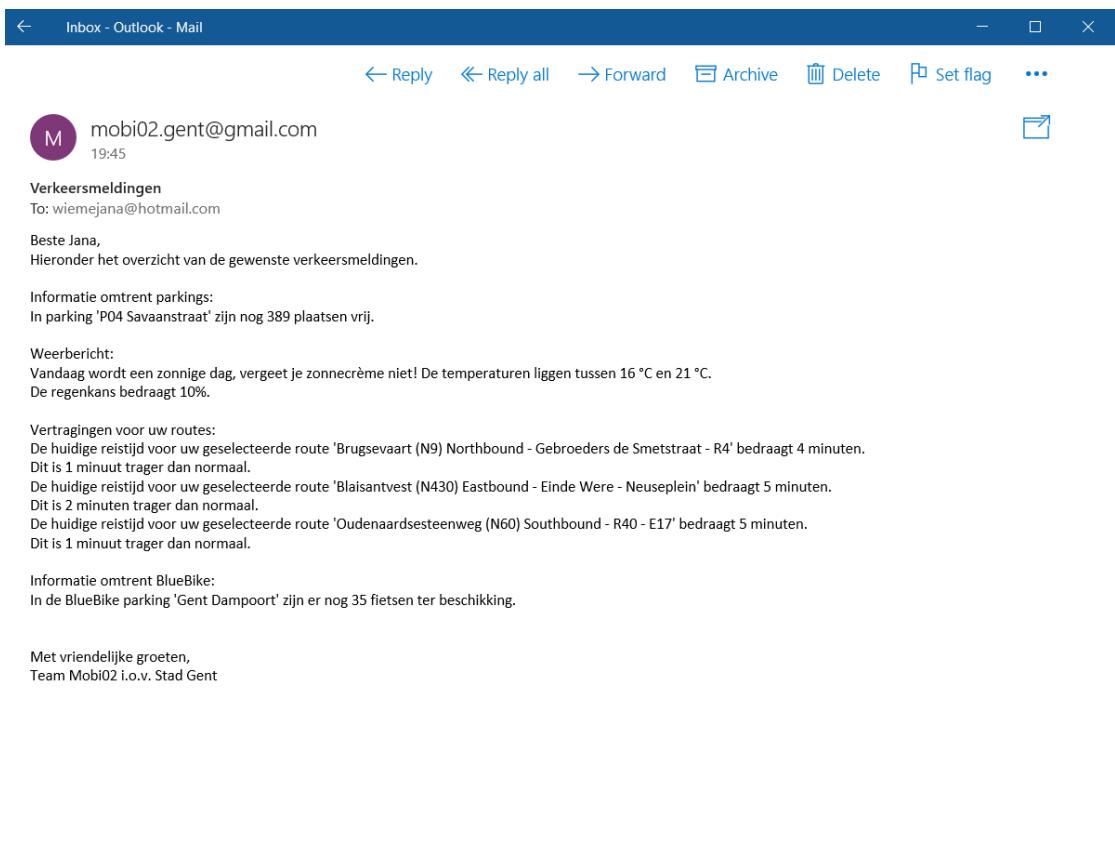


Fig. 6: Mock-up meldingen via mail

#### 4.3 Use case diagrammen

In dit onderdeel van het verslag worden de use case diagrammen weergegeven. Eerst en vooral wordt het use case diagram van sprint 1 herhaald (figuur 7), vervolgens het diagram van sprint 2 (figuur 8, en tot slot wordt dit gevolgd door het use case diagram op het einde van dit project, zie figuur 9. Dit geeft een beter overzicht van hoe de toepassing ondertussen evolueerde. Een belangrijke evolutie hierbij is bijvoorbeeld de vervanging van de websocket backend door een systeem met een REST Service.

Het use case diagram geeft een beschrijving van de functionaliteit van het te bouwen systeem, namelijk het mobiliteitsdashboard. Het heeft als doel een duidelijke omschrijving van de functionaliteit te voorzien, zodat de afspraken tussen klant en ontwikkelaar vastgelegd kunnen worden.

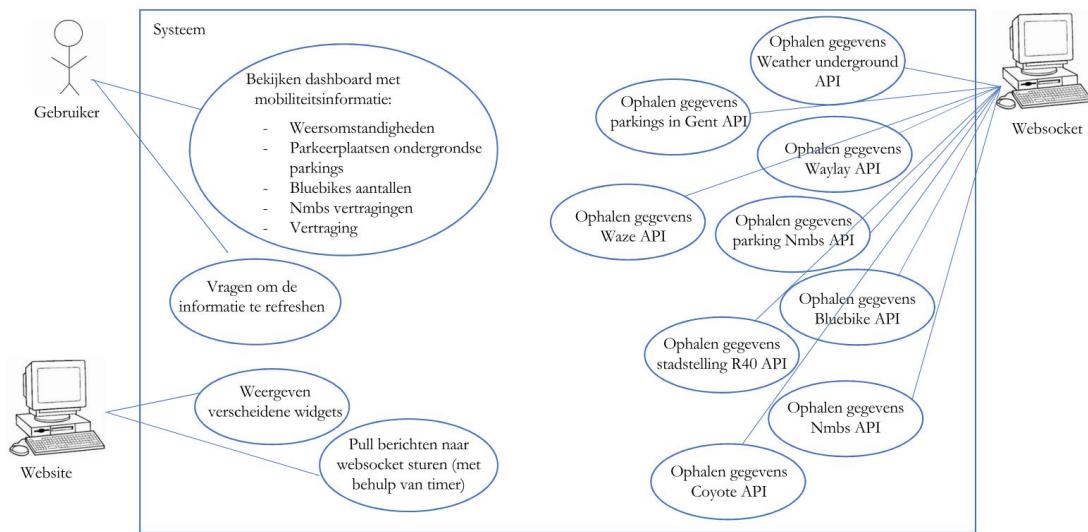


Fig. 7: Use case diagram sprint 1

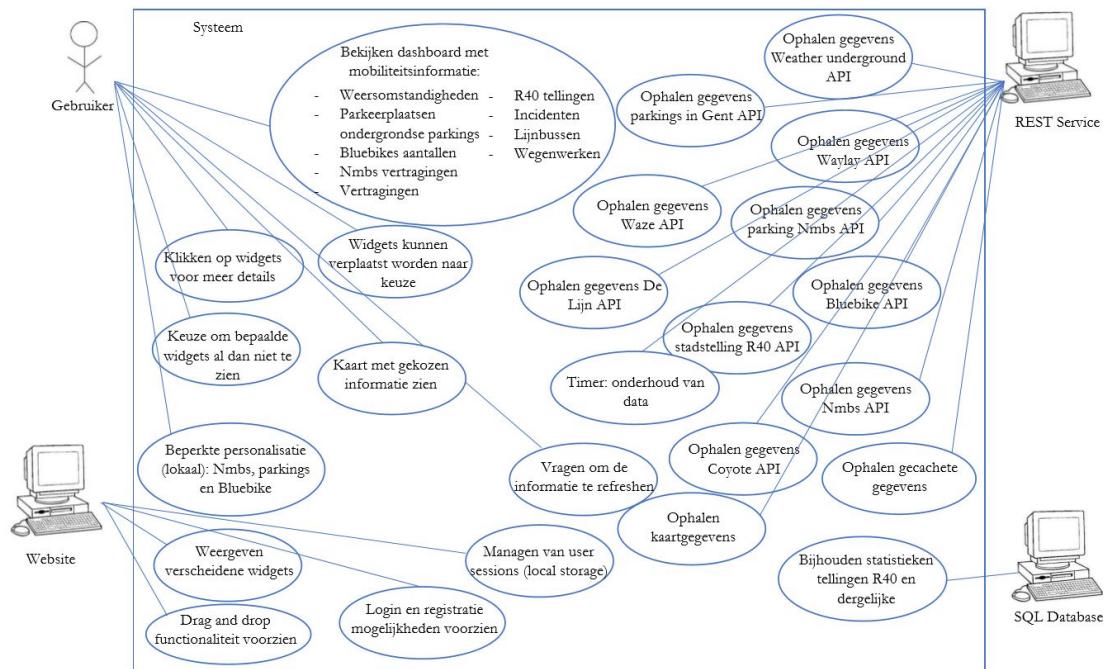


Fig. 8: Use case diagram sprint 2

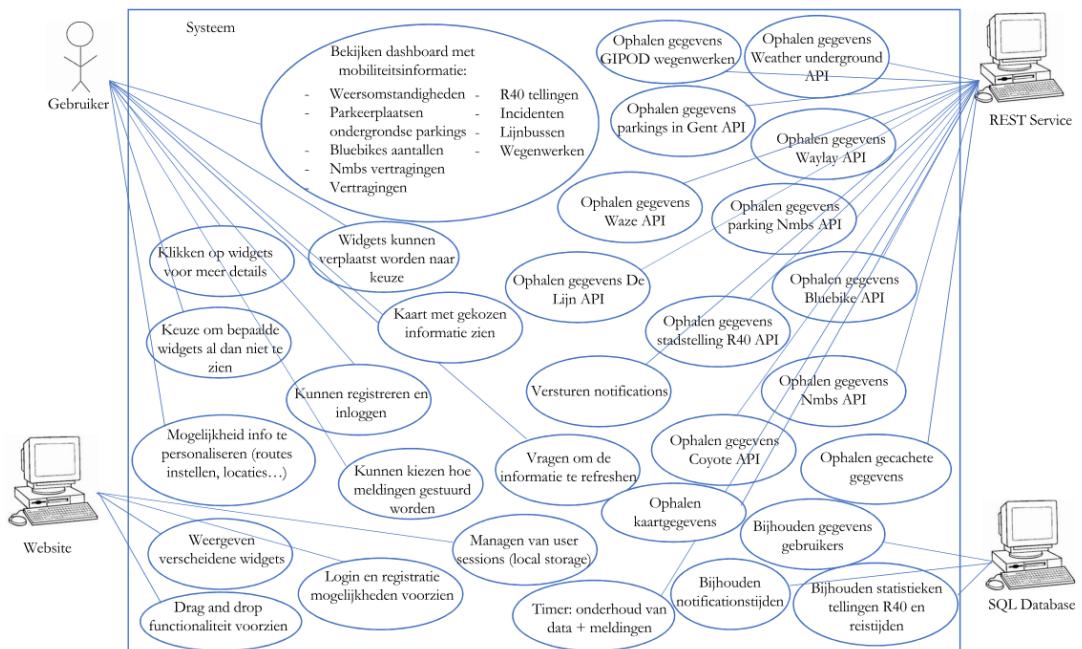


Fig. 9: Use case diagram sprint 3 (einde)

#### 4.4 Use cases tekstueel

De belangrijkste use cases uit bovenstaande diagrammen worden hieronder in een tekstuele vorm weergegeven. Bepaalde use cases werden reeds in sprint 1 uitgewerkt, andere pas in sprint 2 of 3. De onderstaande use cases zijn allen aangepast naar de stand van zaken op het einde van sprint 3, vermits soms enkele actoren gewijzigd werden in de loop van het project.

Use Case	De gebruiker bekijkt het dashboard
Primaire actor	Gebruiker
Secundaire actoren	REST Service, website
Preconditie	Dashboardpagina moet beschikbaar zijn, dynamische velden zijn nog niet opgevuld.
Postconditie	De dashboardpagina wordt weergegeven met correcte informatie.
Successscenario	<ol style="list-style-type: none"> <li>1. De gebruiker wenst het dashboard te bekijken.</li> <li>2. Het systeem toont de dashboardpagina met correct ingevulde informatie in de verschillende panels van het dashboard.</li> <li>3. De informatie wordt automatisch geüpdatet op verschillende tijdstippen, dit naargelang de variabiliteit van de data en mogelijkheden van de API.</li> </ol>
Alternatieve scenario's	<ol style="list-style-type: none"> <li>2.a. Het systeem slaagt er niet in om alle informatie uit de verschillende API's op te halen.</li> <li>2.b. Het systeem geeft een correcte melding.</li> <li>2.c. Het systeem toont de dashboardpagina zonder de ontbrekende API gegevens. Postconditie wordt bereikt.</li> <li>3.a. De gebruiker gaat zelf de refresh-button gebruiken om de gegevens opnieuw op te halen. Postconditie wordt bereikt.</li> </ol>

Fig. 10: Use case 1

### Stand van zaken einde sprint 1

Deze use case werd reeds in zijn geheel geïmplementeerd in sprint 1. Bij het alternatieve scenario 2a, wordt tot nu toe gekozen voor optie 2c, en zal bijgevolg het dashboard getoond worden zonder de onbereikbare gegevens.

Een uitbreiding die voorzien was in sprint 2, is dat de gegevens gecachet worden zodat niet telkens alle API's volledig opnieuw opgevraagd moeten worden, maar enkel diegene met nieuwe informatie.

### Stand van zaken einde sprint 2

Backend werden enkele wijzigingen gedaan in de structuur. De websocket werd namelijk vervangen door de REST service, aangevuld met beans om de data asynchroon op te halen en te refreshen.

Bovendien wordt, zoals hierboven vermeld werd, de data nu ook gecachet van zodra de server start. Hierdoor wordt de informatie sneller weergegeven en werd de asynchrone

manier van ophalen ondersteund.

### Stand van zaken einde sprint 3

Deze use case bleef ongewijzigd in sprint 3.

Use Case	Ophalen van gegevens uit verscheidene API's
Primaire actor	REST Service
Preconditie	De maximumtijd van de gecachete data is verstreken, er dient vernieuwing te gebeuren of er werd een refresh aangevraagd vanuit de webpagina (handmatig of bij inladen)
Postconditie	De gegevens van alle bronnen zijn geüpdatet.
Successscenario	<ol style="list-style-type: none"><li>1. De REST Service wenst de recentste gegevens op te halen uit verscheidene API's.</li><li>2. De REST Service haalt alle gevraagde gegevens op.</li><li>3. De REST Service verwerkt de verzamelde gegevens.</li><li>4. De REST Service schrijft de data weg naar de cache.</li><li>5. De gegevens van alle bronnen worden geüpdatet doorgegeven naar de frontend.</li></ol>
Alternatieve scenario's	
2.a.	De REST Service slaagt er niet in om alle informatie uit de verschillende API's op te halen.
3.a.	De REST Service geeft enkel de gekende informatie door, samen met een foutmelding van de niet werkende API's. De REST Service logt de foutmeldingen naar de database.
3.b.	Ga naar stap 4.

Fig. 11: Use case 2

### Stand van zaken einde sprint 1

Deze use case werd reeds in zijn geheel geïmplementeerd. De mogelijkheid tot het opvragen van slechts enkele API's werd reeds klaargezet, maar voorlopig wordt dit in het frontend nog niet ondersteund.

### Stand van zaken einde sprint 2

Ondertussen werd de backend hervormd, waardoor de informatie per API afzonderlijk opgehaald kan worden alsook gecachet wordt. Deze use case werkt bijgevolg lichtelijk gewijzigd, maar is opnieuw in zijn geheel geïmplementeerd.

### **Stand van zaken einde sprint 3**

Deze use case bleef ongewijzigd in sprint 3.

<b>Use Case</b>	<b>De gebruiker bekijkt details van widgets</b>
<b>Primaire actor</b>	Gebruiker
<b>Secundaire actoren</b>	REST Service, website
<b>Preconditie</b>	Dashboardpagina moet beschikbaar zijn.
<b>Postconditie</b>	Van de gevraagde widget wordt meer gedetailleerde informatie gegeven.
<b>Successscenario</b>	
1.	De gebruiker wenst een bepaalde widget van het dashboard in detail te bekijken.
2.	Het systeem toont een webpagina waarin de details behorende bij de widget worden weergegeven (bijvoorbeeld een kaartje, uitgebreidere uitleg).
<b>Alternatieve scenario's</b>	
2.a.	Het systeem slaagt er niet in om meer informatie weer te geven.
2.b.	Het systeem geeft een correcte melding.
2.c.	Terug naar stap 1.

Fig. 12: Use case 3

### **Stand van zaken einde sprint 1**

Deze use case werd nog niet opgenomen in het backlog van sprint 1 en zal bijgevolg pas in sprint 2 uitgewerkt worden.

### **Stand van zaken einde sprint 2**

In sprint 2 werd deze use case geïmplementeerd, en krijgt de gebruiker bijgevolg de mogelijkheid om sommige widgets groter weer te geven, of op de kaart te tonen.

### **Stand van zaken einde sprint 3**

In sprint 3 werden nog enkele andere widgets uitgebreid, bijvoorbeeld met meer keuzemogelijkheden zoals meerdere soorten grafieken, andere data, meer element op de kaart, andere eenheden etc.

<b>Use Case</b>	<b>De gebruiker kiest welke widgets hij waar wenst te zien en dit wordt onthouden.</b>
<b>Primaire actor</b>	Gebruiker
<b>Secundaire actoren</b>	REST Service, website
<b>Preconditie</b>	Dashboardpagina moet beschikbaar zijn.
<b>Postconditie</b>	Enkel de gewenste widgets worden weergegeven. Dit kan dynamisch gewijzigd worden.
<b>Successscenario</b>	<ol style="list-style-type: none"> <li>1. De gebruiker wenst slechts enkele widgets te zien met relevante informatie voor hem.</li> <li>2. Het systeem toont enkel de gevraagde widgets, en geeft de optie in een menu om andere widgets ook nog toe te voegen.</li> <li>3. De gebruiker kan de widgets slepen in de gewenste volgorde.</li> <li>4. Het systeem onthoudt welke widgets de gebruiker wenst te zien en gaat dus de user sessions correct managen.</li> </ol>
<b>Alternatieve scenario's</b>	<ol style="list-style-type: none"> <li>2.a. Het systeem slaagt er niet in om de informatie van bepaalde widgets weer te geven.</li> <li>2.b. Het systeem geeft een correcte melding.</li> <li>2.c. Het systeem toont enkel informatie in de widgets waar informatie ter beschikking is. Postconditie wordt niet bereikt.</li> <li>4.a. Het onthouden van de gewenste widgets gebeurde niet correct.</li> <li>4.b. Het systeem toont de volledige dashboard pagina.</li> <li>4.c. Terug naar stap 1.</li> </ol>

Fig. 13: Use case 4

### Stand van zaken einde sprint 1

Deze use case werd nog niet opgenomen in het backlog van sprint 1 en zal bijgevolg pas in sprint 2 uitgewerkt worden.

### Stand van zaken einde sprint 2

Tijdens sprint 2 werd deze use case uitgewerkt, en kan de gebruiker bijgevolg widgets al dan niet zichtbaar maken. Ook de gewenste drag-and-drop functionaliteit werd toegevoegd. Deze use cases werd bijgevolg volledig geïmplementeerd.

### Stand van zaken einde sprint 3

Deze use case bleef ongewijzigd in sprint 3.

Use Case	De gebruiker kan registreren.
Primaire actor	Gebruiker
Secundaire actoren	REST Service, website, SQL Database
Preconditie	Dashboardpagina en loginpagina moeten beschikbaar zijn.
Postconditie	De gebruiker heeft een account.
Successscenario	<ol style="list-style-type: none"><li>1. De gebruiker wenst voor een eerste keer te registreren.</li><li>2. Het systeem slaat de gegevens van de gebruiker op.</li><li>3. De gebruiker krijgt een mailtje dat de registratie gelukt is.</li><li>4. De gebruiker kan vanaf nu inloggen.</li></ol>
Alternatieve scenario's	<ol style="list-style-type: none"><li>2.a. Het systeem slaagt er niet in om de gebruikersgegevens op te slaan.</li><li>2.b. Het systeem heeft reeds een gebruiker met de ingevoerde gegevens.</li><li>2.c. Database is niet beschikbaar.</li><li>2.d. Het systeem toont een correcte melding. Ga naar stap 1.</li></ol>

Fig. 14: Use case 5

### Stand van zaken einde sprint 1

Deze use case werd nog niet opgenomen in het backlog van sprint 1 en zal bijgevolg pas in sprint 2 uitgewerkt worden.

### Stand van zaken einde sprint 2

Aan deze use case werd in sprint 2 gewerkt, maar er zal nog verdere optimalisatie nodig zijn in sprint 3. De registratiepagina en dergelijke staan allemaal reeds klaar, ook de databank werd geïnitialiseerd, maar het registratiesysteem zelf is nog niet in gebruik.

### Stand van zaken einde sprint 3

Deze use case werd in sprint 3 volledig geïmplementeerd, de gebruiker kan vanaf nu dan ook inloggen met behulp van een mailadres, krijgt vervolgens een bevestigingsmailtje, en kan dan inloggen en zijn persoonlijke pagina beheren.

<b>Use Case</b>	<b>De gebruiker kan inloggen.</b>
<b>Primaire actor</b>	Gebruiker
<b>Secundaire actoren</b>	REST Service, website, SQL Database
<b>Preconditie</b>	Dashboardpagina en loginpagina moeten beschikbaar zijn.
<b>Postconditie</b>	De gebruiker kan zijn persoonlijke pagina bekijken en zaken wijzigen.
<b>Successscenario</b>	<ol style="list-style-type: none"> <li>1. De gebruiker wenst in te loggen.</li> <li>2. Het systeem controleert de ingevoerde gegevens van de gebruiker met de informatie uit de database.</li> <li>3. De gebruiker is ingelogd en ziet zijn persoonlijke pagina.</li> <li>4. De gebruiker kan wijzigingen aanbrengen aan zijn persoonlijke pagina en deze bewaren.</li> </ol>
<b>Alternatieve scenario's</b>	<ol style="list-style-type: none"> <li>1.a. De gebruiker is reeds ingelogd.</li> <li>1.b. Het systeem toont een correcte melding. Ga naar stap 1.</li> <li>2.a. Het systeem slaagt er niet in om de gebruikersgegevens terug te vinden voor verificatie.</li> <li>2.b. Database is niet beschikbaar.</li> <li>2.c. Het systeem toont een correcte melding. Ga naar stap 1.</li> <li>3.a. De persoonlijke pagina kon niet geladen worden.</li> <li>3.b. Het systeem toont een correcte melding. Postconditie wordt niet bereikt.</li> </ol>

Fig. 15: Use case 6

### Stand van zaken einde sprint 1

Deze use case werd nog niet opgenomen in het backlog van sprint 1 en zal bijgevolg pas in sprint 2 uitgewerkt worden.

### Stand van zaken einde sprint 2

Aan deze use case werd in sprint 2 gewerkt, maar er zal nog verdere optimalisatie nodig zijn in sprint 3. De inlogservice werd klaargezet, maar dient nog verder geïmplementeerd te worden.

### Stand van zaken einde sprint 3

Deze use case werd in sprint 3 volledig uitgewerkt, de gebruiker kan inloggen via het mailadres dat hij in de registratie meegaf, of via zijn account op Facebook.

<b>Use Case</b>	<b>De gebruiker kan specifieke informatie instellen.</b>
<b>Primaire actor</b>	Gebruiker
<b>Secundaire actoren</b>	REST Service, website, SQL Database
<b>Preconditie</b>	Dashboardpagina en loginpagina moeten beschikbaar zijn, de gebruiker moet ingelogd zijn.
<b>Postconditie</b>	De gebruiker heeft gewenste informatie ingesteld en ziet gebeurtenissen op die route.
<b>Successscenario</b>	<ol style="list-style-type: none"> <li>1. De gebruiker wenst een bepaalde trein, parking, bus, BlueBike halte of een gegeven trajectgedeelte te zien op zijn pagina.</li> <li>2. Het systeem slaat de toegevoegde favorieten op in de database.</li> <li>3. De gebruiker ziet vanaf nu de gegevens specifiek van deze favorieten.</li> <li>4. De gebruiker kan wijzigingen aanbrengen aan zijn routes of locaties.</li> </ol>
<b>Alternatieve scenario's</b>	<ol style="list-style-type: none"> <li>2.a. Het systeem slaagt er niet in om de voorkeuren op te slaan.</li> <li>2.b. Database is niet beschikbaar.</li> <li>2.c. Het systeem toont een correcte melding. Ga naar stap 1.</li> <li>3.a. De persoonlijke pagina kon niet geladen worden.</li> <li>3.b. De favorieten worden niet getoond.</li> <li>3.c. Er is geen informatie beschikbaar voor deze favoriet.</li> <li>3.d. Het systeem toont een correcte melding. Postconditie wordt niet bereikt.</li> </ol>

Fig. 16: Use case 7

### Stand van zaken einde sprint 1

Deze use case werd nog niet opgenomen in het backlog van sprint 1 en zal bijgevolg pas in sprint 2 of 3 uitgewerkt worden.

### Stand van zaken einde sprint 2

Aan deze use case werd in sprint 2 gewerkt, maar er zal nog verdere optimalisatie nodig zijn in sprint 3. Er werd reeds een persoonlijke pagina voorzien, waarbij de gebruiker wel reeds enkele instellingen kan kiezen. Hierbij kan hij kiezen om bepaalde parkings, informatie over BlueBike, alsook bepaalde treinen te zien. Deze mogelijkheden worden in sprint 3 nog uitgebreid.

### **Stand van zaken einde sprint 3**

In sprint 3 werd deze use case uitgebreid, waarbij de gebruiker bijgevolg verschillende zaken kan toevoegen aan zijn persoonlijk dashboard. Deze elementen zijn: bezettingsgraad van de parkings, bezettingsgraad BlueBike, De Lijn bussen (te zoeken op adres of huidige locatie), Nmbs routeplanner, bepaalde trajecten in Gent met bijhorende real time reistijd.

<b>Use Case</b>	<b>De gebruiker kan kiezen op welke manieren hij meldingen krijgt en krijgt meldingen op het correcte moment.</b>
<b>Primaire actor</b>	Gebruiker
<b>Secundaire actoren</b>	REST Service, website, SQL Database
<b>Preconditie</b>	Dashboardpagina en loginpagina moeten beschikbaar zijn, de gebruiker moet ingelogd zijn. De gebruiker heeft reeds routes of locaties ingesteld.
<b>Postconditie</b>	De gebruiker krijgt meldingen op de gewenste manieren over de gebeurtenissen op zijn routes en locaties.
<b>Successscenario</b>	<ol style="list-style-type: none"><li>1. De gebruiker wenst meldingen te krijgen over gebeurtenissen op zijn routes/locaties.</li><li>2. Het systeem slaat op op welke manieren meldingen gestuurd moeten worden.</li><li>3. Het systeem stuurt op de correcte momenten de meldingen.</li><li>4. De gebruiker krijgt meldingen.</li></ol>
<b>Alternatieve scenario's</b>	<ol style="list-style-type: none"><li>2.a. Het systeem slaagt er niet in om de manieren op te slaan.</li><li>2.b. Database is niet beschikbaar.</li><li>2.c. Het systeem toont een correcte melding. Ga naar stap 1.</li><li>3.a. De meldingen kunnen niet verstuurd worden. Postconditie wordt niet bereikt.</li></ol>

Fig. 17: Use case 8

### **Stand van zaken einde sprint 1**

Deze use case werd nog niet opgenomen in het backlog van sprint 1 en zal bijgevolg pas in sprint 3 uitgewerkt worden.

### **Stand van zaken einde sprint 2**

Deze use case werd nog niet opgenomen in het backlog van sprint 2 en zal bijgevolg pas in sprint 3 uitgewerkt worden.

### **Stand van zaken einde sprint 3**

In sprint 3 werd deze use case geïmplementeerd. Er wordt hierbij de mogelijkheid gegeven om de meldingen ofwel via mail, ofwel via Messenger berichten te sturen. In dit laatste geval moet de gebruiker uiteraard zijn Facebook account hieraan gekoppeld hebben. Deze meldingen werden pas op het einde van het project toegevoegd, en daar zou bijgevolg in de toekomst nog naar gekeken moeten worden om het systeem flexibeler en correcter te maken.

## 4.5 Backlog

In onderstaande topic worden zowel het backlog van sprint 1 als 2 alsook het masterbacklog getoond. Het masterbacklog komt overeen met het totale backlog, inclusief sprint 3. Hierin komen bijgevolg alle features aan bod die in het project voorzien zijn en eventueel al uitgewerkt zijn.

### 4.5.1 Sprint 1

Tabel 1: Backlog sprint 1

#	Naam	Story	Prioriteit	Schatting	Toegewezen
15	Documentatie backend	Opstellen van analyse-documenten over de werking van de backend	1	5	Jana & Ruben
1	Test programma	Maken van een simpel test programma die via een simpele webpagina de resulterende json van een API weergeeft	2	3	Ruben
8	Documentatie frontend	Opstellen van analyse-documenten over de werking van de frontend	3	4	Sven & Arnout
4	Ringverkeer API	Onderzoeken van de verkeer-op-de-R40 API en geldige, nuttige data kunnen opvragen	4	1	Sven
9	iRail API	Onderzoeken van de iRail API en geldige, nuttige data kunnen opvragen	5	1	Sven

5	Coyote API	Onderzoeken van de Coyote API en geldige, nuttige data kunnen opvragen	6	3	Jana
11	Parking Nmbs API	Onderzoeken van de parkeergarages API van de Nmbs en geldige, nuttige data kunnen opvragen	7	1	Jana
10	Waylay API	Onderzoeken van de Waylay API en geldige, nuttige data kunnen opvragen	8	2	Jana
13	Waze API	Onderzoeken van de Waze API en geldige, nuttige data kunnen opvragen	9	2	Jana
12	BlueBike API	Onderzoeken van de BlueBike API en geldige, nuttige data kunnen opvragen	10	1	Sven
2	Weer Api	Onderzoeken van de weer API en geldige, nuttige data kunnen opvragen	11	1	Ruben
3	Parkeergarages API	Onderzoeken van de parkeergarages API en geldige, nuttige data kunnen opvragen	12	1	Sven
6	Site Template	Het aanmaken van een template waar later de data in weergegeven zal worden	13	3	Sven & Arnout
14	Mockup Site	Het invullen van de data in de template	14	3	Sven & Arnout

#### 4.5.2 Sprint 2

Tabel 3: Backlog sprint 2

#	Naam	Story	Prioriteit	Schatting	Toegewezen
29	Documentatie backend	Opstellen van analyse-documenten over de werking van de backend	15	15	Jana & Ruben
30	Documentatie frontend	Opstellen van analyse-documenten over de werking van de frontend	16	4	Sven & Arnout
24	Caching & sessions	Voorzien van caching van gegevens om te frequent inladen van API's te vermijden	17	7	Ruben
32	Database	Aanmaken database en instellen Glassfish connection pool	18	4	Ruben
40	Registratie foutmeldingen	Het opslaan van de foutmeldingen in de database	19	5	Ruben
7	De Lijn API	Opvraging informatie De Lijn	20	10	Jana
20	Statistiek tov historiek	Voorzien van grafiek met R40 tov gemiddelde	21	5	Sven & Ruben
18	Kaart	Voorzien van het op de kaart weergeven van informatie	22	7	Sven
22	Aanpassingen dashboard	Betere lay-out	23	5	Arnout & Sven
28	Bevraging	Navraag (enquête) om te zien wat gebruikers wensen te zien	24	2	Jana
25	Drag and drop	Voorzien drag and drop systeem	25	2	Sven
27	Aanpassingen mobiele applicaties	Voorzien van css voor mobiele apparaten	26	3	Sven & Arnout
37	Login	Voorzien van popup voor login	27	3	Arnout
35	Taalontaf- hankelijkheid	Voorzien vertalingen (beschikbaarheid Nederlands, Engels, Frans)	28	2	Jana
44	Unit tests	Getters unit tests voorzien	29	3	Ruben

23	Persoonlijke pagina	Voorzien persoonlijke pagina met voorkeursmogelijkheden	30	10	Sven & Arnout
38	Registratie	Voorzien van popup voor registratie	31	3	Arnout

#### 4.5.3 Sprint 3

Tabel 5: Backlog sprint 3

#	Naam	Story	Prioriteit	Schatting	Toegewezen
45	Documentatie backend	Opstellen van analyse-documenten	32	10	Ruben & Jana
46	Documentatie frontend	Opstellen van analyse-documenten	33	5	Sven & Arnout
54	Lay-out	Wijzigingen lay-out naar aanleiding van gesprek met klant	34	5	Sven
53	Reistijden opvragen	Reistijden opvragen via Coyote	35	5	Jana
55	Totaal aantal wagens	Voorzien van call voor totaal aantal R40 wagens	36	2	Ruben
41	Databank	Voorzien van data voor databank vanuit frontend	37	8	Sven & Arnout
17	Analyse gebruiker - routers	Kijken naar bestaande applicatie voor routes stad Gent	38	7	Jana
21	Widget foutmeldingen	Weergeven van de foutmeldingen in de frontend	39	3	Sven
47	Optimalisatie persoonlijke pagina	Persoonlijke header bij persoonlijke pagina	40	15	Arnout & Sven
48	Databank	Zorgen voor correcte data-opslag van gebruikersgegevens	41	6	Ruben

49	Loginopties	Voorzien van de mogelijkheid voor Facebook of registratie login	42	5	Arnout & Sven
51	Instellingen persoonlijke pagina	Gebruikersinstellingen op persoonlijke pagina voorzien	43	6	Arnout & Sven
52	Instellingen persoonlijke routes	Frontend voorzien persoonlijke routes Coyote	44	5	Arnout
34	Bugfixes	De bugs uit het project zoveel mogelijk verwijderen	45	5	Ruben & Jana
58	Error Handling	Betere errorhandling frontend in de widgets	46	5	Sven & Arnout
59	De Lijn frontend call	Server niet belasten met De Lijn opvragingen, verplaatsing naar frontend	47	3	Jana
60	Https server	Server https werken voor bepaalde functionaliteiten	48	2	Ruben
50	Notifications	Mogelijkheid tot gebruik van notifications bij belangrijke meldingen via Messenger (Facebook)	49	8	Jana & Ruben
57	Mailing notifications	Voorzien mogelijkheid tot notifications per mail	50	5	Jana
61	Lay-out	Algemene wijzigingen	51	5	Sven & Arnout
63	Custom styles	Mogelijke stijlen laten kiezen op webpagina	52	3	Sven
56	Herorganisatie lijsten	Grafischere weergave van sommige data	53	3	Sven
66	Wegenwerken GIPOD	Meerdere wegenwerken ophalen via GIPOD API	54	4	Jana

## 5 Hoofdstuk 3: Systeemarchitectuur

In dit hoofdstuk worden verschillende analysesdocumenten toegelicht, om een beter inzicht te krijgen in de systeemarchitectuur van het project. Daarenboven wordt in bijlage 2 de analyse van de verscheidene API calls toegevoegd. Vermits elke API een eigen structuur heeft, wordt in deze bijlage toegelicht welke request gedaan moet worden om de specifiek gewenste API data te bekomen, en wat vervolgens het antwoord is. Deze bijlage vergemakkelijkt bijgevolg verdere ontwikkelaars die met één of meer van deze specifieke API's wenst te werken.

### 5.1 High-level systeem model

In figuur 18 wordt even kort de structuur geschetst van de applicatie. Dit is een sterk vereenvoudigde weergave, maar heeft als doel om reeds bij aanvang van dit hoofdstuk een globaal beeld te kunnen vormen van de structuur. Deze structuur komt verderop in het deployment diagram uitgebreider aan bod.

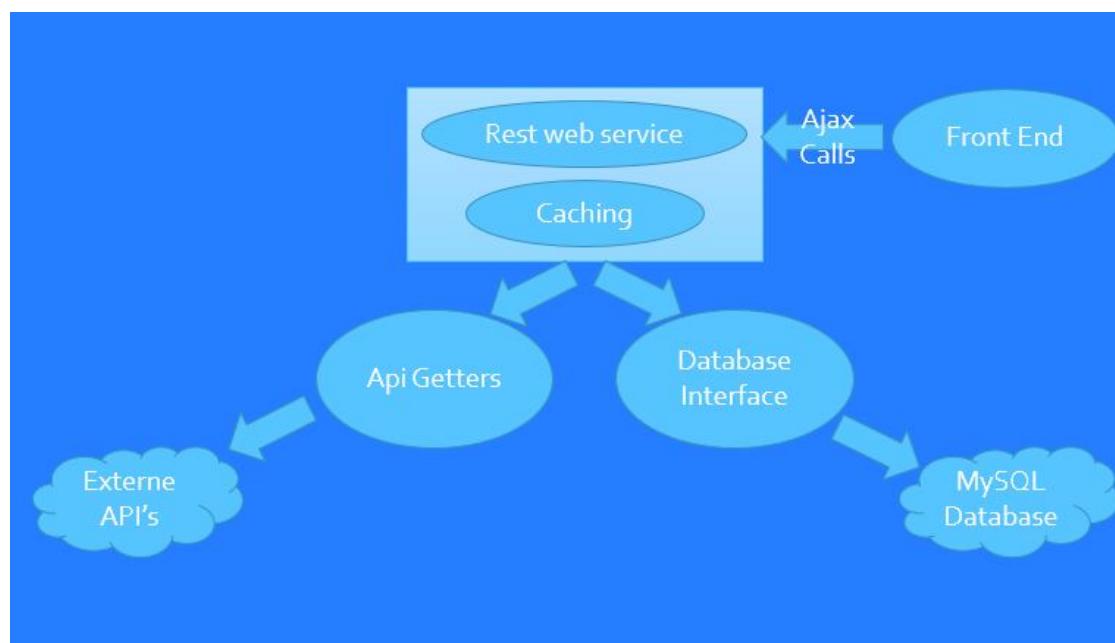


Fig. 18: Structuur applicatie

Het deployment diagram geeft weer welke technologien in dit project gebruikt worden. Zoals in figuur 19 getoond wordt, zijn er twee belangrijke devices, de webserver enerzijds en de database server anderzijds. De applicatie is een Java-webapplicatie, waarbij

uitvoering gebruik gemaakt wordt van een Glassfish-webcontainer zoals weergegeven wordt in de execution environment.

Daarnaast is er uiteraard ook een front- en backend project nodig om iets te kunnen weergeven. De frontend wordt volledig opgebouwd met html, css en javascript, welke compatibel zijn met de webserver. De backend waarmee geïnterageerd wordt, wordt met behulp van EJB beans en dus Java geschreven.

De database is een relationele database die werkt met de MySQL-standaard. De koppeling tussen de database en webserver gebeurd met behulp van JPA en JDBC, vermits er gekozen werd om een Java-webcontainer te gebruiken en dit de samenwerking tussen beide devices vereenvoudigt.

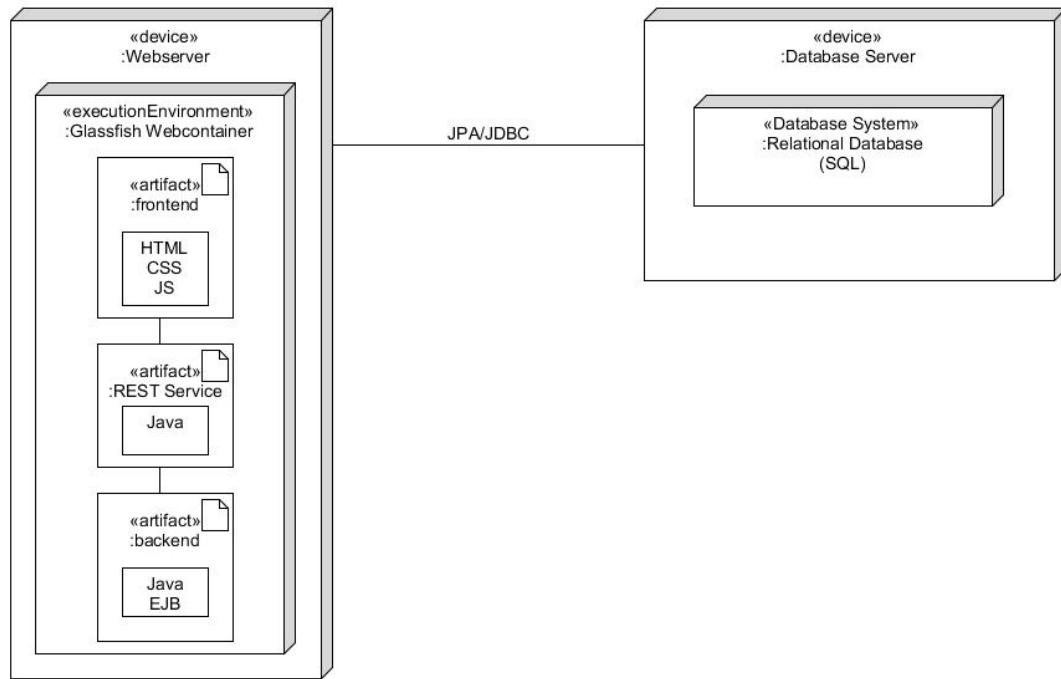


Fig. 19: Deployment diagram

In de hieropvolgende subsecties komen verscheidene diagrammen aan bod. Elk deel van de code wordt in minstens één soort diagram weergegeven. De manier waarop de code gedocumenteerd wordt, is afhankelijk van de functionaliteit en de geschiktheid van voorstellen met een bepaald diagram.

Met behulp van een overzichtsdiagram in figuur 20, gebaseerd op het packagediagram, wordt voor de backend aangegeven welke manier van documentatie voor welk deel ge-

bruikt worden. Deze diagrammen worden verderop in dit hoofdstuk toegelicht, en zijn omwille van hun omvang meestal ook toegevoegd in bijlage. Verder uitleg bij de diagrammen wordt in de subsecties gegeven. Deze figuur dient louter ter overzicht van de documentatie van de backend.

Legende bij figuur 20:

- Rode kader (module getters): klassendiagram (bijlage 3)
- Paarse kader (package getters module getters): klassendiagram (bijlage 4)
- Lichtblauwe kader (package model in module getters): klassendiagram (bijlage 5)
- Oranje kader (package database in module database): database diagram
- Zwarte kader (package Caching in module back): sequentiediagram (bijlage 8)
- Gele kader (package Statistics in module back): sequentiediagram (bijlage 9)
- Groene kader (package Rest in module back): oplijsting mogelijke REST calls die mogelijk zijn (bijlage 10)
- Bruine kader (package Notifications in module back): sequentiediagram (bijlage 11)

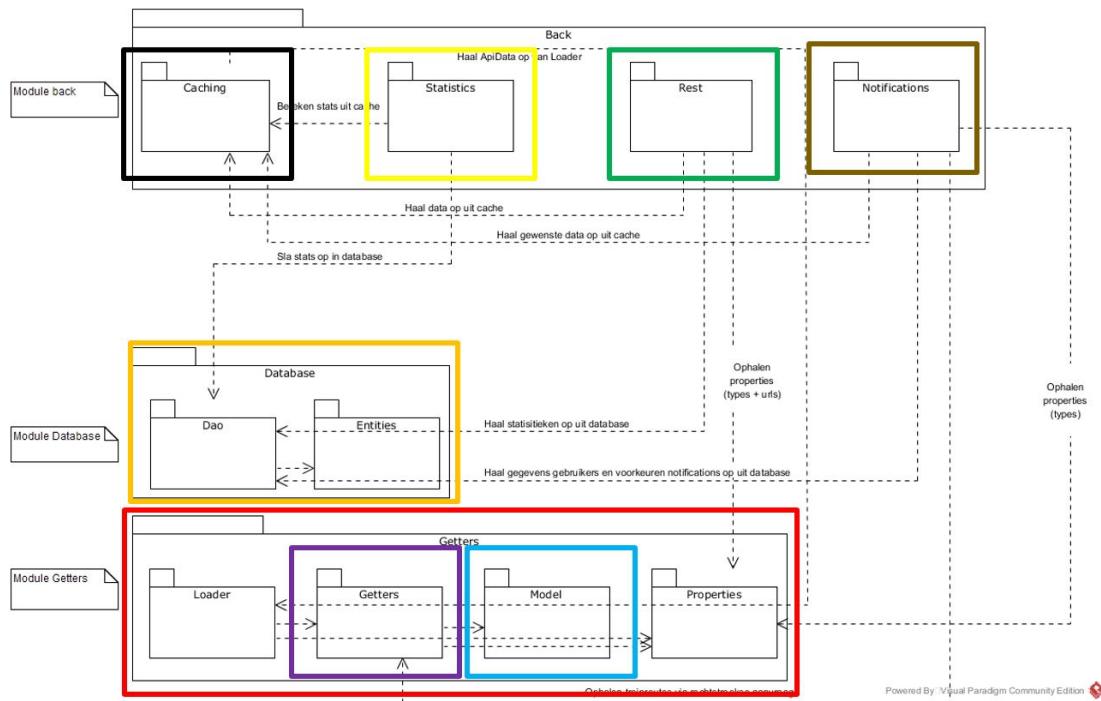


Fig. 20: Overzichtsdiagram

## 5.2 Klassendiagrammen

In dit onderdeel worden de noodzakelijke klassendiagrammen weergegeven, die bijgevolg deel uit maken van het statisch model. Vermits het volledige klassendiagram te uitgebreid is, wordt in het verslag zelf enkel het packagediagram opgenomen. De effectieve uitwerking van het gewenste deel van dit package diagram, zijn zoals vermeld terug te vinden in bijlagen 3 tot en met 5.

### 5.2.1 Package diagram backend

In het package diagram, figuur 21 wordt duidelijk een opsplitsing gemaakt in drie onderdelen, de modules back, database en getters. Binnen de module back is nog een drievoudige opsplitsing gemaakt, die verderop nog aan bod komt. In bijlage 3 wordt enkel een uitgewerkte versie van de deze getters module gegeven, vermits deze de belangrijkste statische informatie bevat.

De nauwe samenwerking die tussen de verscheidene modules, alsook hun packages, wordt in de figuur geïllustreerd met stippelpijlen. De module back wordt opgesplitst in 4 grote delen, caching, statistics, REST en notifications. De caching package draagt de verantwoordelijkheid voor het cachen van gegevens die via de module getters opgehaald worden. Deze gegevens worden op regelmatige tijdstippen bijgewerkt, maar het cachen is van uiterst belang om niet continu API calls te moeten maken.

De statistics package is verantwoordelijk voor het mogelijk maken van vergelijkingen tussen gemiddelde en real time waarden. Hiervoor interageert het eerst en vooral met de caching package om de gecachte data op te halen, maar ook met de database om de statistieken in op te slaan.

De REST package wordt vanuit de frontend opgeroepen via de REST calls die daar gemaakt worden. Van zodra zo'n REST call gemaakt wordt, zal de afhandeling van zo'n call de informatie ophalen uit de caching package. Omdat heel wat REST calls mogelijk gemaakt zijn, wordt in bijlage 10 een overzicht gegeven van alle gecreëerde REST calls en hoe deze te gebruiken. Daarnaast haalt de REST package ook informatie uit de database, met name de informatie die door de statistics klassen werd opgeslagen in de database.

De getters module werd, zoals vermeld, uitgewerkt toegevoegd in bijlage 3. Zoals ook in figuur 21 zichtbaar is, is er ook in deze module een verdere opsplitsing in packages. Zoals eerder aangegeven gaat de back module via de getters module de informatie ophalen van de verscheidene API's. De getters module heeft dan ook als functionaliteit het uitvoeren van de API calls op een gestructureerde manier, deze informatie te verwerken en in objecten te steken, en deze objecten vervolgens als json door te geven zodat ze frontend gebruikt kunnen worden.

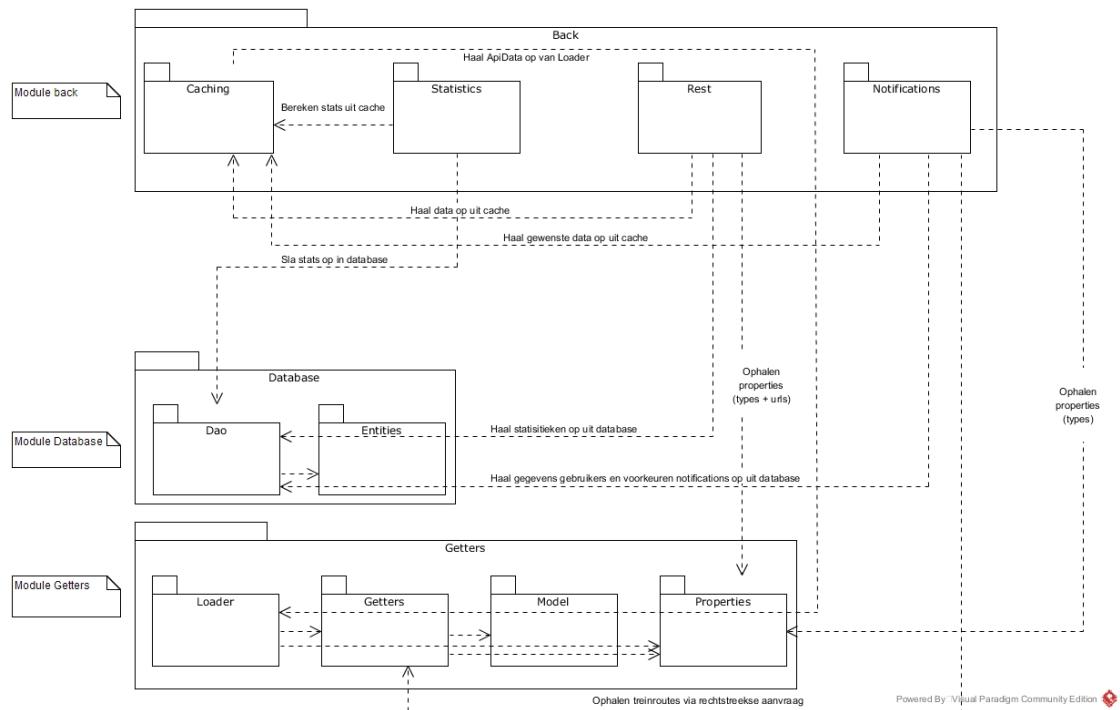


Fig. 21: Package diagram

### Module getters

Een eerste subpackage betreft de getters, allemaal klassen die een bepaalde API voorstellen. Al deze getters zijn stateless EJB beans die, met behulp van de gegevens vanuit de properties package, hun specifieke API call zullen uitvoeren. Vervolgens maken ze hun eigen type 'modelobject' aan op basis van de data die ze uit de call haalden, wat dus objecten uit de model package zijn.

Om de informatie uit deze propertiesbestanden te halen, wordt gebruik gemaakt van een singleton EJB bean, namelijk PropertyLoader. Deze klasse bevat verscheidene methoden die informatie inleest uit de juiste propertiesbestanden, afhankelijk van wat er gevraagd wordt. De keuze om dit af te splitsen in een afzonderlijke singleton EJB bean werd gemaakt omdat er anders ofwel heel wat duplicate code in elke klasse zou komen, ofwel heel wat informatie hardgecodeerd in de klassen zou staan. Nu worden de propertiesbestanden maar eenmaal ingelezen, en wordt via deze bean op een eenvoudige manier de data opgehaald. Bovendien is het niet de bedoeling dat de keys voor bepaalde API's publiek ter beschikking gesteld zouden worden, en bijgevolg zijn deze afgezon-derd in een apart propertiesbestand. De PropertyLoader bean wordt in elke getterklasse geïnjecteerd. De getters worden nog iets gedetailleerder weergegeven in bijlage 4, zoals de paarse kader op de figuur ook aangeeft.

Getters gaan dus hun informatie opslaan in modelobject. Deze objecten zijn alle-

maal afgeleid van de interface IApiModel en bevatten de data die op de webpagina weergegeven dient te worden. In bijlage 5 wordt de uitwerking van deze modelklassen geïllustreerd. Typisch aan deze modelklassen is dat ze heel wat private variabelen hebben, alsook de daarbijhorende get- en setmethodes. Bovendien hebben ze een getAsJson() methode, die ervoor zorgt dat het modelobject teruggegeven wordt als jsonstring.

Om de getters te 'regelen', wordt een ApiLoader package voorzien. Deze package bevat de ApiLoader bean, een klasse die op basis van een ApiType parameter de juiste getterklasse zal oproepen. Elke getterklasse wordt dan ook gekenmerkt door een bepaald type, namelijk een ApiType. Alle types worden verzameld in de ApiType enumeration, wat ervoor zorgt dat de ApiLoader op een consequente en efficiënte manier telkens de juiste getterklasse kan aanspreken. De ApiLoader klasse en de getters zijn beiden Enterprise Java Beans en worden ook door de webcontainer geïnjecteerd.

Tot slot is er ook een Http bean, die reeds start voor alle andere beans, en ervoor zorgt dat de httpclient geconfigureerd wordt. De reden dat deze bean in het project werd opgenomen, is omdat er bepaalde certificaten genegeerd dienen te worden om httpsaanvragen te kunnen voltooien. Op dit moment is dit echter enkel nog in gebruik voor aanvragen naar de Coyote API.

### 5.2.2 Frontend

De frontend bestaat zoals vermeld uit een klassieke website, in dit geval bestaande uit twee effectieve webpagina's. De hoofdpagina heeft, zoals in de mock-up fig:mainpg getoond werd, als doel om alle informatie overzichtelijk weer te geven. Dit is de basisfunctionaliteit van dit project, beschreven in use case 1. In bijlage 6 is een overzicht terug te vinden van alle javascript-bestanden die nodig zijn om deze pagina haar functionaliteit te geven. Centraal staat ApiLoader.js, omdat deze de hoofdzaken bevat voor de opbouw van de webpagina. Op de figuur in bijlage staan ook heel wat opmerkingen, die aangeven welk deel van de functionaliteit een bepaald javascript-bestand vervult.

De tweede webpagina werd reeds getoond in figuur 3, en staat in voor een centraal concept in deze bachelorproef, namelijk personalisatie. In bijlage 7 is van deze webpagina een overzicht te vinden van de verschillende javascript-bestanden en hun methodes.

Om deze webpagina's op te bouwen werd, zoals het deployment diagram reeds aangaf, in hoofdzaak gebruik gemaakt van javascript, css en html. Daarbovenop werd er gebruik gemaakt van verschillende externe open source frameworks. Zo werd er onder meer Jquery<sup>1</sup> gebruikt om de site dynamisch op te bouwen.

Om de structuur van de hoofdpagina op te bouwen werd gebruik gemaakt van Packery<sup>2</sup>.

---

<sup>1</sup>JQuery - write less, do more. Geraadpleegd op 2 maart 2017 via <https://jquery.com/>

<sup>2</sup>Packery - Gapless, draggable grid layouts. Geraadpleegd op 30 maart 2017 via <https://packery.metafizzy.co/>

Dit zorgt ervoor dat de widgets automatisch herschikt worden naargelang de grootte van de verschillende widgets, zodat ze allen een optimale positie op de site hebben. Packery maakt ook de drag-and-drop functie mogelijk. Ook Jquery UI<sup>3</sup> was hiervoor nodig.

De structuur binnen de widgets werd vaak opgebouwd met Bootstrap<sup>4</sup>, en analoog werd ook 'Mijn Pagina' hiermee opgebouwd. De R40-grafiek op de hoofdpagina werd dan weer opgebouwd met Chart.js<sup>5</sup>, omdat hiermee enkele specifieke functionaliteiten voor de bouw van een grafiek mogelijk gemaakt werden.

Tot slot werden er nog enkele kleinere frameworks toegevoegd om bepaalde aspecten van de site mooier en praktischer te maken. Zo werden de toggle-buttons opgebouwd met Bootstrap Toggle<sup>6</sup> en zijn de select-menu's aangemaakt met Bootstrap Select<sup>7</sup>.

Om een zeker inzicht te krijgen in deze frontend structuur, wordt in bijlage 5 het klassendiagram toegevoegd.

In dit klassendiagram wordt enerzijds de ApiLoader.js getoond, omdat deze javascript-file centraal staat voor de opbouw van de webpagina. In de bijlage staan meerdere opmerkingen die de structuur verder verduidelijken.

### 5.3 Sequentiediagrammen

Om een beter inzicht te krijgen in het verloop, de werking en bijgevolg ook de communicatie binnen het systeem, wordt in onderstaande figuur 24 het sequentiediagram gegeven van de globale werking van het project. Indien een gebruiker naar het dashboard navigeert, zal de html-pagina het 'onReady'-event oproepen, waardoor een REST call uitgevoerd zal worden voor het ophalen van bepaalde data. Deze REST call gebeurt voor elke API, zoals ook in de opmerking van figuur 24 vermeld wordt.

Backend wordt de REST call vervolgens in de klasse ApidataRest verwerkt, en zal er mer behulp van de PropertyLoaderBean gekeken worden van welk type API data teruggegeven dient te worden. Vervolgens gaat de ApidataRest klasse de gecachte data ophalen behorende bij de gewenste API. Eenmaal die data opgehaald is, geeft de REST service deze door naar de frontend, waar de gebruiker vervolgens de informatie te zien zal krijgen.

---

<sup>3</sup>JQuery user interface. Geraadpleegd op 30 maart 2017 via <https://jqueryui.com/>

<sup>4</sup>Bootstrap - Designed for everyone, everywhere. Geraadpleegd op 28 februari 2017 via <http://getbootstrap.com/>

<sup>5</sup>Chart.js - Simple yet flexible JavaScript charting. Geraadpleegd op 19 april 2017 via <http://www.chartjs.org>

<sup>6</sup>Bootstrap Toggle is a highly flexible Bootstrap plugin that converts checkboxes into toggles. Geraadpleegd op 21 april via <http://www.bootstraptoggle.com/>

<sup>7</sup>Bootstrap-select is a jQuery plugin that utilizes Bootstrap's dropdown.js to style and bring additional functionality to standard select elements. Geraadpleegd op 28 april via <https://silviomoreto.github.io/bootstrap-select/>

Indien ergens in het traject iets misging, bijvoorbeeld een verkeerde ApiType of lege cachedata, dan zal er een exceptie opgeworpen worden, die meegegeven wordt aan de frontend. Op de webpagina zal het vervolgens duidelijk getoond worden dat er op zo'n moment geen data beschikbaar is.

Na verloop van tijd raakt data uiteraard verouderd, en zal deze geüpdatet moeten worden. Elke API heeft echter zijn eigen specifieke kenmerken van hoe statisch hun data is. Zo zullen wegenwerken niet zo vaak veranderen als bijvoorbeeld files op bepaalde trajecten. Bijgevolg is nood aan een andere timeoutwaarde. Van zodra de timeout van een bepaalde API verstreken is, zal het cachingmechanisme in gang schieten, zoals weergegeven in figuur 22. Deze figuur is ook terug te vinden in bijlage 8 in een groter formaat. In dit diagram wordt bijgevolg de package Caching uit de module back geïllustreerd.

In dit cachingmechanisme triggert een timer de updatemethode indien nodig. In deze updatemethode zal de CachingManger aan de ApiLoader klasse de gewenste informatie opvragen. Deze ApiLoader bean gaat vervolgens een lijst teruggeven met de gevraagde informatie in objecten. Deze objecten worden dan opgeslagen, en zo is de gecachte data opnieuw up to date.

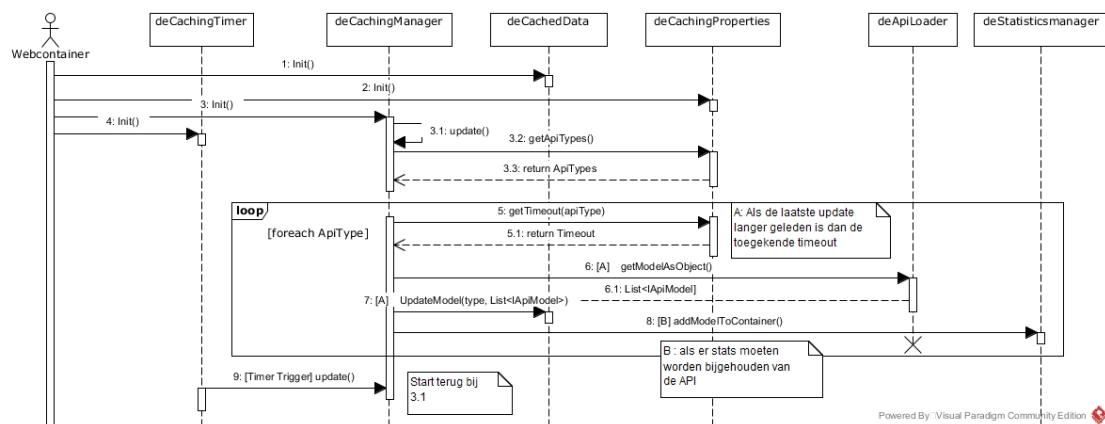


Fig. 22: Sequentiediagram cachingmechanisme

De manier waarop de ApiLoader bean te werk gaat, is bij elke API hetzelfde. Omdat het ophalen van de informatie van De Lijn echter een stuk complexer is dan de andere API's, wordt het specifieke sequentiediagram hiervan opgenomen in figuur 23. Vermits het sequentiediagram vrij uitgebreid is, wordt een grotere versie weergegeven in bijlage 12. Hoe de ApiLoader bean hierbij te werk gaat, wordt specifiek voor De Lijn uitgelegd. Alle andere API's worden op een analoge, maar eenvoudigere manier opgehaald.

Gezien het mechanisme van De Lijn een trage ophaalsnelheid heeft, wordt de data van De Lijn niet automatisch ingeladen. De gebruiker krijgt echter de mogelijkheid om ofwel via een adres, ofwel via de persoonlijke locatie bussen op te vragen. Eens de gebruiker dit

mechanisme triggert, zullen de coördinaten opgevraagd worden via Google Maps (ofwel de locatie zelf ofwel omzetting van adres naar coördinaten).

Vervolgens wordt de controle doorgegeven aan de ApiLoader frontend, die met de gegeven coördinaten een REST call gaat uitvoeren. Hierbij wordt de controle opnieuw doorgegeven, ditmaal naar de backend. De ApidataRest klasse gaat vervolgens deze specifieke lijn REST call uitwerken door rechtstreeks, zonder tussenkomst van de caching manager, het datamodel op te vragen bij de ApiLoader bean. De Lijn data cachen heeft weinig zin, vermits deze zeer variabel is en een lange tijd nodig heeft om in te laden.

De ApiLoader bean is verantwoordelijk voor het oproepen van de juiste getterklassen, hier dus de getter LijnApiGetter. Bij de meeste API's, dient maar één type data via een call opgevraagd te worden. Bij De Lijn is dit echter een stapsgewijs proces. Hierbij moeten namelijk eerst de dichtstbijzijnde haltes opgevraagd worden, om vervolgens de eerstvolgende bussen aan deze haltes op te halen. Tijdens sprint 2 gebeurde dat ophalen van die bussen ook via een call in de backend, waarbij de LijnApiGetter dus minstens twee calls deed alvorens het resultaat te kunnen teruggeven.

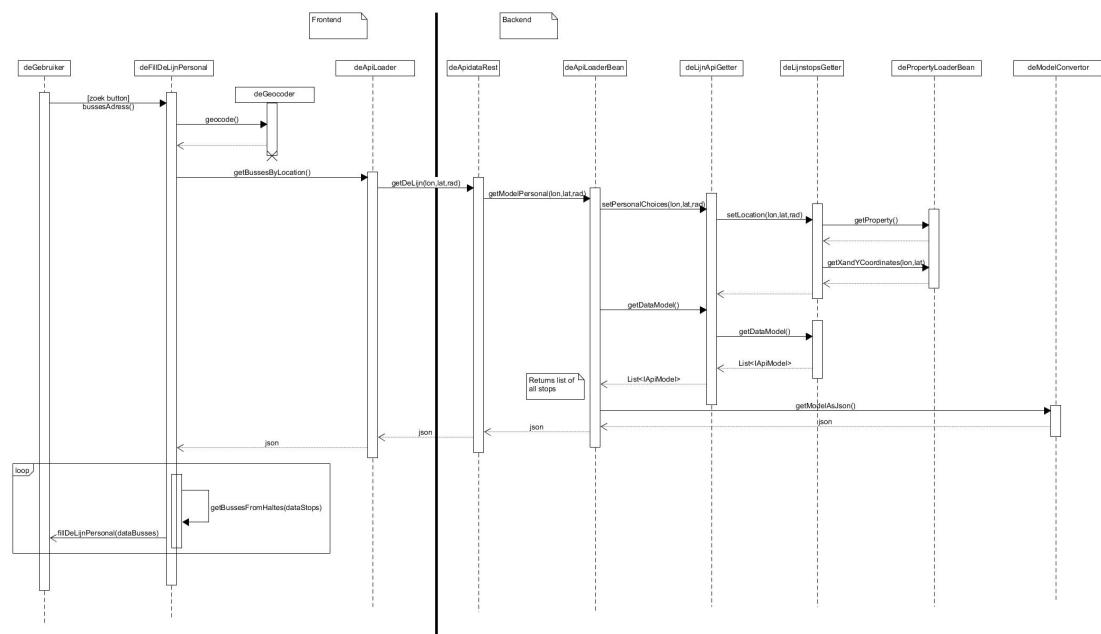


Fig. 23: Sequentiediagram De Lijn

Echter duurt het sowieso al relatief lang om bussen op te halen, en kwam hierbij nog dat voor soms twee of drie haltes in de buurt sychroon een call gedaan moest worden. Kortom, het duurde veel te lang. Daarom werd in sprint 3 beslist om de opvraging van de bussen niet meer backend, maar wel frontend door te voeren, zoals ook in figuur 23 getoond wordt. Hierbij gebeurt de opvraging met ajax calls, die asynchroon uitgevoerd

kunnen worden, en bijgevolg sneller tot hetzelfde resultaat leiden.

Om de dichtstbijzijnde halte op te vragen, worden eerst de coördinaten via de PropertyLoaderBean geconverteerd naar het Lambert72 coördinatenstelsel vermits De Lijn dit vereist. Eens de coördinaten omgezet zijn, zal de ApiLoader bean het datamodel opvragen, welke een lijst van haltes in de buurt teruggeeft.

Tot slot zal de ApiLoader bean het model laten omzetten naar een json, dit met behulp van de ModelConvertor. Deze json wordt vervolgens volledig terug in het traject doorgegeven, en op basis hiervan zal de gebruiker de data te zien krijgen. Bij De Lijn volgt dus hierna pas de frontend ajax call om de bussen op basis van de gekregen haltes te kunnen weergeven.

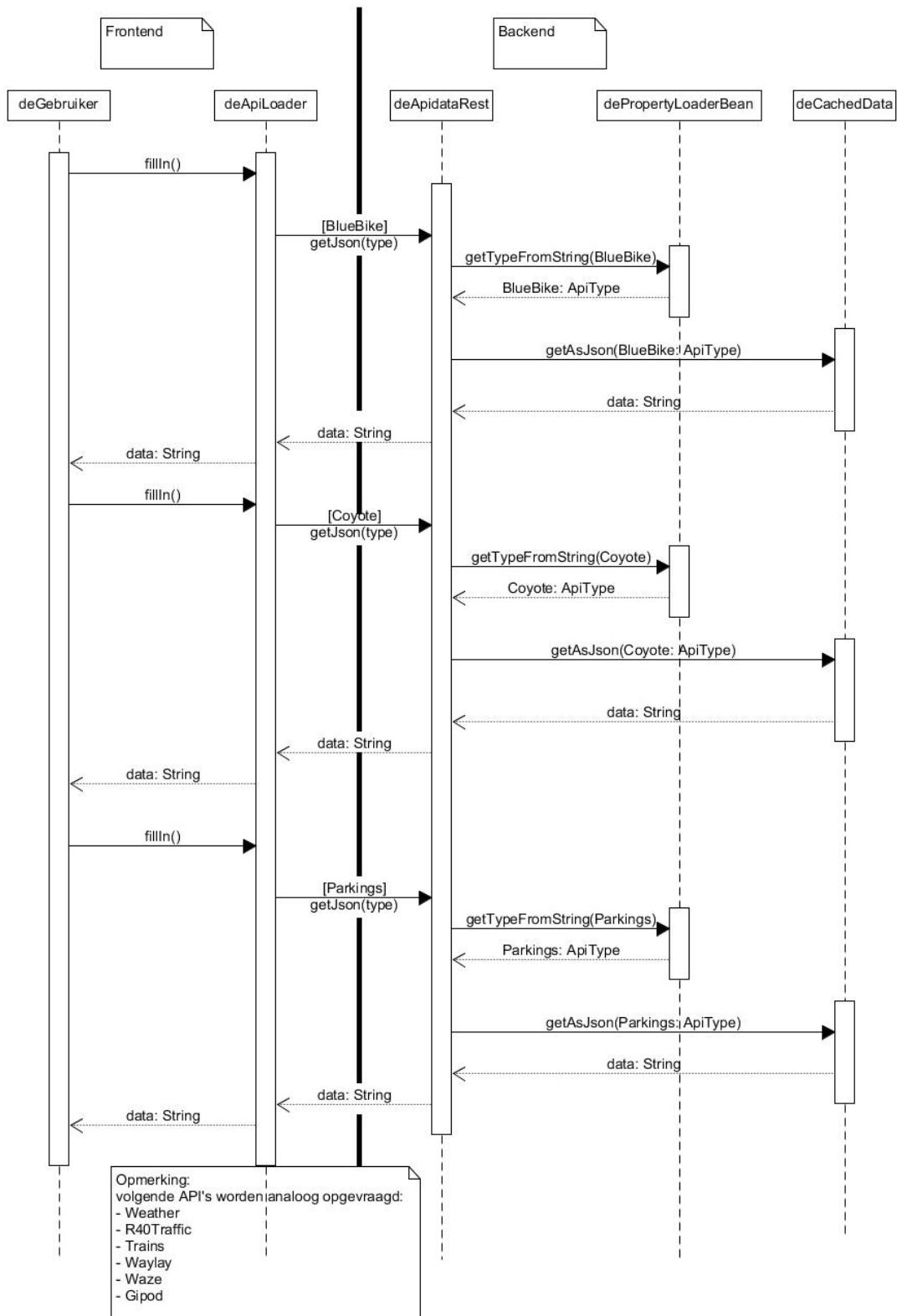


Fig. 24: Sequentiediagram globaal <sup>42</sup>

Een andere package die zich ook in de module back bevindt, is de notification package. Deze package werd in sprint 3 toegevoegd, als uitbreiding op de bestaande functionaliteiten, en maakt mogelijk dat een gebruiker op gewenste tijdstippen, de gewenste data krijgt. Dit is dus de implementatie van use case 8.

Omdat zo'n notificationsysteem heel wat acties met zich meebrengt, werd gekozen om dit met behulp van een sequentiediagram te documenteren. Het volledige sequentiediagram is terug te vinden in bijlage 11, maar vermits dit een zeer groot diagram is, wordt het hieronder in drie delen weergegeven, respectievelijk in figuren 25, 26 en 27.

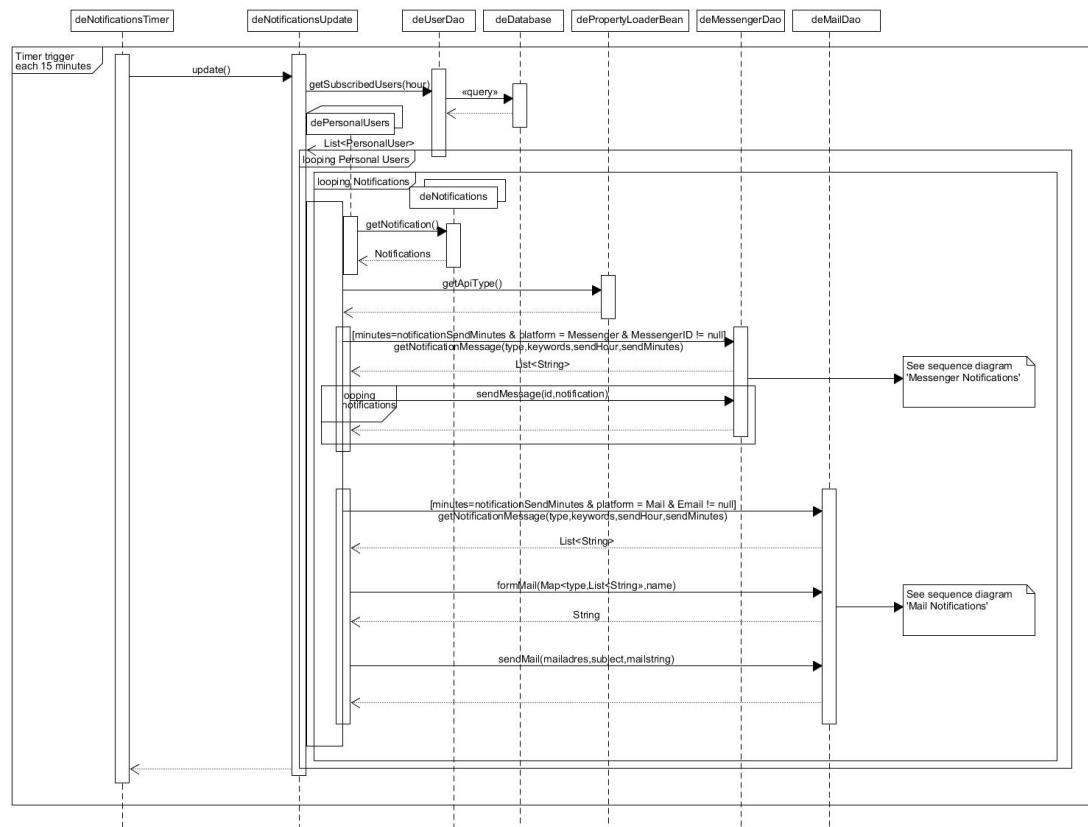


Fig. 25: Sequentiediagram notifications (vereenvoudigde versie)

De basis van het gehele systeem van notifications, zit in een timerklasse NotificationTimer, die elk kwartier gaat kijken of er nieuwe meldingen gestuurd moeten worden. Op dit moment is het dus enkel mogelijk om elk kwartier meldingen te krijgen, maar wanneer dit gewenst is kan hier een ander interval gekozen worden. De meldingen gebeuren bijgevolg niet volgens het push-systeem, waarbij anders, telkens wanneer de data wijzigt, een melding gestuurd zou moeten worden. De reden hiervoor is dat de data vaak heel variabel blijkt te zijn, en dat bovendien Facebook-Messenger niet toestaat dat er

pushberichten gestuurd worden op die manier.

Wanneer de timer afloopt, zal de updatemethode van de klasse NotificationsUpdate opgeroepen worden. In deze methode wordt eerst en vooral contact opgenomen met de database, om een lijst te bekomen van alle gebruikers die zich ingeschreven hebben op notificaties. Eens deze lijst in de updateklasse aanwezig is, worden al deze gebruikers (PersonalUsers) systematisch overlopen. Per gebruiker wordt vervolgens nog een lus doorlopen, namelijk welke notifications ze allemaal gevraagd hadden. Deze notifications worden vervolgens allen bekeken om te kijken of ze op dat moment wel gestuurd moeten worden.

Indien dat het geval is, zijn er twee mogelijkheden. Ofwel koos de gebruiker voor Messenger als platform om meldingen te ontvangen, en bijgevolg dient de gebruiker gekoppeld te zijn met Messenger (zie ook uitleg hieromtrent verder in het verslag). Ofwel koos de gebruiker om meldingen per mail te krijgen, en zal hij een geldig mailadres opgegeven moeten hebben.

Afhankelijk van het gekozen platform, worden vervolgens de meldingen eerst opgebouwd om tot slot ook de meldingen te sturen. In figuur 26 wordt het geval van Messenger geïllustreerd. De opbouw van de lijst met meldingen is volledig analoog voor het mailingplatform, zoals in figuur 27 gezien kan worden. Hieronder wordt geschreven hoe algemeen te werk gegaan wordt.

Eerst wordt in de afhandelende dao gekeken voor welk type API er informatie gevraagd wordt. Op basis daarvan gaat een specifieke methode van de klasse MessagesData opgeroepen worden. In deze methode wordt alle data behorende bij die API uit de cache gehaald, en wordt vervolgens uit deze gecachte data de gewenste data geselecteerd.

Met deze data-objecten, wordt dan een soort van formatterklasse aangeroepen. In zo'n klasse wordt voor elk data-object uit de lijst van dat type de correcte melding gegenereerd. De opbouw van deze meldingen is uiteraard verschillend voor elke soort data. De formatterklasse geeft tot slot een lijst terug van alle meldingen van dat type in String formaat.

Een uitzondering op dit systeem zijn de meldingen voor de treinen. Vermits de gebruiker de mogelijkheid krijgt om over alle treinen meldingen te krijgen, en bijgevolg niet enkel over de treinen die op de hoofdpagina getoond worden, kan de data niet uit de cache gehaald worden. Op de hoofdpagina wordt zoveel mogelijk relevante informatie voor een breed publiek weergegeven, en dat betreft bijgevolg alleen de treinen van en naar Gent-Sint-Pieters en Gent-Dampoort. Hierdoor zal er vanuit de MessagesData klasse eerst een aanvraag gedaan worden voor het datamodel van de gekozen trein(en), waarbij een API call uitgevoerd wordt, alvorens de melding opgebouwd kan worden.

Een verschil tussen beide platformen is wel de manier waarop vervolgens de meldingen gestuurd worden. Bij Messenger gaat gewoon voor elke notification een afzonderlijk berichtje gestuurd worden. Bij de mailserver daarentegen zal eerst en vooral een volledige

mail opgebouwd worden, waarin dan ook alle meldingen opgenomen zijn. Pas na het vormen van een gestructureerde mail, zal de mail ook effectief verzonden worden. De reden hiervoor is dat Messengerberichten typisch gebruikt worden voor korte en snelle communicatie, terwijl heel veel korte mailtjes na elkaar dan weer niet zo optimaal zijn. Kortom het opbouwen van mails en messengernotifications loopt dus volledig gelijk, maar het ontvangen bericht wordt aangepast aan het communicatiemedium.

Indien van een bepaalde API geen data beschikbaar is, zal die melding gewoon niet gestuurd worden, of zal dat gedeelte in de mail weggelaten worden. Een mail zal bovendien ook niet verzonden worden indien geen enkele API data doorgeeft. Op dat moment zullen er dan ook geen Messenger berichten en geen mails verzonden worden.

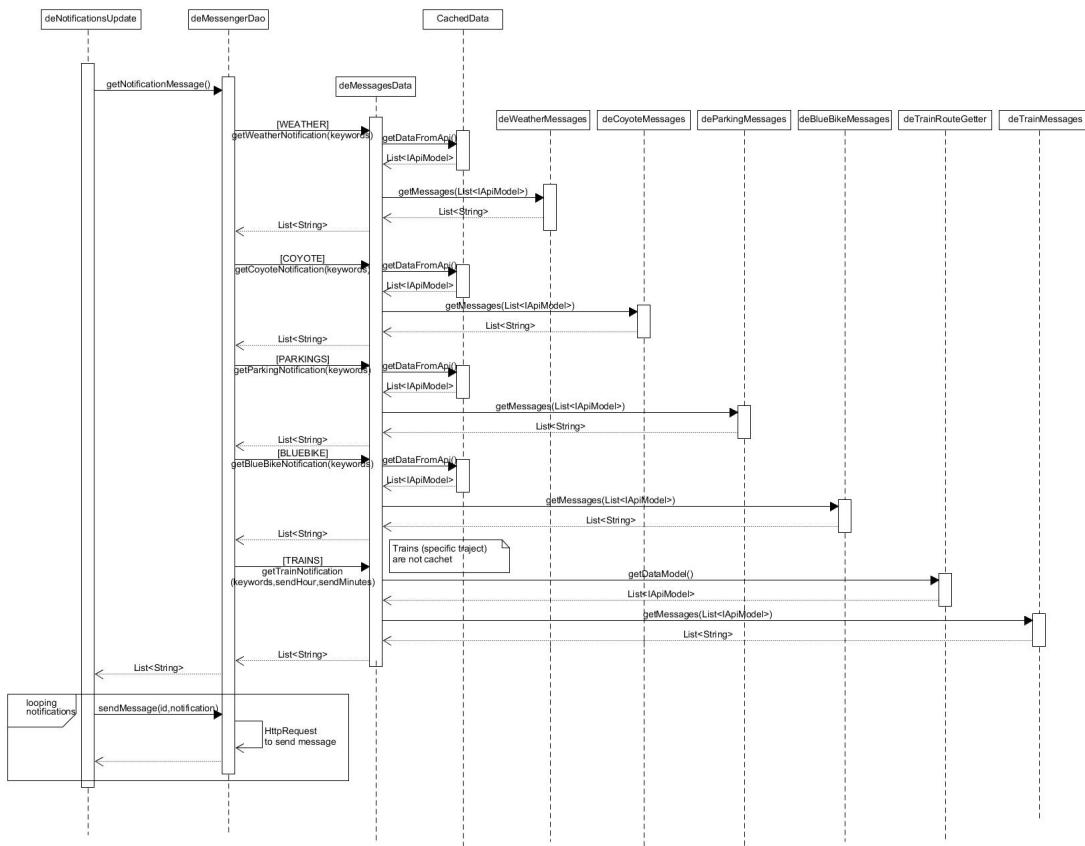


Fig. 26: Sequencediagramm gedeelte Messenger notifications

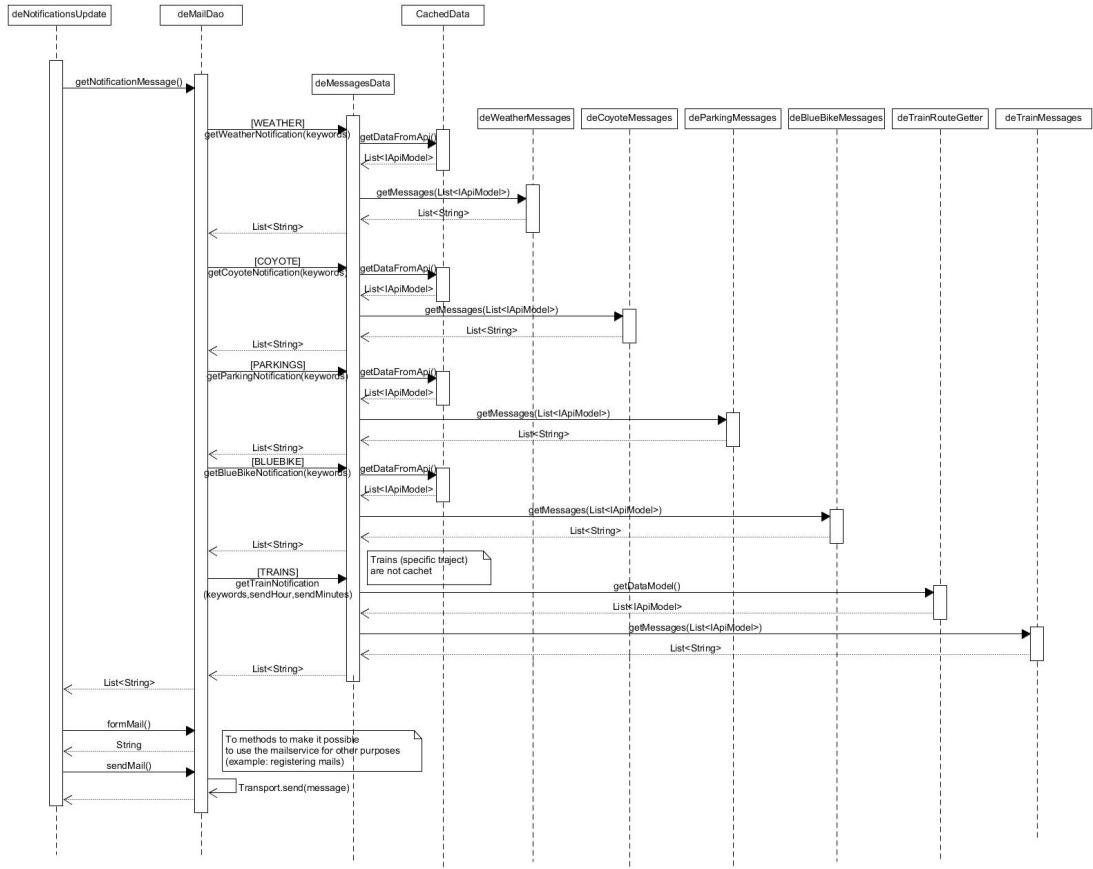


Fig. 27: Sequentiediagram gedeelte mail notifications

## 5.4 Activiteitendiagram

Vermits in de widget met de informatie omrent De Lijn met behulp van enkele frontend opties enkele stappen doorlopen moeten worden alvorens een API call uitgevoerd wordt, wordt in figuur 29 een activiteitendiagram getoond. Dit activiteitendiagram geeft een gedetailleerde beschrijving van de activiteiten die binnen deze widget worden uitgevoerd om een inzicht te geven in het verwerkingsproces.

Op het beginpunt wordt gewoon de widget getoond, zonder dat er iets wordt geladen, zie ook figuur 28. De gebruiker krijgt echter radiobuttons te zien met een keuze tussen een 'adres' of 'mijn locatie', alsook een schuifbaar om een straal in te stellen.

Eens de gebruiker een keuze maakt om iets aan te klikken, zal afhankelijk van datgene wat hij koos een reeks activiteiten gestart worden, met allen als eindactie de triggering van een APIcall. In het geval dat ergens iets misloopt, zoals bijvoorbeeld een ongeldig

adres, zal een exceptie opgeworpen worden en wordt dit ook in de widget weergegeven.

Zoals in het diagram zichtbaar is, krijgt de gebruiker, na selectie van de adres radiobutton, de keuze om een adres in te typen of het standaardadres te nemen. Na het intypen kan er op enter gedrukt worden, of op de zoekbutton zelf geklikt worden.

Bij het verschuiven van de slider zal, eens de gebruiker de slider loslaat, gecontroleerd worden welk van beide radiobuttons aangevinkt staat, en vervolgens zo een API call triggeren.

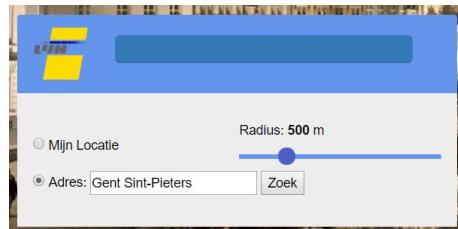


Fig. 28: De Lijn widget bij aanvang

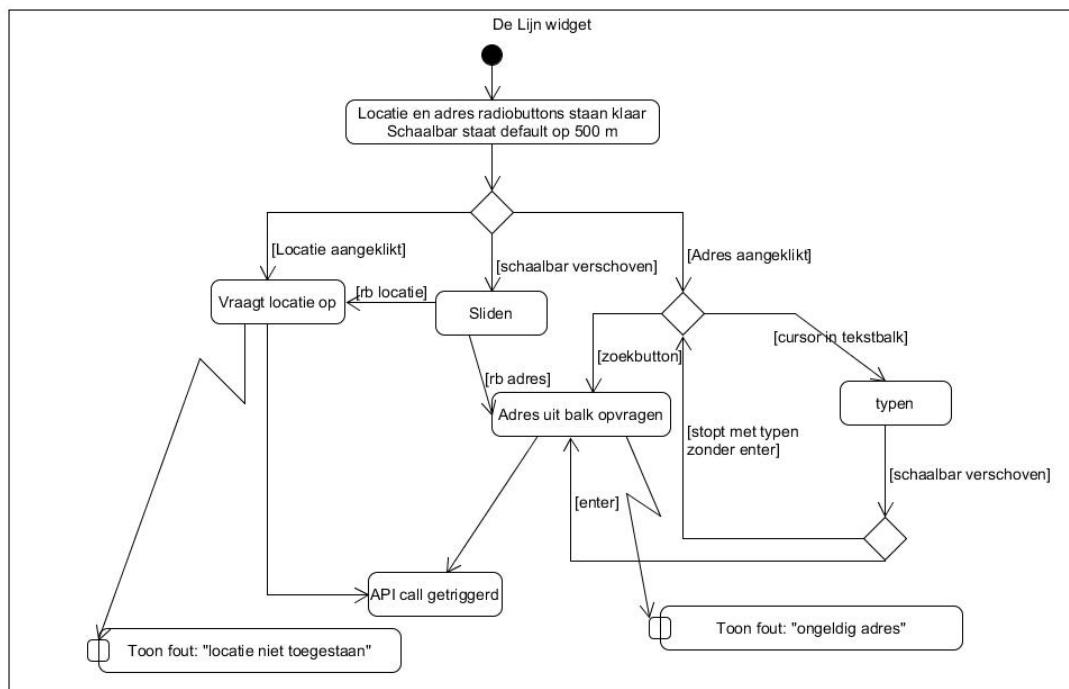


Fig. 29: Activiteitendiagram De Lijn widget

## 5.5 Databank diagram

In dit project kan de databank eigenlijk in twee grote delen gesplitst worden. Enerzijds is er een gedeelte noodzakelijk voor het opslaan van informatie van de gebruikers, hun voorkeuren, hun widgets, hun gebruikersgegevens etc. Anderzijds is er een gedeelte met enkele tabellen die eerder informatief zijn, en niets te maken hebben met gebruikers en hun persoonlijke keuzes.

In figuur 30 worden links de vijf tabellen behorende bij gebruikersinformatie getoond. Ook de verbanden tussen de tabellen staan in dit diagram aangegeven. Elke gebruiker zit in de tabel 'PersonalUser', en afhankelijk van of hij al dan niet met Facebook inlogde, zal hij eventueel ook in de FacebookUserInfo tabel opgeslagen zitten. Een andere tabel die aan de gebruiker gekoppeld is, betreft de 'Widgets'. Hierin wordt opgeslagen welke widgets een gebruiker al dan niet wenst te zien op zijn persoonlijke dashboard.

Zoals reeds in paragraaf 5.3 vermeld werd, zal men in de database moeten bijhouden welke notifications een gebruiker wenst te ontvangen. Deze worden opgeslagen in de tabel 'NotificationPreferences', met bijhorende type, platform, keywords (indien bijvoorbeeld meerdere parkings gewenst zijn) en het tijdstip.

Daarentegen zijn er ook de 4 losstaande, informatieve tabellen met verschillende doeleinden. Een korte beschrijving van de functionaliteiten hiervan is:

- ApiLog: slaat de logging op, zodat gekeken kan worden waarom en wanneer er eventuele errors plaatsvonden.
- NmbsStops: slaat alle stations van de nmbs op, dit omwille van performantiedelen zodat niet elke keer opnieuw alle stations opgevraagd moeten worden.
- R40Statistics: bevat data van de verschillende magnetische tellussen op de R40.
- Traveltimes: bevat de reistijden in beide richtingen om de ganse R40 rond te rijden.

In de tabel 'R40Statistics' zitten op reeds gegevens sinds begin april, echter is de API reeds een tijdje inactief (tot op heden), waardoor maar een drietal weken opgeslagen werd. Frontend wordt bijgevolg een gemiddelde getoond van die data (uit april), maar zal er geen real time informatie te zien zijn.

Wat betreft de informatie in de tabel 'Traveltimes', deze wordt nog maar bijgehouden sinds half mei, en bovendien heeft de server het al af en toe laten afweten, waardoor deze informatie in de database nog te beperkt is om reeds gemiddelden frontend te kunnen tonen.

Zoals in het deployment diagram beschreven werd, wordt gebruik gemaakt van JPA en JDBC. De tabellen in de database worden in de database module dan ook verwerkt als JPA objecten.

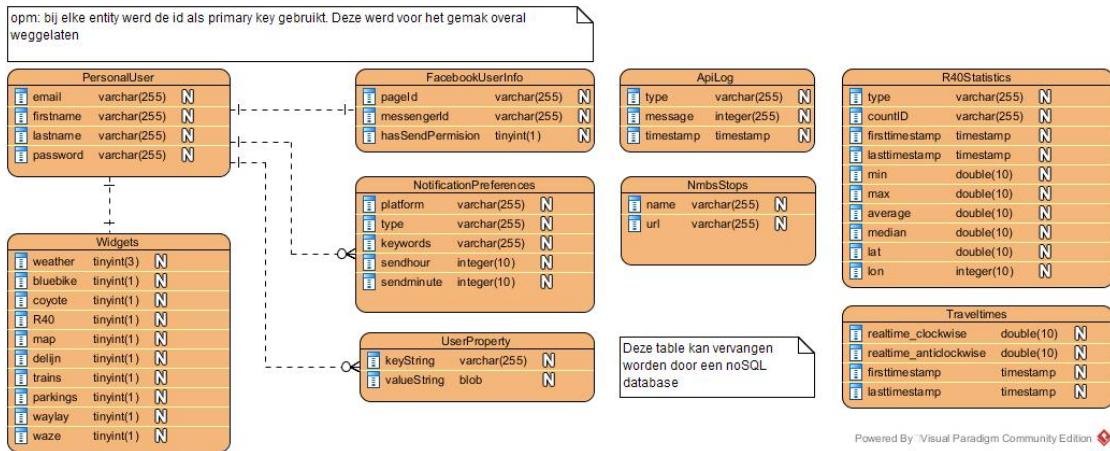


Fig. 30: Database tabellen

## 6 Hoofdstuk 4: Testplan

### 6.1 Testplan frontend

Als leidraad voor het testplan, werden de verscheidene use cases gebruikt, omdat deze beschrijven welke functionaliteiten de gebruiker zou moeten kunnen gebruiken, allen beschreven in sectie 4.5 in dit verslag.

Een overzicht van de verscheidene testen kan teruggevonden worden in onderstaande tabel.

Tabel 7: Overzicht frontend testen

Type	Naam	Use cases
Integration test	Weergeven van gegevens	Use case 1: de gebruiker bekijkt het dashboard
Integration test	Uitzetten API's	Use case 2: ophalen van gegevens uit verscheidene API's
Integration test	Testen van instellingen	Use case 3: de gebruiker bekijkt details van widgets & Use case 4: de gebruiker kiest welke widgets hij wenst te zien
Integration test	Testen van login en registratie	Use case 5: de gebruiker kan registreren. & Use case 6: de gebruiker kan inloggen
Integration test	Testen van persoonlijke pagina	Use case 7: de gebruiker kan specifieke informatie instellen
Integration test	Testen van notifications	Use case 8: de gebruiker kan kiezen op welke manieren hij meldingen krijgt, en krijgt deze op het correcte moment
Usability test	Testen door familie	Alle bovenstaande use cases

#### 6.1.1 Integration testen

##### Weergeven van gegevens

Een eerste use case beschrijft het volgende: de gebruiker bekijkt het dashboard. Wanneer het successscenario hierbij bekeken wordt, kan vastgesteld worden dat er een alternatief scenario dient te zijn wanneer het systeem er niet in slaagt om alle informatie uit de verscheidene API's op te halen. De uitwerking van dit alternatieve scenario wordt in deze subsectie beschreven.

De gegevens die worden weergegeven in de frontend, worden verkregen via REST calls naar de backend. Indien bij het inladen van de site de backend niet werkt, zullen er geen gegevens getoond worden. Deze situatie is zichtbaar in figuur 31, er worden dus gewoon allemaal lege widgets getoond.

De gebruiker kan dus merken dat er problemen zijn met de gegevens. Eens de backend terug opgestart kan worden, worden er meteen gegevens weergegeven. Hetzelfde resultaat wordt verkregen indien de gebruiker zonder internet valt. Een werkende backend en internetverbinding zijn dus beiden noodzakelijk.

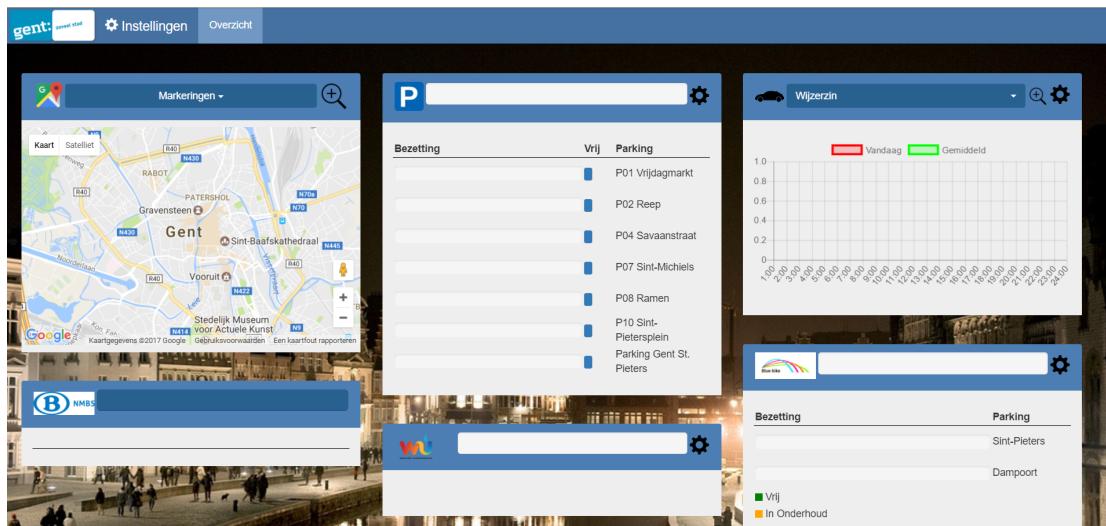


Fig. 31: Frontend zonder backend

### Uitzetten van API's

Een tweede use case gaat over het ophalen van gegevens uit verschillende API's, hierbij kan het namelijk voorkomen dat bepaalde API's al dan niet geladen kunnen worden.

In deze test wordt ervoor gezorgd dat bepaalde REST calls niet worden gemaakt, en wordt bekijken wat voor effect dit heeft op andere widgets. Indien de REST call voor parkings bijvoorbeeld uitgezet wordt, zal een melding gegeven worden dat er geen gegevens beschikbaar zijn. Hierbij kan nog een onderscheid gemaakt worden in de parking van de nmbs, en de andere parkings. Indien één van beide API's geen data doorgeeft, zullen de andere parkings wel nog geladen worden.

Bovendien mogen ze de andere widgets niet beïnvloeden, dit is dan ook het geval voor alle API's. Indien sommige gegevens niet meer of foutief worden doorgegeven, zal de widget stoppen met werken. Alle andere widgets laden namelijk nog steeds zonder problemen.

### Testen van instellingen

Bij deze test horen twee belangrijke use cases, namelijk de gebruiker die widgets meer in

detail kan bekijken en eventueel enkele kleine instellingen doen enerzijds, en anderzijds de gebruiker laten kiezen welke widgets hij wenst te zien en waar op de pagina die moeten komen.

Bij deze testen wordt dan ook gekeken of deze instellingen correct werken, en dit hoort natuurlijk manueel te gebeuren. Op de indexpagina zijn er twee opties voor instellingen. Enerzijds de algemene instellingen die men in de linkerbovenhoek kan terugvinden, en anderzijds de widgetinstellingen die in de rechterbovenhoek van elke instelbare widget te vinden zijn. In de algemene instellingen kan men bepaalde widgets naar keuze uitzetten. Indien men dit doet verdwijnt de widget, indien men ze terug aanzet komt de widget ook terug.

Daarnaast is er een drag-and-drop instelling. De index ondersteunt een drag-and-drop functionaliteit van de widgets, en kan men ook aan- of uitzetten. Bovendien kan men het aantal kolommen kiezen waarin de widgets verschijnen. Al deze instellingen werken correct en worden onthouden in de localStorage van de browser. Bij het herladen van de site worden de instellingen van de gebruiker onthouden.

De andere instellingen zijn dus die van de widgets, waarin bepaalde functionaliteiten per widget aan- of uitgezet kunnen worden en ook deze werken correct. Net als de algemene instellingen worden deze ook in de localStorage onthouden.

### **Testen van login en registratie**

Ook hier horen twee use cases bij, namelijk het registreren van een nieuwe gebruikersaccount en het inloggen met die gebruikersaccount. Het testen hiervan hoort ook manueel te gebeuren. Het registreren zal enkel succesvol zijn indien zowel het wachtwoord en het emailadres aan de voorwaarden voldoen. Een wachtwoord moet minstens 6 karakters lang zijn en minstens 1 cijfer bevatten. Het emailadres moet een geldig adres zijn. Indien niet aan deze voorwaarden voldaan wordt zal aan de gebruiker gemeld worden wat er precies verkeerd is. Indien het emailadres al geregistreerd is zal dit ook gemeld worden.

In het geval dat de backend niet werkt zal de registratiepopup een foutmelding weergeven, en is het registreren dus niet geslaagd. Ook zal de registratie niet slagen als de mailserver niet werkt, omdat dan geen verificatiemail kan gestuurd worden. Ook dit wordt gemeld aan de gebruiker.

Inloggen kan op 2 manieren. De eerste optie is inloggen met een geregistreerde gebruiker. Dit zal enkel werken als de backend werkt. Zoniet wordt een foutbericht weergegeven. De tweede optie is inloggen via Facebook. Hiervoor is geen registratie nodig. Ook dit zal enkel werken indien de backend werkt, omdat de personalisatie van deze gebruiker opgehaald moet kunnen worden.

### **Testen van persoonlijke pagina**

Deze use case beschrijft het gebruiken van de persoonlijke pagina. De gebruiker kan hier bijvoorbeeld parkings toevoegen. Foute input is niet mogelijk aangezien enkel de bestaande parkings weergegeven en geselecteerd kunnen worden. Indien er geen data

beschikbaar is over die parkings (enerzijds door het uitvallen van de backend of de API) zal dit weergegeven worden in de progress bar horende bij die parking.

Ook kan de gebruiker NMBS trajecten toevoegen in hun eigen widget. Dit doet hij door begin- en eindstation te selecteren uit een lijst van alle mogelijke stations, of door zelf de naam in te typen. Foute data ingeven is niet mogelijk, aangezien er gecontroleerd wordt of het ingegeven station zich wel degelijk in de lijst bevindt. Dit wordt dan ook duidelijk weergegeven aan de gebruiker. Hier zijn wel nog problemen met speciale tekens (zoals é, è, ç, etc.) omdat deze foutief worden doorgegeven vanop de API. Er moet ook een tijd meegegeven worden, maar dit gebeurt aan de hand van een kalender-inputvakje dus hier kan ook enkel correcte data doorgegeven worden. Tot slot kan elk traject toegevoegd worden aan favorieten. Dit is enkel toegestaan als alle ingevoerde gegevens correct zijn. Indien er geen gegevens beschikbaar zijn door uitgevallen backend of API wordt een passende boodschap weergegeven in de widget.

Specifieke trajecten van Coyote kunnen ook toegevoegd worden aan hun eigen widget. Deze trajecten kunnen enkel geselecteerd worden uit een lijst met alle mogelijke trajecten, dus een foute invoer is niet mogelijk. Bij het uitvallen van de backend of de API wordt bij elke route een volledig rode bar weergegeven met een bericht dat de gegevens niet opgevraagd kunnen worden.

Bovendien kunnen ook haltes van de Lijn opgevraagd en opgeslagen worden. Dit gebeurt door een adres in te geven of door uw eigen locatie te gebruiken. Bij een ongeldig adres kunnen er geen haltes opgevraagd worden, en dit wordt dan ook weergegeven aan de gebruiker. Het opslaan van een locatie is enkel mogelijk eens er haltes gevonden zijn voor die locatie. Bij het uitvallen van de backend of de API wordt weergegeven dat er geen bussen gevonden kunnen worden.

Al deze info wordt opgeslagen in localStorage en periodiek opgeslagen in de database, gelinkt aan de gebruiker. Ook bij het afsluiten van de pagina wordt de localStorage opgeslagen, zodat geen enkele wijziging verloren gaat. Indien de backend uitvalt zal dit opslaan niet lukken en zullen de gegevens niet opgeslagen worden. Dit wordt niet gemeld aan de gebruiker, aangezien de localStorage lokaal ook bijgehouden wordt en de gebruiker dit waarschijnlijk nog zal hebben de volgende keer dat hij de site bezoekt.

Tot slot kan de gebruiker nog specifieke instellingen van zijn gebruikersaccount wijzigen. Hier kan het wachtwoord gewijzigd worden, maar enkel indien het oude wachtwoord correct ingevoerd wordt en het nieuwe wachtwoord aan de voorwaarden voldoet. Indien er iets verkeerd ingevoerd wordt zal dit correct gemeld worden aan de gebruiker. Ook kan een profielfoto ingesteld worden door een url naar een foto in te voeren. Enkel geldige url's die een foto bevatten worden toegestaan. Ook hier worden fouten gemeld aan de gebruiker. Bovendien kan de naam gewijzigd worden. Hier liggen geen restricties op. En tenslotte kan de account verwijderd worden. Al deze instellingen zullen enkel werken indien de backend functioneel is. Zoniet wordt een foutbericht weergegeven.

### **Testen van notifications**

Deze use case beschrijft het instellen van notificaties. Voor elke opgeslagen parking, NMBStraject, Lijnhalte en Coyotetraject kunnen meldingen ingesteld worden. Dit gebeurt door per soort een tijdstip in te geven waarop de gebruiker een melding wil ontvangen en de manier waarop deze notificatie gestuurd moet worden aan te vinken. Het selecteren van het tijdstip gebeurt door een dropdown-menu dus verkeerde invoer is hier onmogelijk. Indien de gebruiker probeert een uur toe te voegen dat al toegevoegd is zal dit gemeld worden.

Notificaties kunnen verstuurd worden via mail of via Facebook Messenger. Alle testen hierop moeten via de server gebeuren. Mails zijn grondig getest en werken volledig, maar Facebook is nog in ontwikkelingsfase aangezien de site nog niet officieel geregistreerd is.

### **Wanneer worden de integration testen uitgevoerd?**

De meeste testen worden uitgevoerd vooraleer een nieuwe versie op de server wordt geplaatst, of wanneer een nieuwe build op github wordt geplaatst.

#### **6.1.2 Usability testen**

##### **Testen door familie**

De site werd getest door twee personen. De eerste persoon heeft veel ervaring met computers en sites. Deze persoon vond de site mooi en ontdekte al snel veel functionaliteiten zoals de drag-and-drop. Ook instellingen en de personalisatie werden snel gevonden. Dit volgt mooi de opgestelde use cases. Enkele opmerkingen waren dat de achtergrond te veel in contrast staat met de widgets, maar dit is opgelost in sprint 3. Weer een personalisatie aspect wat een belangrijk punt voor de site is.

Ook was het niet altijd duidelijk wat elke widget juist weergaf. Hiervoor zijn in sprint 3 tooltips voorzien met informatie over de inhoud.

De andere gebruiker was wat ouder. Deze vond alles heel duidelijk, wat goed overeenstemt met het idee van een overzichtelijk mobiliteitsdashboard. Enkele instellingen zoals drag-and-drop werden niet meteen gevonden. Hiervoor is in sprint 3 een instellingen knop voorzien met duidelijk opschrift 'drag-and-drop' zodat de gebruiker dit eenvoudiger kan ontdekken.

Er kan bijgevolg besloten worden dat de interface goed zit, hij voldoet aan de vooropgestelde use cases. Ook de instellingen personalisatie voldoen aan de use cases.

#### **6.2 Testplan backend**

Een overzicht van de verscheidene testen kan teruggevonden worden in onderstaande tabel.

Tabel 9: Overzicht backend testen

Type	Naam	Status
Automatische unit testen	Testen van elke getter afzonderlijk	Geïmplementeerd + geslaagd
Automatische unit testen	Testen van klasse ApiLoader	Geïmplementeerd + geslaagd
Integration test	Testen RESTful webservice	Geïmplementeerd + geslaagd
Integration test	Testen van het cachen	Geïmplementeerd + geslaagd
Integration test	Testen van de database	Geïmplementeerd + geslaagd

### 6.2.1 Automatische unit testen

Unit testen hebben enkel zin in de modules 'Getters' en 'Database'. Aangezien de module 'back' vooral de voorgaande modules combineert, worden daar integration testen voorzien. Bij het opzetten van automatische testsscripts is het zeer moeilijk om JPA-modules te integreren. Daarom worden enkel automatische unit testen voor de module 'Getters' voorzien.

#### Testen van elke getter afzonderlijk

Voor elke getterklasse wordt een unit test voorzien. De unit test voert vooral tests uit op de methodes van de interface 'ApiGetter'. Deze methodes worden gebruikt in andere modules en bijgevolg is het belangrijk om deze methodes zeker te testen.

#### Testen van de klasse ApiLoader

'ApiLoader' is een soort dao naar de apiGetters toe. De unit test voert elke public methode uit en controleert op een juiste returnwaarde. Ook bij foute invoer wordt gecontroleerd of een juiste exception wordt teruggegeven. Zo dient bijvoorbeeld een 'ApiNotFoundException' opgeworpen te worden bij het opvragen van een API die niet bestaat.

#### Frequentie van de unit tests

Een unit test wordt telkens uitgevoerd bij het creëren van een nieuwe getterklasse of bij een grote wijziging van de interne structuur ervan. Bij elke wijziging wordt ook de unit test voor de Loader opnieuw uitgevoerd. De testen simuleren immers de verschillende aanvragen van andere modules.

## 6.2.2 Integration testen

### Testen van de RESTfull webservice

Een goede test is om via een httpclient zelf calls te sturen naar de backend. Op de GitHub Wiki staan alle mogelijk calls opgesomd. Door via een client (bv. Postman) deze calls, al dan niet met foute input, zelf te verzenden kan op een betrouwbare manier achterhaald worden of de service naar behoren werkt. Om dit proces te automatiseren maken we gebruik van een online framework, 'Assertible', te zien in figuur 32. Op dit framework hebben we alle mogelijke requests verzameld. Met een enkele druk op de knop worden alle calls uitgevoerd en wordt overzichtelijk weergegeven of ze al dan niet slagen. Failen sommige testen, dan heeft het zeker nog geen nut om de frontend verder te testen.

Deze testen worden het best uitgevoerd telkens een nieuwe backend wordt gedeployed naar de server. Op die manier kan er in een oogopslag worden geverifieerd of de frontend zal kunnen werken.

The screenshot shows the Assertible interface with the URL 'mobi-02.project.tiwi.be'. The top navigation bar includes tabs for Overview, Tests (selected), Results, Monitoring, Deployments, and Settings. Below the tabs is a search bar and a 'All environments' dropdown. The main area displays a grid of test results. Each row represents a different API endpoint and its status. The columns are: Endpoint, Status, and Result. Most results are 'passing' with a green button, while some are 'pending' with a grey button. The endpoints listed include: '/', '/back/res/apidata/train/Gent-Sint-P...', '/back/res/apidata/bluebike', '/back/res/statistics/traveltimes/hist...', '/back/res/apidata/gipod', '/back/res/user/properties/key', '/back/res/statistics/traveltimes/', '/back/res/apidata/coyote', '/back/res/statistics/r40/avg/hist/tot...', '/back/res/apidata/waze', and '/back/res/user/properties/'. The status column shows '200 OK' for most rows.

Fig. 32: Online framework Assertible

### Testen van het cachen

Het cachen gebeurt door samenwerking van een groot aantal klassen (zie sequentiediagram hierboven). De gemakkelijkste manier om de cache te testen is via de REST service. Bijna alle API's worden onafhankelijk van de REST service opgehaald in de backend. Bij een REST call wordt data uit de cache gehaald. Het weer wordt hierbij als voorbeeld gebruikt. Bij een get call naar 'serverlocatie/res/apidata/weather' zijn er 3 mogelijkheden:

1. Het antwoord is een ApiRequestException.

2. Het antwoord heeft geen status 200.
3. Je krijgt het verwachte antwoord.

Geval 1 en 2 duiden duidelijk op een probleem, maar geval 3 sluit ook niet uit dat er iets niet klopt. Het is aangewezen om deze test een aantal keer te herhalen met een tussenpauze. Als de test wordt uitgevoerd in de voor- en namiddag, is het onmogelijk dat je bijvoorbeeld treinen na een paar uur nog hetzelfde antwoord krijgt. In dit geval is er dus ook een probleem met het cachen.

### **Testen van de database**

Het testen van de database is iets ingewikkelder. De database werkt enkel in een volledige Glassfish-omgeving, bij testscripts wordt bijvoorbeeld enkel een tijdelijke omgeving opgezet. Toch zijn er een aantal mogelijke testen die handmatig kunnen worden uitgevoerd:

1. Een eerste goede test is het controleren of Glassfish wel degelijk met de database kan communiceren. Dit kan door pingen via de admin console, zie subsectie 6.1.4 voor meer informatie. Controleer ook of de connection pool wel degelijk 'jdbc/mobi02' heet.
2. Het deployen zelf van de applicatie faalt als er geen communicatie is met de database. Dit is ook een goede indicatie, controleer bijgevolg zeker stap 1 als dit het geval is.
3. Indien exceptions optreden bij het ophalen van data via een getter, worden deze gelogd naar de database. Je kan deze exceptions forceren door bijvoorbeeld de toegang tot het internet te blokkeren (indien de database lokaal draait) of door een foute URL te configureren in de propertiesbestanden. De tabel 'api\_log' zou dan nieuwe exceptions moeten bevatten.
4. Elk uur worden ook statistieken van de R40 gepersist naar de database. Laat de applicatie een uurtje draaien en controleer of de tabel 'R40' data bevat.

### **Wanneer worden de integration testen uitgevoerd?**

De meeste testen worden uitgevoerd voor een nieuwe versie op de server wordt geplaatst. Of wanneer een nieuwe build op de github wordt geplaatst.

## **7 Hoofdstuk 5: Evaluatie en discussies**

### **7.1 Performantie**

De applicatie bestaat uit een aantal afzonderlijke componenten. Hierdoor is het mogelijk om eventuele bottlenecks makkelijker op te sporen en ervoor te zorgen dat ze de volledige

applicatie niet vertragen. Ook kan de performantie van elk onderdeel eenvoudig worden besproken.

### 7.1.1 Op component niveau

#### De database

De databaseconnectie wordt gerealiseerd door JPA. Dit heeft als voordeel dat de applicatie volledig onafhankelijk is van het type SQL-database. Enkele database afhankelijke queries worden geïmplementeerd met stored procedures en kunnen dus worden aangepast voor andere databases. De implementatie van de queries voor toevoegen, verwijderen, updaten van tabellen en records wordt ook volledig geïmplementeerd door JPA. Hierdoor is een initiele configuratie van de database eigenlijk niet nodig. Bij een eerste opstart zal de applicatie zelf de nodige queries uitvoeren voor het creeëren van tabellen. De entity manager zelf en zijn transacties worden beheerd door de webcontainer zelf. Door gebruik te maken van EJB's is het immers mogelijk om overal waar nodig via dependency injection een referentie naar die manager te bekomen. Indien er een groot aantal calls naar de database wordt uitgevoerd, zal de container op een efficiënte manier de transacties beheren.

#### De API-getters

Voor elke afzonderlijke API bevat de applicatie een afzonderlijke getterklasse en een modelklasse. Er is ook een extra model die een exceptie voorstelt. Er is immers geen enkele garantie dat een API wel degelijk bereikbaar is. Vertoont een API onverwacht gedrag dan zal er in plaats van een API-model van het juiste type, een exceptiemodel worden aangemaakt met de beschrijving van het probleem. Beide soorten models implementeren immers dezelfde interface en er zal in de verdere loop van de applicatie geen onderscheid gemaakt worden tussen exceptie of verwacht resultaat. Deze manier van werken zorgt ervoor dat onbereikbare API's de rest van de applicatie niet beïnvloeden.

### 7.1.2 Op systeem niveau

#### Schaalbaarheid

De communicatie tussen back- en frontend gebeurt via een RESTful API. Deze component moet dan ook het grootst aantal requests realiseren. Het cachen van data speelt hierbij een belangrijke rol. Voor de cache is er besloten een Hashmap te gebruiken. Het was immers geen goed idee om deze op te slaan in een database of weg te schrijven naar een bestand. Er wordt frequent naar geschreven en nog frequenter van gelezen. De data bevindt zich dus in het geheugen. Dit heeft als nadeel dat de applicatie ietwat meer geheugen verbruikt. Als grote voordeel is de applicatie een stuk schaalbaarder. Het maakt niet uit hoeveel requests er gemaakt worden, de data blijft beschikbaar in het geheugen. Ook de database speelt hier een belangrijke rol, maar deze component werd

reeds besproken in de vorige sectie.

### **Robuustheid**

Zoals reeds besproken bij de API-getters, worden fouten vanuit externe bronnen meteen opgevangen. Dit zorgt ervoor dat de applicatie daar verder geen rekening moet mee houden. Ook de EJB-container speelt hier een rol. Loopt er toch ergens iets mis, dan zal automatisch een eventuele transactie teruggedraaid worden. Foute input naar de RESTful webservice of database wordt eveneens opgevangen. Ter info: omdat de applicatie zich nog in een testfase bevindt werd er gekozen om alle fouten nog steeds uit te schrijven naar de logs, ook al wordt deze fout opgevangen.

## 7.2 Security

In de applicatie zijn niet meteen extra beveiligingsmechanismen ingebouwd. Het leek dan ook niet prioritair voor dit project vermits op dit moment nog weinig kennis van security-aspecten aanwezig is. Toch werd er gebruikgemaakt van enkele technieken om de beveiliging te garanderen:

1. Er werd een SSL-certificaat geïnstalleerd op de server. Hierdoor is alle communicatie tussen front en back versleuteld.
2. De server is zo geconfigureerd dat het enkel verbinding maakt met vertrouwde sites. Dit wil zeggen dat verbindingen met versleutelde sites waarvan het certificaat zich niet in de trustore bevindt niet zullen worden opgezet.
3. SQL -en javascript injectie is niet mogelijk.
4. Voor calls naar de REST-API die te maken hebben met een gebruiker moet men ingelogd zijn. Zoniet, dan zal een filter de request opvangen.

## 7.3 Problemen en geleerde lessen

Wanneer globaal teruggekeken wordt op de gehele evolutie van het project, zijn er drie zaken die in het oog sprongen op vlak van problemen. Deze worden hieronder even kort toegelicht.

### **De Lijn**

Zoals in voorgaande secties reeds aan bod kwam, was De Lijn telkens een buitenbeentje wat betreft het ophalen en verwerken van informatie. Een eerste probleem hierbij kwam vrij snel aan het licht, De Lijn werkt met Lambert72, terwijl via Google Maps de coördinaten niet in dat stelsel bekomen kunnen worden. Er was hier dus sprake van een

conversieprobleem. Dit is gedurende sprint 2 opgelost geraakt door het zoeken naar zo'n transformatie, en dit volledig om te zetten naar Java-code.

Vervolgens kwam voornamelijk het performantieprobleem naar voren. De Haltelink API die gebruikt wordt, blijkt veel trager te gaan dan andere API's, waardoor het laden te lang duurt, en de gebruiker ongewenst lang moet wachten op zijn gevraagde data.

Tijdens het ontwikkelen werd bovendien gekozen om De Lijn zodanig op te bouwen, dat op basis van een adres of locatie de dichtstbijzijnde haltes bepaald worden, om vervolgens de eerstkomende bussen te bepalen. Echter is het zo dat men zich vaak in de buurt van meerdere haltes bevindt, zeker in Gent. Hierdoor dient er niet één, maar dienen wel meerdere calls te gebeuren. Tijdens sprint 2 verliepen al deze calls synchroon, met als gevolg dat de wachttijd dus zeer sterk opliep.

In sprint 3 werd dit probleem aangepakt, en werd gekozen om bussen op te halen via een ajax call, met andere woorden asynchroon. Dit werd ook reeds aangehaald in paragraaf 5.3.

Na testen van deze vernieuwing, blijkt dat deze aanpak zorgt voor minder belasting op de server, en voor een kortere wachttijd. Kortom, deze verandering was alvast een stap in de juiste richting voor het optimaliseren van De Lijn.

### Glassfish memory

Gedurende het gehele project bleek de server regelmatig stil te vallen. Soms lag dit aan errors die niet correct afgehandeld werden, maar soms ook door het 'out-of-memory' treden van de server. Door het bekijken van de logfiles, werden de aanwezige errors tijdens de ontwikkeling telkens opgespoord, om vervolgens ook aan te pakken.

Echter bleef tot het einde de server soms nog crashen, terwijl in de logfiles niet altijd iets terug te vinden was. Na veel zoeken naar een oorzaak, en het monitoren van het project, kon nog steeds geen exacte oorzaak aangewezen worden. Dit heeft ertoe geleid dat gekozen werd om een versie van het project op de server te plaatsen, die zo weinig mogelijk geheugen verbruikt. Op dit ogenblik is dat een versie zonder de timer die de notifications zou triggeren. Notifications op zich waren ook nog in ontwikkelingsfase, en zijn nog niet voldoende geoptimaliseerd om op de relatief onstabiele server erbij te voegen. Zie ook hieronder het puntje notificaitons voor meer informatie.

Met behulp van VisualVM werd het project lokaal heel wat uren gemonitord, en bekamt men een resultaat zoals in figuur 33 zichtbaar is. Hierbij is een duidelijke zaagtand te zien, met een piek om de 2 minuten. Dit kan eenvoudig verklaard worden door het feit dat elke 2 minuten de caching timer gaat checken of er informatie bijgewerkt moet worden en al dan niet enkele API calls uitgevoerd worden. Hieruit kan gezien worden dat het geheugenverbruik toch niet uitermate groot is.

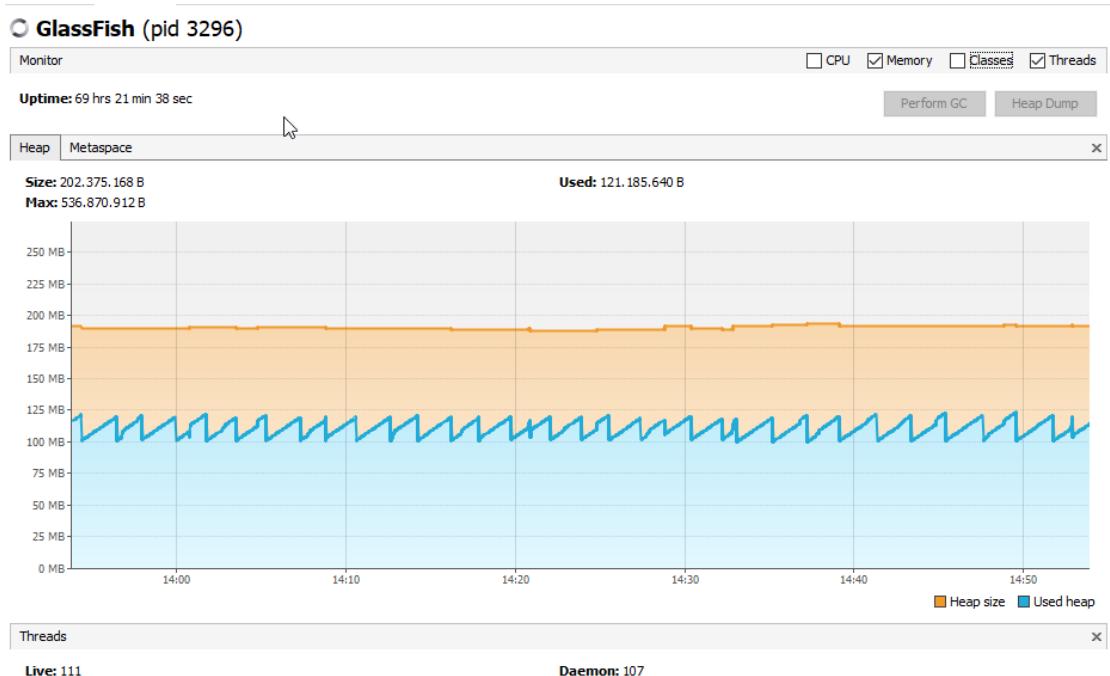


Fig. 33: Monitoring VisualVM

Ook bleek met momenten dat het geheugenverbruik van Glassfish zelf soms heel hoog lag, zelfs wanneer op dat moment geen code op de server gedeployed was. Doordat hier dus reeds een groot deel van het geheugen naartoe gaat, was de vrije geheugencapaciteit te beperkt om het systeem met notifications in te laten draaien op de server.

## Notifications

Zoals in het sequentiediagram van de notifications zichtbaar is, worden er heel wat lussen doorlopen om tot de notifications te komen. Het systeem hierbij werkt lokaal wel, maar is zeker en vast niet het meest efficiënte. Het bevindt zich duidelijk nog in de ontwikkelingsfase. De ruwe bolster is er, maar het moet zeker nog geoptimaliseerd worden alvorens het in productie kan treden.

Het zoeken naar een betere uitwerking, waarbij teveel geneste forlussen vermeden worden, focust zich in dit geval niet enkel op de code in deze package, maar ook op een optimalisatie van het opslaan van de gegevens in de databank. Er moet geoptimaliseerd worden voor meer toegankelijke manier van opslag, misschien gedetailleerder. Er moet gezocht worden naar een query die reeds meer gefilterd is, zodat achteraf geen grote lussen doorlopen moeten worden. Om zo'n query te realiseren, zal ook de frontend afgestemd moeten worden, vermits er eventueel andere data of op een andere manier data opgeslagen moet worden.

## 8 Hoofdstuk 6: Handleidingen

### 8.1 Installatiehandleiding

#### 8.1.1 Beschrijving software

##### **Java webcontainer**

De applicatie is een Java-webapplicatie. Die heeft een compatible webomgeving nodig om te functioneren. Tijdens het ontwikkelen en testen wordt gebruikgemaakt van een Glassfish-webcontainer. De installatieomschrijving zal dan ook specifieke Glassfish instellingen en screenshots bespreken. Het is sterk aangeraden om dus ook een Glassfish server te gebruiken, maar een andere compatibele webcontainer zou hetzelfde resultaat moeten geven.

##### **Database**

Elke database die de MySQL-standaard implementeert is compatibel met de applicatie. Andere SQL-database varianten echter niet.

##### **Applicatiebestanden**

De applicatie wordt gebundeld in twee .war-bestanden. Ze bevatten de frontend en de backend, en worden bij de installatie gedeployed in de webcontainer.

#### 8.1.2 Benodigheden

- Glassfish-webcontainer
- Html, CSS en Javascript compatibele webserver
- MySQL-database (MariaDB 10.1 bij voorkeur)
- Applicatie .war-bestand
- glassfish-resources.xml configuratiebestand

Opmerking: Indien men gebruik wenst te maken van de stored procedures uit dit project, is een MySQL-database nodig die de functies day(), date() en hour() ondersteunt, zie paragraaf 'Stored procedures'. De stored procedures zijn de enige componenten die database-afhankelijk zijn. Er kan van een andere, JPA compatibele, database gebruikgemaakt worden indien gewenst, als de stored procedures worden aangepast.

### 8.1.3 Installatie en configuratie van de database

#### Installatie

Bekijk voor de installatie van de MySQL-database de documentatie van de database die wordt gebruikt. Enkele voorbeelden zijn: MySQL-Database ([mysql.com](http://mysql.com)), MariaDB ([mariadb.com](http://mariadb.com))...

#### Configuratie

Na het installeren moet er een nieuwe database worden geconfigureerd. Stappen 3 en 4 zijn optioneel, maar het is sterk aangeraden om niet met de rootgebruiker te werken in de applicatie.

1. Open na de installatie een MySQL command prompt en log in met het root account.
2. Maak een nieuwe database
3. Maak een nieuwe gebruiker
4. Geef de nieuwe gebruiker rechten op de database

#### SQL-script voor het aanmaken van een nieuwe database en gebruiker:

```
1 create database <Databasenaam>;
2 create user <Gebruikersnaam> identified by <Wachtwoord>;
3 grant ALL on <Databasenaam> to <Gebruikersnaam>;
```

Opmerkingen: - Onthoud zeker de databasenaam, gebruikersnaam en wachtwoord. Die zijn in de volgende stappen nodig om de Glassfish te installeren. - Er zijn gebruikers met verschillende rechten voorzien, iedereen heeft dezelfde rechten. - Om de stations voor de API calls ook lokaal te kunnen gebruiken, dient het bestandje "Stops.sql", te vinden in de map 'SQL', uitgevoerd worden.

### 8.1.4 Stored Procedures

De backend maakt gebruik van een aantal stored procedures, die geven allemaal een dubbele array terug, afhankelijk van de gevraagde waarde. De scripts zijn te vinden op: <https://github.ugent.be/iii-vop2017/mobi-02/wiki/MySQL-Stored-Procedures>. Andere implementaties met hetzelfde resultaat zijn uiteraard ook mogelijk (bijvoorbeeld als een ander databasedistributie wordt gebruikt).

### 8.1.5 Installatie en configuratie van de Glassfish-webcontainer

Bekijk de documentatie van Glassfish voor de initiële installatie van de container.

#### Een nieuw domein aanmaken en starten

Na de installatie is het nodig om een domein aan te maken en het domein te starten. Hiervoor wordt gebruikgemaakt van de **asadmin** tool, te vinden in de bin/folder van de Glassfish-map.

#### Asadmin commando's voor het aanmaken van een domein:

```
1 asadmin create-domain <domeinnaam>
2 asadmin start-domain <domeinnaam>
```

#### Glassfish configureren voor communicatie met de database

Voor het volgende deel van de configuratie wordt gebruikgemaakt van de adminconsole. Dit is een grafische interface, te bereiken via de url '<https://localhost:4848>'.

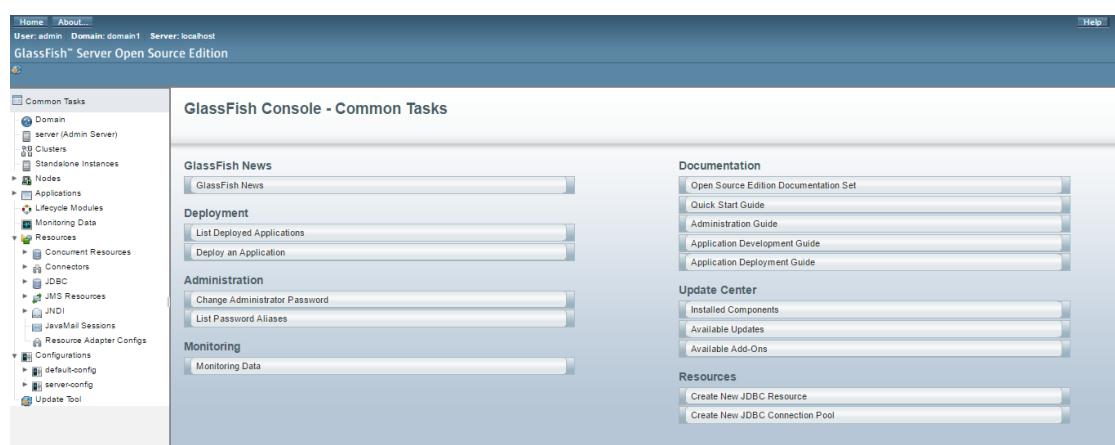


Fig. 34: Glassfish-adminconsole

De configuratie van de database gebeurt via een configuratiebestand. Dit is het bestand 'glassfish-resources.xml'.

#### Fragment uit glassfish-resources.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish
   Application Server 3.1 Resource Definitions//EN" "http://
   glassfish.org/dtds/glassfish-resources_1_5.dtd">
3 <resources>
```

```

4   <jdbc-resource enabled="true" jndi-name="jdbc/mobi02">
5     object-type="user" pool-name="mobidb">
6       <description/>
7     </jdbc-resource>
8     <jdbc-connection-pool allow-non-component-callers="false"
9       ... wrap-jdbc-objects="false">
10      <property name="URL" value="jdbc:mysql://localhost:3306/
11        {Databasenaam}?zeroDateTimeBehavior=convertToNull"/>
12      <property name="User" value="{Gebruikersnaam}"/>
13      <property name="Password" value="{Wachtwoord}"/>
14    </jdbc-connection-pool>
15  </resources>

```

Pas de variabelen 'Databasenaam', 'Gebruikersnaam' en 'Wachtwoord' aan naar de gegevens van de database. Vervolgens kan de webcontainer ingesteld worden voor communicatie met de database:

1. Glassfish heeft een connector nodig voor MySQL, die is te vinden op de site: '<https://dev.mysql.com/downloads/connector/j/5.1.html>'
2. Plaats de .jar in de lib-folder van het ingestelde domein. Dit is meestal: 'glassfish-installatie-pad\domains\domein-naam\lib'
3. Herstart de Glassfish server
4. Vervolgens kan het aangevulde configuratiebestand, 'glassfish-resources.xml', worden opgeladen via de admin console. Dit kan door het bestand te uploaden in de subsectie 'Resources'. Druk vervolgens op 'Add Resources' en specifieer het bestand.
5. Voer een database-ping uit ter controle. Dit kan via de subsectie 'JDBC' , druk vervolgens op 'JDBC Connection Pools' en kies de 'mobidb' connection pool.
6. Als alle voorgaande stappen correct zijn uitgevoerd, verschijnt na het pingen het bericht: 'Ping Succeeded'. Is dit niet het geval, verifieer dan of stap 2 en 3 correct zijn uitgevoerd en of de juiste databasegebruiker is ingesteld. Dit laatste kan worden gecontroleerd in het tablad 'Additional Properties' in de subsectie 'JDBC Connection Pools'.

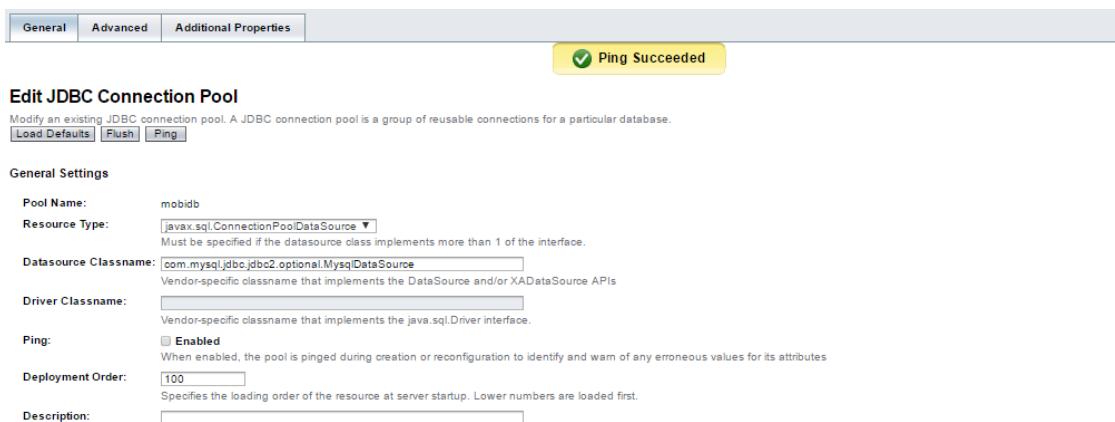


Fig. 35: Geslaagde ping

## Private certificaten

Voor bepaalde functionaliteiten heeft de server een geldig https-certificaat nodig. Op deze pagina meer info over het importeren van een certificaat in glassfish via Letsencrypt: [https://github.ugent.be/iii-vop2017/mobi-02/wiki/Certificaat-installeren-in-glassfish-\(via-letsencrypt\)](https://github.ugent.be/iii-vop2017/mobi-02/wiki/Certificaat-installeren-in-glassfish-(via-letsencrypt))

Opmerking: locatiebepaling in Chrome en Messenger integratie zal niet werken zonder geldig certificaat. Self-signed blijkt niet voldoende te zijn.

## Publieke certificaten

Om er zeker van te zijn dat de back end enkel communiceert met vertrouwde servers moeten de publieke certificaten worden toegevoegd aan de glassfish trust-store (cacerts.jks). In de map 'Public Certificaten' in de root van de repository zijn de certificaten verzameld met een shellscript die de juiste imports zal uitvoeren.

### 8.1.6 Installatie van de applicatie

#### Deployen van de backend in de Glassfish-webcontainer

Het deployen van de applicatie gebeurt via de sectie 'Applications' in de admin console. Kies daar voor 'Deploy' en kies vervolgens het juiste .war bestand. Na het deployen is het mogelijk om de applicatie te starten via hetzelfde menu.

#### Configuratie en installatie van de frontend

De frontend is een simpel webproject. Het is echter wel nodig om het javascript bestand: 'constants.js' aan te passen. De variabele host moet naar het extern ip-adress van de backend wijzen. Bij een juiste configuratie van zowel de front- als backend, wordt de data nu correct ingeladen.

## **constants.js met backend url https://mobi-02.project.tiwi.be:8181**

```
1 var host = "https://mobi-02.project.tiwi.be:8181";
```

Opmerking: indien de url zonder https protocol werkt, zullen sommige browsers de site als onveilig beschouwen.

## 8.2 Gebruikershandleiding frontend

### 8.2.1 Gebruikershandleiding hoofdpagina

#### **Site algemeen**

De hoofdpagina van de site geeft standaard alle widgets weer. In elke widget is de informatie over één specifiek onderwerp terug te vinden. Dit onderwerp wordt weergegeven door het logo linksboven elk widget. Voor nog meer informatie, kan over het logo gehoverd worden met de muis, waarna een tekstuele beschrijving verschijnt. Bovendien is het ook mogelijk om widgets te verplaatsen door middel van een drag-and-drop mechanisme. De site past zichzelf continu aan en plaatst alles weer in een raster, er is sprake van een responsive design.

#### **Widgetinstellingen**

Bij sommige widgets is er rechtsboven een icoontje voor instellingen te vinden. Door hierop te klikken ziet u de specifieke instellingen van deze widget. Zo kan bij de parking-widget bijvoorbeeld ingesteld worden welke parkings wel of niet weergegeven worden, en kan men bijvoorbeeld bij de weerwidget de eenheden aanpassen. Alle widgetinstellingen worden opgeslagen en zullen nog steeds hetzelfde zijn indien u de site later nogmaals bezoekt.

#### **Algemene instellingen**

In de linkerbovenhoek van de site is ook een instellingenknop te vinden. Hier kan je de algemene dingen op de site instellen. Zo is het mogelijk om elke widget naar believen te verwijderen of weer toe te voegen. Ook kan de drag-and-drop functie uitgeschakeld (en weer ingeschakeld) worden, en kan het aantal kolommen waarin de widgets worden weergegeven ingesteld worden.

Tot slot kan het thema van de site veranderd worden en kan de achtergrond gewijzigd worden. Dit instellingenpaneel kan weer gesloten worden door opnieuw op de instellingenknop te klikken, door op het kruisje rechtsboven te klikken indien de instellingenknop niet meer zichtbaar is, of door gewoon ergens naast het paneel te klikken. Alle instellingen worden opgeslagen en zullen steeds hetzelfde zijn indien u de site later opnieuw bezoekt.

### **Kaartje**

Het kaartje (de widget die origineel rechtsboven staat) heeft nog wat extra functionaliteiten. Deze zijn te vinden in het dropdown-menu bovenaan de widget. Eerst en vooral kan u uw eigen locatie weergeven op de kaart. Dit doet u door het locatie-vakje aan te vinken. Indien het de eerste keer is dat u dit doet zal uw browser u om toestemming vragen om uw locatie door te geven. Dit moet u accepteren om deze functionaliteit te laten werken.

Vervolgens kan veel informatie uit de andere widgets ook op de kaart weergegeven worden. Als u dan in die widget klikt op een bepaald onderdeel van de widget, zal dit onderdeel gecentreerd worden op de kaart. Tot slot kan de kaart ook vergroot worden door op het vergrootglasicoontje rechtsboven in de widget te klikken. Hierdoor wordt de kaart de breedte van het volledige scherm. Door nogmaals op dit icoontje te klikken wordt de widget weer dezelfde grootte als de andere widgets.

### **Tellingen R40**

De widget met de tellingen van de R40 heeft ook wat extra functionaliteit. Deze widget staat standaard rechtsboven. Origineel wordt hier de gemiddelde telling van alle meetpunten van vandaag getoond (in het groen), ten opzichte van de gemiddelde telling van alle meetpunten in het verleden (in het rood). Om de informatie te zien van één specifiek meetpunt, kan dit meetpunt geselecteerd worden in het drop-down menu rechtsboven de widget.

Ook kunnen alle meetpunten op de kaart getoond worden en kan je het weer te geven meetpunt selecteren door er gewoonweg op te klikken op de kaart. Op de grafiek wordt per uur een telling weergegeven. Je kan het precieze getal van deze telling zien door over het punt op de grafiek te hoveren. Ook is het mogelijk om één van de grafieken uit te schakelen door er in de legende op te klikken.

Bovendien kan u, via de instellingen van die widget, de gegevens op de grafiek veranderen naar de gemiddelde reistijd rond de R40, zowel in wijzerzin als in tegenwijzerzin. Zowel deze data als de tellingen op de verschillende meetpunten kunnen weergegeven worden voor elke dag van de week, door de gewenste dag te selecteren in de instellingen. Tot slot kan ook deze widget vergroot worden door op het vergrootglasicoontje rechtsboven in de widget te klikken. Hierdoor wordt de widget 2/3 van de breedte van het volledige scherm. Door nogmaals op dit icoontje te klikken wordt de widget weer dezelfde grootte als de andere widgets.

## 8.2.2 Gebruikershandleiding persoonlijke pagina

### **Registreren en inloggen**

Rechtsboven de pagina vindt u een registratie- en loginknop. Hier kan u zich registreren

als nieuwe gebruiker of inloggen in het geval u al een account hebt. Inloggen via Facebook is ook mogelijk. Eens ingelogd hebt u toegang tot uw persoonlijke pagina.

### **Mijn Paneel**

In 'Mijn Paneel' kan u verschillende ondergrondse parkings en BlueBike parkings toevoegen, zodat u een makkelijk overzicht heeft van uw meest bezochte parkings.

### **Mijn NMBS**

Hier kan u uw NMBS route plannen. Uw begin- en eindstation kan geselecteerd worden uit de lijst met stations, mogelijks door de zoekfunctie. Bij tijdstip kan u zelf een datum en tijd invullen via de kalender-widget of gewoon het huidige moment kiezen door op de NU-knop de klikken. Eens op de OK-knop geklikt is wordt een overzicht getoond van de eerstvolgende mogelijke routes, inclusief overstappen. Indien u deze route vaak gebruikt, kan deze opgeslagen worden bij favorieten. Vanuit favorieten kan u dan makkelijk uw opgeslagen routes weer inladen.

### **Mijn Routes**

Hier kan u uw routes toevoegen. Routes zijn korte stukjes van de R40, R4 of N70. Dit toevoegen doet u door, eens op de Toevoegen-knop geklikt te hebben, een route te selecteren uit de linkertabel en deze toe te voegen aan de rechtertabel. Voor al uw toegevoegde routes kan u zien hoeveel vertraging er individueel is. Voor meer info kan u op het info-symbool naast elke route klikken. Bovenaan de widget ziet u dan de totale vertraging op al uw routes.

### **De Lijn**

Hier kan u de verschillende bussen en hun vertraging van een specifieke halte bekijken. Deze halte wordt bepaald door ofwel te zoeken in een bepaalde in te stellen radius rond uw eigen locatie, of rond een bepaald adres. Eens de haltes ingeladen zijn kan u die locatie ook instellen als favoriet en vanuit favorieten weer inladen.

### **Instellingen**

Ook hier is in de linkerbovenhoek een instellingen-knop te vinden. Hierdoor opent u het instellingenvenster. Op de eerste tabpagina van dit venster kan u algemene instellingen over uw account wijzigen, zoals bijvoorbeeld uw wachtwoord, uw profielfoto of uw naam. U kan uw account hier ook verwijderen. Op de tweede tabpagina kan u notificaties instellen. Dit is mogelijk voor elk onderdeel van Mijn Pagina. Om te zien waar u precies notificaties over zal krijgen, kan u hoveren over het vraagteken-symbool naast elke soort notificatie. U kan de uren van deze notificaties instellen door een uur te selecteren uit de dropdown en op het plusje te drukken. Ook het type kan u selecteren door het juiste vakje aan te vinken naast elk type. Voor Messenger-notificaties moet u wel eerst toestemming geven vanop uw Facebook-account. Voor meer info hierover, hover over het vraagteken-symbool.

## 9 Hoofdstuk 7: Besluit

Waar zijn er files? Hoe zit het met de bezetting van de parkings? Welke treinen hebben vertraging? Waar zijn er wegenwerken? Een stad zonder verkeersinformatie, blijkt onmogelijk te zijn. Stad Gent startte dan ook in 2014 met het oprichten van een regionaal verkeerscentrum, om zo een beter zicht te krijgen en kunnen geven omtrent de verkeerssituatie in Gent.

In Gent heeft men op dit moment wel een Twitterpagina die informatie geeft over de real time verkeerssituatie, maar voor de rest blijft de communicatie naar buiten toe vrij beperkt. Een oplossing voor dit probleem zou kunnen zijn om een dashboard te bouwen, waarbij de gebruiker eenvoudig en gestructureerd alle informatie omtrent het verkeer in één oogopslag kan waarnemen.

In dit verslag werd het project 'Mobiliteitsdashboard op maat van Stad Gent' algemeen geschatst, alsook gedocumenteerd hoe de implementatie ervan opgebouwd werd. De ontwikkeling van dit project liep in drie fasen, die eigenlijk kunnen samengevat worden als: "structureren, optimaliseren en personaliseren".

Eerst en vooral werd de focus gelegd op 'hoe' krijg je zoveel mogelijk zinvolle informatie tot bij de gebruiker. Het dashboard werd opgebouwd met informatie van talrijke API's, op een eenvoudige manier weergegeven.

Vervolgens werd gefocust op de optimalisatie van deze gestructureerde weergave. Kortom, er werden extra widgets toegevoegd zoals een kaartje en grafieken. De informatie werd op een nog grafischere en dus overzichtelijker manier weergegeven. Daarnaast werd ook de inleiding naar de derde fase gerealiseerd, de gebruiker kon bepalen welke widgets hij wou zien, alsook waar die moeten staan.

Tot slot werd de personalisatie doorgedreven, het dashboard heeft als doel de mensen een algemeen beeld te geven, maar ook de kans te geven om enkel de voor hen relevante informatie te kunnen zien. Er werd een persoonlijke pagina ontwikkeld, waarbij men bussen en treinen op maat kan opzoeken en deze al dan niet kan onthouden als favorieten. Daarnaast werd een mogelijkheid gebouwd tot het bekomen van meldingen op gewenste tijdstippen en met gewenste data.

Conclusie is dat er een dashboard opgebouwd werd, dat real time informatie weergeeft over de mobiliteitssituatie in Gent. Dit dashboard kent bovendien reeds de uitbreiding naar een persoonlijke pagina, en er werd reeds een eerste versie van meldingen mogelijk gemaakt.

## 10 Referentielijst

- Morlion P. (2017). *Online Mobiliteitsdashboard*. Geraadpleegd op 13 maart 2017 en 24 maart 2017 via <http://slides.com/pietermorlion>
- Oracle (2015). *GlassFish Server Documentation*. Geraadpleegd op 22 maart 2017 via <https://glassfish.java.net/documentation.html>
- Oracle (2017). *MySQL Documentation*. Geraadpleegd op 22 maart 2017 via <https://dev.mysql.com/doc/>
- Assertible (2017). *Assertible Dashboard*. Geraadpleegd op 15 mei 2017 via <https://assertible.com/>

## 11 Bijlagen

**Bijlage 1:** Resultaten enquête

**Bijlage 2:** Analyse API's

**Bijlage 3:** Klassendiagram module Getters

**Bijlage 4:** Klassendiagram package Getters

**Bijlage 5:** Klassendiagram 'Model' package

**Bijlage 6:** Klassendiagram frontend hoofdpagina

**Bijlage 7:** Klassendiagram frontend persoonlijke pagina

**Bijlage 8:** Sequentiediagram package Caching

**Bijlage 9:** Sequentiediagram package Statistics

**Bijlage 10:** RESTFul API Documentation

**Bijlage 11:** Sequentiediagram package Notifications

**Bijlage 12:** Sequentiediagram De Lijn

**Bijlage 13:** Facebook Messenger Integratie