3e Bachelor Informatica
Academiejaar 2015-2016

Faculteit Ingenieurswetenschappen en Architectuur
Valentin Vaerwijckweg 1 - 9000 Gent

Verkeerscentrum

Verslag voor bachelorproef (sprint 1)

Groep 2 Mike BRANTS
Tobias VAN DER PULST
Thomas VANDE WEGHE
Simon VERMEERSCH

Inhoudsopgave

Hoofdstuk 1: Behoefteanalyse

1.1 Beschrijving project

Het Mobiliteitsbedrijf van de stad gent is sinds 2014 bezig met het opzetten van een regionaal verkeerscentrum. Het is de bedoeling dat op termijn het verkeer in de regio constant gemonitord wordt, op semi-automatische basis op normale werkdagen en bemand tijdens piekmomenten en evenementen. Tijdens de week is het de bedoeling dat onverwachte incidenten, calamiteiten of significante verhogingen van de reistijden automatisch gesignaleerd worden aan de verantwoordelijke, die dan de nodige acties kan ondernemen. De gegevens zouden ook constant beschikbaar zijn voor het publiek via een website, sociale media en open data. Op die manier kunnen mensen de beste route en het beste moment kiezen om hun verplaatsingen te maken in de regio.

1.2 Functionaliteiten

1. Ophalen van kwalitatieve en vergelijkbare data bij verschillende bronnen
2. Real-time overzicht van de verkeersdruke op vooraf vastgelegde trajecten
3. Analyse op basis van opgehaalde data
4. Aanbieden van gegevens aan externen via REST API
5. Kwaliteitscontrole van de verkregen data
6. Platform gelinkt met sociale media om snelle communicatie aan te bieden
7. Meldingen genereren wanneer reistijden overschreden worden
8. Bepalen van de oorzaak van een vertraging

1.3 Use Case

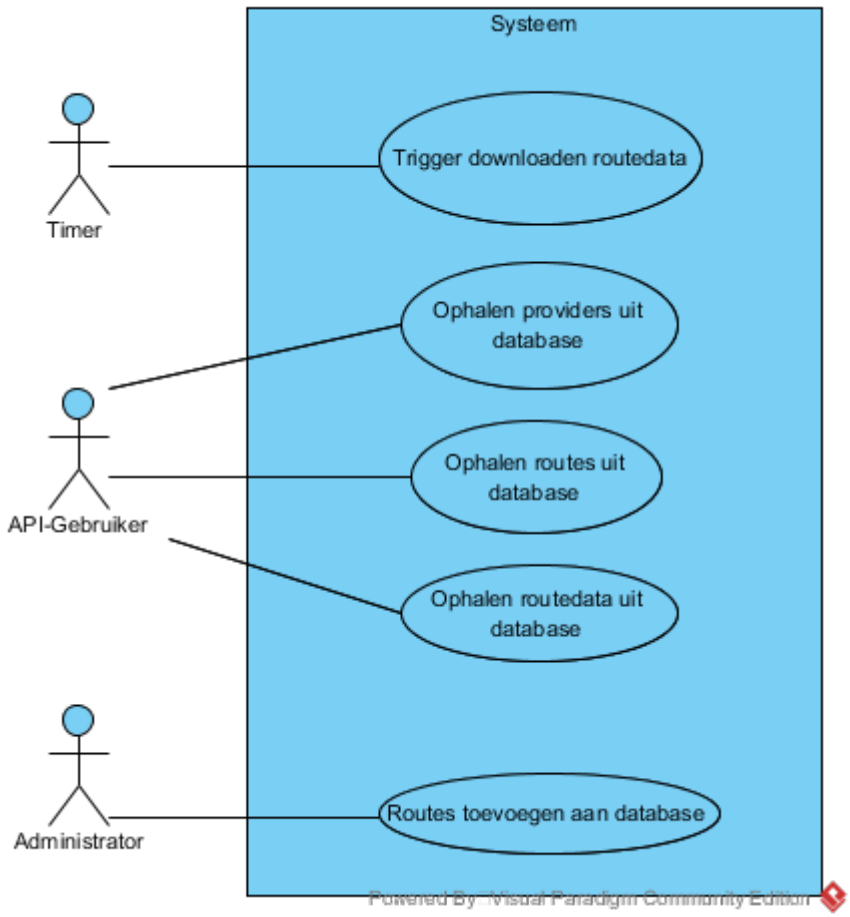

Diagram 1 Use Case

Er zijn drie actoren aan het werk. In de eerste plaats is er een timer, deze zal een trigger sturen naar het programma zodat data afkomstig van de verschillende providers opgehaald wordt. Verder is er nog een API-gebruiker, dit is een persoon die data kan ophalen uit de database gebruik makend van de REST API. Als laatste is er nog een administrator, hij kan routes toevoegen aan de database zodat ook van deze routes data wordt opgehaald.

1.4 Functieanalyse van de omgeving

1. Gebruikers
   (a) Ontwikkelaar
   (b) Adminstrator
   (c) Operator in het verkeerscentrum

2. Doelstellingen
   De doelstellingen representeren de product backlog en ze bevatten de taken die het systeem moet kunnen.
   (a) Basisfunctionaliteit
      i. Data ophalen uit meerdere bronnen
         A. Google Maps
         B. Here
         C. Waze
         D. TomTom
         E. Coyote
      ii. Databank creëren en opvullen met opgehaalde data
   (b) API met verschillende parameters
      i. Periode
      ii. Traject
      iii. Leverancier
      iv. Vertraging
   (c) Dashboard
      i. Grafische opbouw van de GUI
      ii. Grafieken/Tabellen genereren
      iii. Grafische weergave op kaart
      iv. Selecteren parameters gebruik makend van API
   (d) Kwaliteitscontrole
      i. Procedures via MySQL
      ii. Grafieken/Tabellen genereren

Hoofdstuk 2: Ontwerp

2.1 Functioneel ontwerp

Er zijn 2 types gebruikers in het systeem. Enerzijds zijn er de API-gebruikers, zij hebben de mogelijkheid om data uit de API op te vragen en eventueel verder te verwerken. Anderzijds is er een administrator, deze gebruiker kan routes toevoegen aan het systeem.

2.2 Technisch ontwerp (Tobias)

2.2.1 Hardware

Tijdens de ontwikkelingsfase zal beroep gedaan worden op de servers van de UGent.

2.2.2 Paketten

1. Java EE
2. Glassfish

2.3 Software ontwerp

2.3.1 Dataproviders

Google Maps

Om de reistijden die Google Maps heeft verzameld te kunnen raadplegen, moet er gebruik gemaakt worden van de Google Maps Distance Matrix API (https://developers.google.com/maps/documentation/distance-matrix/) . In de URL kunnen verschillende start- en eindpunten worden meegegeven, alsook of het antwoord json dan wel XML moet teruggeven. Hou er wel rekening mee dat er voor elke combinatie van start- en eindpunt die worden meegegeven, een reistijd wordt opgenomen in het resultaat. Als er dus 3 startpunten en 3 eindpunten worden meegegeven, zal het resultaat een 3X3-matrix zijn. Dit komt dan ook overeen met 9 aanvragen naar de API. De limiet per URL is een 10X10-matrix.

Om effectief de actuele reistijden te verkrijgen moeten de parameters

traffic_model en departure_time=now ook zeker in de URL opgenomen worden. Om deze aanvragen te kunnen doen moet er een sleutel aangevraagd worden, die gekoppeld wordt aan het project. Met een gratis sleutel kunnen 2500 elementen per dag opgevraagd worden (het aantal elementen is gelijk aan het aantal startpunten vermenigvuldigd met het aantal eindpunten). Indien dit overschreden wordt zal er per schijf van 1000 extra elementen 0.5 dollar aangerekend worden.

Indien je voor de 34 routes om de 5 minuten de reistijden opvraagt en dit 18u per dag, dan zit je al aan meer dan 7300 aanvragen per dag. Er rekening mee houdend dat eventuele tussenpunten het aantal aanvragen bij deze API nog eens heel sterk doen oplopend, mag duidelijk zijn dat het gratis model niet zal volstaan. Er kan overgeschakeld worden het Google Maps APIs Premium Plan zodat er tot 100 000 aanvragen per dag gedaan kunnen worden.

Here

Here stelt reistijden ter beschikking via zijn Routing API (https://developer.here.com/rest-apis/documentation/routing). In de URL geef je aan of je json of XML als antwoord wilt en kan je een route meegeven door de geocoördinaten in te stellen voor start- en eindpunt en eventuele tussenpunten mee te geven. Verder moet voor de toepassing die hier ontworpen wordt steeds gekozen worden voor de kortste route in plaats van de snelste en moet er rekening gehouden worden met het huidige verkeer. Op deze manier zal steeds actuele info over een vaste route worden teruggeven.

Om de API van Here te kunnen gebruiken moeten er 2 sleutels aangevraagd worden die gekoppeld worden aan de applicatie. De eerste 90 dagen kan dit gratis en mogen er tot 100 000 aanvragen per maand gedaan worden. Als je de 34 routes om de 5 minuten wil opvragen komen je echter al aan meer dan 220 000 aanvragen per maand en zou je sowieso een betalende formule nodig hebben. Om tot 275 000 aanvragen per maand te kunnen doen moet er ingestapt worden in het standaard plan dat 99 euro per maand kost.

2.3.2 Structuur van de data (API)

De REST API bestaat uit 3 categorieën: routes, routedata en providers. Wanneer je hier aanvragen naar doet krijg je gegevens uit de database terug. In de komende secties wordt een korte toelichting gegeven per categorie, gevolg door een voorbeeld van aanvraag en antwoord in JSON-formaat. Er is een online API beschikbaar waarin je meer informatie kan terugvinden, hieronder krijg je alvast een kort overzicht.

Routes

De routes zijn de trajecten waar realtime data van opgeroepen wordt. Er kan gekozen worden om naam, id en/of alle tussenpunten te tonen voor de trajecten, indien je geen parameters meegeeft zullen standaard naam, id en tussenpunten

teruggegeven worden. In de toekomst zal het ook mogelijk zijn om routes toe te voegen via de API.

Patroon
GET ../routes/id/?fields=route.name,route.id,route.geolocations

Voorbeeld
GET ../routes/1,2,3/?fields=route.name,route.id,route.geolocations

---

Voorbeeld


```
[
 {
    "name": "R4 Zelzate",
    "id": 2,
    "geolocations": [
    {
        "latitude": 51.192226,
        "name": "Zelzate",
        "longitude": 3.776342
    },
    {
        "latitude": 51.086447,
        "name": "Gent",
        "longitude": 3.672188
    }
   ]
 }
]
```

RouteData

De routedata is de opgehaalde data per route van de verschillende providers op verschillende tijdstippen. Je kan kiezen voor welke routes je data wil terugkrijgen, tevens zal je begin- en eindpunt kunnen instellen. Dit laatste is nuttig om data in een bepaald interval terug te krijgen. In de parameters heb je ook de mogelijkheid om de provider mee te geven, je kan er op deze manier voor zorgen dat enkel data van bepaalde providers getoond wordt.

Patroon
GET ../routes/id/data/timeStart/timeEnd/?fields=route.name,route.id, route.geolocations&provider=provider.name

Voorbeeld
GET ../routes/1,2,3/data/1456761535931/huidig tijdstip/?fields=route.name,

route.id,route.geolocations&provider=GoogleMaps,Here

Voorbeeld


```
[
 {
    "data": [
      {
       "duration": 753,
       "distance": 14677,
       "provider": "GoogleMaps",
       "timestamp": "1456761535931"
      },
      {
       "duration": 681,
       "distance": 14685,
       "provider": "Here",
       "timestamp": "1456761535931"
      }
    ],
    "name": "R4 Zelzate",
    "id": 2,
    "geolocations": [
     {
        "latitude": 51.192226,
        "name": "Zelzate",
        "longitude": 3.776342
     },
     {
        "latitude": 51.086447,
        "name": "Gent",
        "longitude": 3.672188
     }
    ]
 }
]
```

Providers

Het is mogelijk om via de REST API alle providers op te vragen, dit kan handig zijn om in bijvoorbeeld RouteData te gebruiken als parameter.

Patroon
GET ../providers

Voorbeeld
GET ../providers

```
[
 "Here",
 "GoogleMaps"
]
```

2.3.3 Databank

De database bestaat uit 3 tabellen. RouteData is op termijn de grootste tabel, hierin worden alle opgehaalde gegevens bewaard. In routes staan alle trajecten waarvan data zal worden opgehaald. Deze routes bestaan uit geolocaties die het traject bepalen.

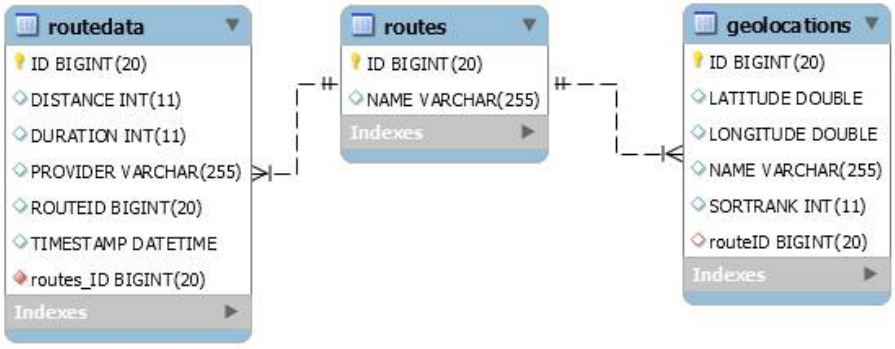

Diagram 2 Databank

2.3.4 Klassendiagram

In dit onderdeel vindt u 3 klassendiagrammen, al kan je het in dit geval ook interfacediagrammen noemen. Om te beginnen is er een diagram voor de basiscomponenten, de meest elementaire klassen in het systeem. Deze klassen vormen ook de database. Hierna komt het diagram van het gegevensbeheer, deze bevat de samenwerking van klassen die data ophalen en verwerken. Als laatste, maar daarom niet minder belangrijk, vindt u de BeanFactory. Dit diagram bevat slechts 1 klasse die de klassen uit diagram 3 zal beheren.

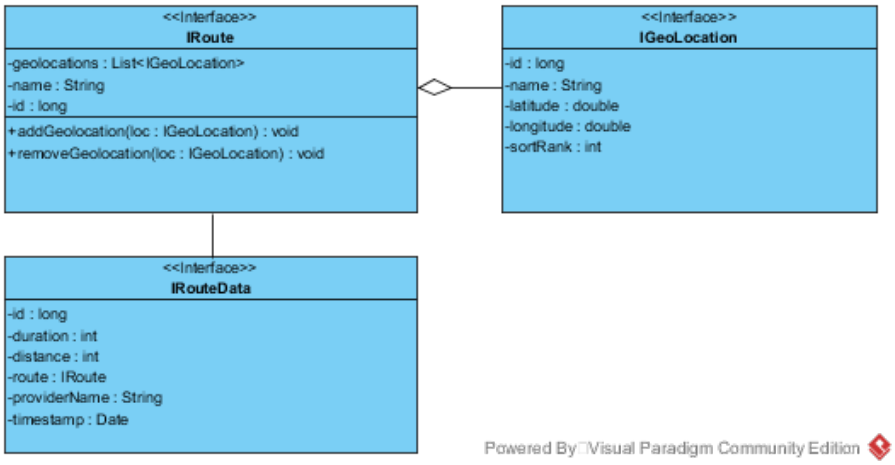

Diagram 3 Basiscomponenten/Database

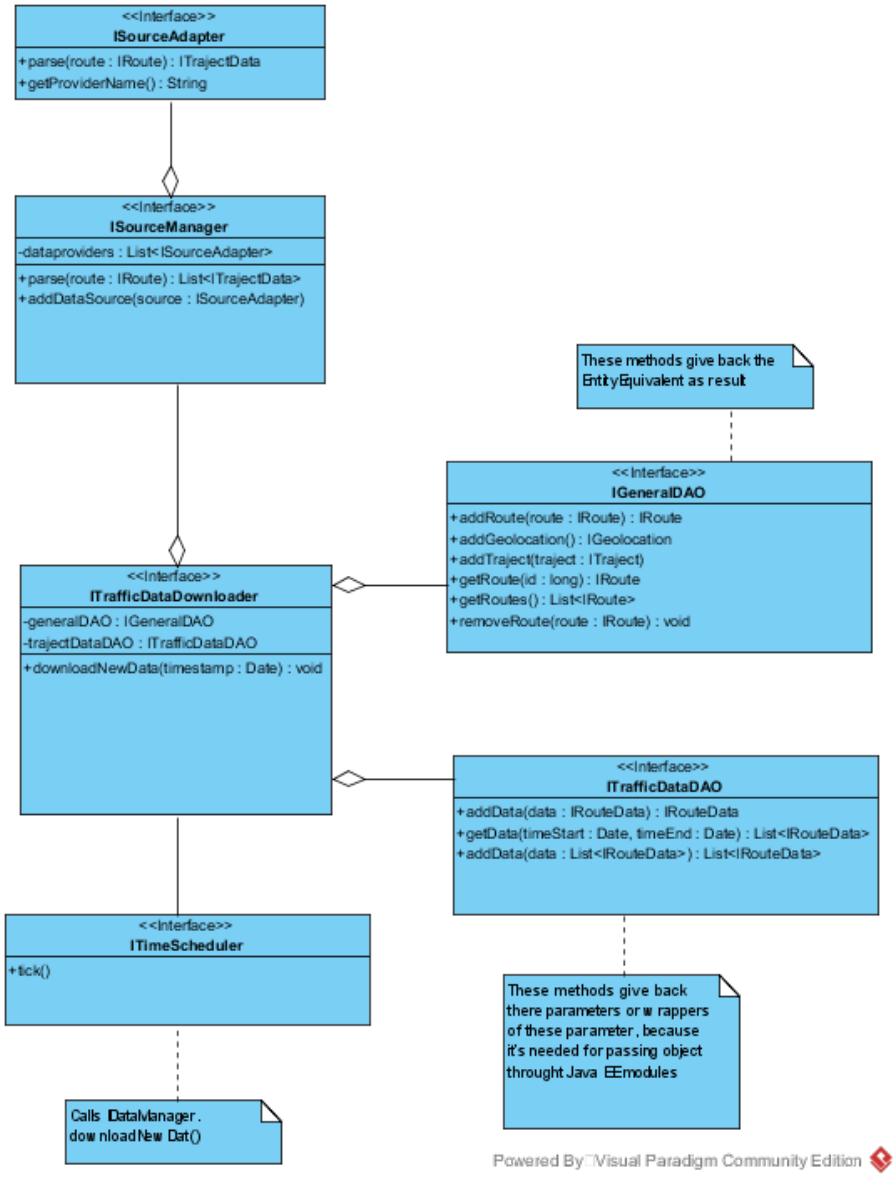

Diagram 4 Gegevensbeheer

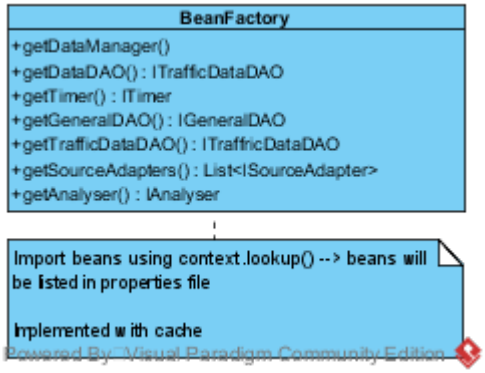

Diagram 5 BeanFactory

2.3.5 Verantwoordelijkheid per klasse

Klasse	**Verantwoordelijkheid**
Route	bevat informatie over een route
GeoLocation	bevat informatie over een locatie
RouteData	bevat verkeersinformatie van één route, één provider en dit op één bepaald moment in de tijd
IgnoredPeriod	vertegenwoordigt periodes die genegeerd moeten woden tijdens het berekenen van gemiddelde reistijd per dag
[Object]Entity	deze klassen vertegenwoordigen bovenstaande objecten zodat ze in de database kunnen opgeslagen worden
HereSourceAdapter	omzetten van data, aangeboden door Here, naar RouteData
GoogleMapsSourceAdapter	omzetten van data, aangeboden door Google Maps, naar RouteData
SourceManager	beheren van adapters
TimerScheduler	automatische triggering voor ophalen nieuwe data
TrafficDataDownloader	beheren van routes en geeft commando om routedata op te halen door
TrafficDataDownstreamAnalyser	data afkomstig van SourceAdapters controleren op correctheid en nadien verdere acties ondernemen indien nodig
BeanFactory	deze klasse zal dependency injection vertegenwoordigen in alle klassen
GeneralDAO	verbinding vormen tussen core en database
TrafficDataDAO	RouteData opslaan in de database

2.3.6 Modules

De volledige applicatie streeft naar de richtlijnen van een Line Of Business applicatie.

Flexibel en Uitbreidbaar

Door gebruik te maken van Java EE, waarin de gehele applicatie in verschillende modules wordt opgedeeld, kunnen nieuwe modules eenvoudig afzonderlijk worden gecreerd en worden toegevoegd.

Onderhoudbaarheid

Dit analysedocument bevat alle nodige informatie over de klasses en hun onderlinge relaties na sprint 1. Op het einde van de ontwikkelingsperiode zal een documentatie worden voorzien met alle nodige informatie voor andere ontwikkelaars die de applicatie zouden willen wijzigen of uitbreiden.

Testbaarheid

De verschillende componenten werden getest gebruik makend van unittests en integratietests.

Late Binding

Java EE biedt de mogelijkheid om een applicatie op te delen in verschillende modules die afzonderlijk van elkaar kunnen worden gecompileerd. Er werden twee DAOs voorzien zodat de algemene data over routes en de verkeersinformatie over de routes kunnen worden opgeslagen in twee verschillende databases. Zo zal na de ontwikkelingsperiode de verkeersinformatie waarschijnlijk worden opgeslagen in een NoSQL-database, omdat de hoeveelheid data enorm groot zal worden.

Parallelle ontwikkeling

Door opdeling in modules, die Java EE aanbiedt, kunnen programmeurs afzonderlijk van elkaar code implementeren.

Losse koppeling van objecten

Modules kunnen eenvoudig worden ontkoppeld en worden vervangen door een andere. De BeanFactory, die de module-objecten aanbiedt, wordt eenvoudig geconfigureerd in een propertiesbestand. Zo kan eenvoudig een nieuwe DAO worden toegevoegd daar de link in het configuratiebestand te wijzigen naar een andere DAO. Zo kan bijvoorbeeld de manier van opslaan van data eenvoudig worden gewijzigd van een SQL-database naar een NoSQL-database.

Crosscutting concerns (Logging)

De manier van logging kan eenvoudig worden gewijzigd door het hierboven beschreven principe van losse koppeling. Voor logging werd ook een module voorzien die kan worden gewijzigd door de link aan te passen in het propertiesbestand bij de BeanFactory.

2.3.7 Gegevensstroomdiagram

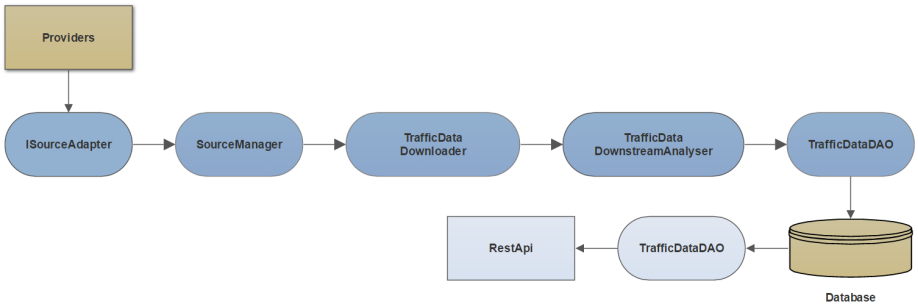

Diagram 6 Gegevensstroomdiagram

Downstream

De data van de verschillende providers wordt opgehaald met behulp van de SourceAdapters, per provider bestaat er een adapterklasse die de ISourceAdapter interface implementeert. De verschillende adapters bevinden zich in de SourceManager-klasse. Vanuit de TrafficDataDownloader wordt data per route aangevraagd, de SourceManager zal deze aanvragen doorsturen aan elke adapter en de ontvangen data per adapter teruggeven aan de TrafficDataDownloader. Vervolgens passeert de data ook nog de TrafficDataDownstreamAnalyser die controleert of de data geldig is en eventueel meldingen genereert. Om af te sluiten wordt de data doorgegeven aan de TrafficDataDAO, deze klasse zorgt ervoor dat de data in de database terechtkomt.

Upstream

De mogelijkheid bestaat om via API-aanvragen data uit de database te halen, dit gebeurt via de TrafficDataDAO die contact heeft met de database.

2.3.8 Bestandsstructuur (Tobias)

Iets over de splitings in beans denk ik dan?

2.3.9 Tests

Databank

De databank werd gedurende het ontwikkelingsproces getest via het principe van integratietesting. De data die in de database werd opgeslagen kwam overeen met de data die gestuurd werd naar de database.

SourceAdapters

De SourceAdapters werden getest aan de hand van unittests en later integratietesting in combinatie met de database. In de unittests werd nagekeken of excepties werden opgegooid bij verkeerde invoer. Het vergelijken van data in onze database met de data die op de sites van de providers staat werd manueel uitgevoerd.

Sortering geolocaties

In één route zitten meerdere geolocaties om te verzekeren dat het gevolgde pad correct is. Om zeker te zijn dat de sortering van de locaties correct gebeurt werd een unittest geschreven.

REST API

De gegevens uit de database worden correct weergegeven via de REST API.

GeoLocation

In de klasse geolocation bestaan 2 variabelen om de coördinaten te bepalen namelijk latitude en longitude. Deze moeten binnen bepaalde grenswaarden liggen, bijgevolg werd hiervoor een exceptie met bijkomende unittest geschreven.