



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

Inhoudsopgave	1
1 Behoeftanalyse	2
1.1 Beschrijving project	2
1.2 Functionaliteiten	2
1.3 Use Case	3
1.4 Functieanalyse van de omgeving	3
2 Ontwerp	5
2.1 Functioneel ontwerp	5
2.2 Technisch ontwerp (Tobias)	5
2.2.1 Hardware	5
2.2.2 Paketten	5
2.3 Software ontwerp	7
2.3.1 Dataproviders	7
2.3.2 Structuur van de data (API)	8
2.3.3 Databank	11
2.3.4 Klassendiagram	11
2.3.5 Verantwoordelijkheid per klasse	13
2.3.6 Modules	14
2.3.7 Gegevensstroomdiagram	15
2.3.8 Bestandsstructuur (Tobias)	15
2.3.9 Tests	16

Hoofdstuk 1: Behoeftanalyse

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

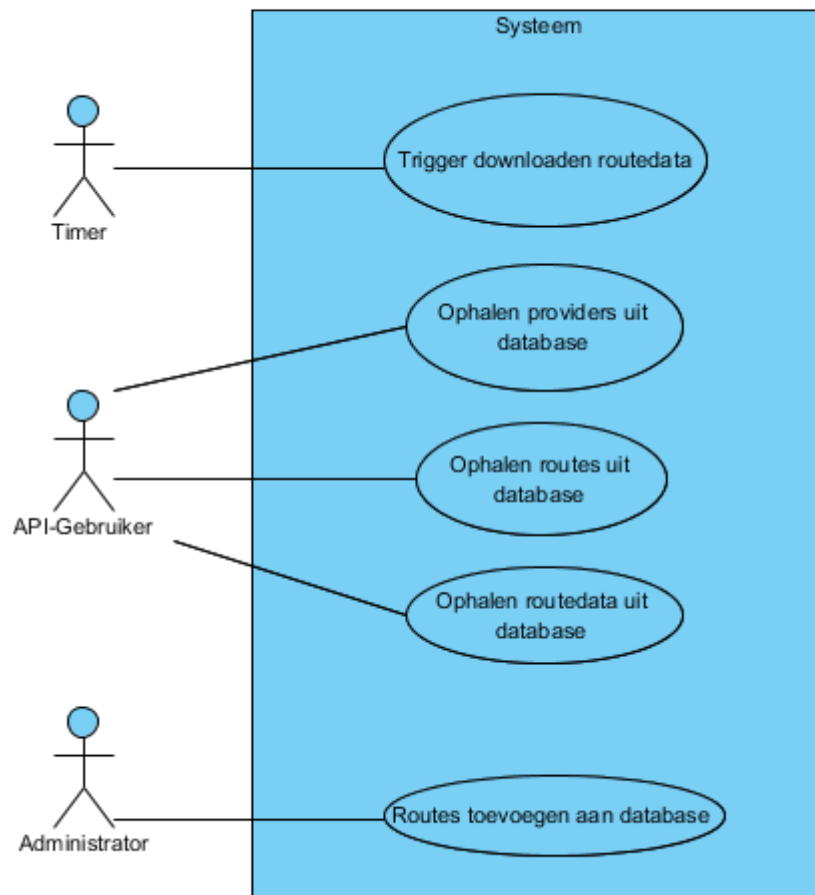


Diagram 1 Use Case

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

Dit project wordt uitgewerkt in Java met behulp van het Java EE (*Enterprise Edition*) *framework*. Dit framework omvat verschillende technologieën die in dit project gebruikt worden.

1. JavaServer Faces (JSF) 2.2
2. Context and Dependency Injection for Java 1.1
3. Enterprise JavaBeans (EJB) 3.2
4. Java Persistence API (JPA) 2.1
5. Java Transaction API (JTA) 1.2
6. Java API for RESTful Web Services (JAX-RS) 2.0

Dit wordt aangevuld met enkele Glassfish voorzieningen.

1. Java Naming and Directory Interface (JNDI)
2. Java Database Connection (JDBC) connection pools

Hieronder zal elk deel uitgebreid behandeld worden.

JSF

Deze technologie laat toe webpaginas te genereren op basis van data in de applicatie. Deze data wordt aangeleverd via *backing beans*. Over de jaren is JSF uitgebreid met Ajax ondersteuning alsook luisteraars bij data wijzigingen. Dit alles laat toe een zeer flexibele, *responsive* interface te maken voor de gebruikers.

Contexts and Dependency Injection

Alle beans verwachten diensten waarvan zijzelf afhangen. Zo wordt bij de Data Access Beans (DAO) gebruik gemaakt van de EntityManager voor interactie met de databank. Bij de TrafficDataDownloader wordt dan weer de context van de applicatieserver verwacht om andere beans op te vragen. Al deze diensten zijn niet de verantwoordelijkheid van deze individuele beans maar van de applicatieserver. Deze laatste gedraagt zich als *injector* en zal alle vereiste *services* (*dependencies*) plaatsen in de beans aan de hand van annotaties en objecttypes.

EJB

JavaBeans zijn door software beheerde modulaire bouwblokken. In deze beans wordt de *business logic* voor een Enterprise Applicatie verwerkt. De grootste kenmerken voor deze beans zijn hun modulariteit, onafhankelijkheid van elkaar en schaalbaarheid.

Modulariteit

Iedere module (JavaBean) in het project is uitwisselbaar met een nieuwe module. Dit is aan te passen in een extern properties bestand. Zo kan op ieder moment bijvoorbeeld een leverancier worden toegevoegd, een database worden vervangen door een andere of een nieuwe *web service* worden toegevoegd.

Onafhankelijkheid

Alle beans zijn onafhankelijk. Alle objecten die voorkomen in meerdere beans (zoals interfaces) zijn gebundeld in een gemeenschappelijke bibliotheek. Hierdoor zullen beans geen enkele invloed ondervinden wanneer een andere bean wijzigt.

Uitbreidbaarheid

De modules zijn niet enkel onafhankelijk, maar hebben eveneens geen vaste relatie met de locatie waar ze werken. Zo kan een *database bean* op een andere server staan dan de *analyser bean*. De enige vereiste hiervoor is dat JNDI van de ene server gelinkt is aan de JNDI van de andere server. De beans zullen hun parameters en teruggeefwaardes steeds serialiseren en doorsturen naar de zogenaamde *remote bean*.

JPA

De Java variant voor Object Relational Mapping (ORM) laat toe de gegevens in een databank rechtstreeks af te beelden op objecten door middel van annotaties. Deze manier van interactie met de databank laat een zeer eenvoudige werking toe. Het zal echter niet de performantie van handmatige SQL-commando's evenaren.

JTA

Deze API start (zonder enige configuratie) steeds een transactie bij het aanroepen van een functie in een *managed bean*. Indien die functie er fout zou opwerpen zal een rollback gebeuren tot de toestand vlak voor de aanroep van de desbetreffende functie. In het project wordt op deze API vertrouwd voor opslag van gegevens in de databank. Bij een fout zal de dataopslag voor dat bepaalde interval niet plaatsvinden. Hierna zal de applicatie echter wel blijven verderwerken.

JAX-RS

Deze API laat toe om services aan te bieden volgens het Representational State Transfer (REST) patroon. In dit project wordt het gebruikt om de API uit te werken.

JNDI

Deze technologie laat toe data of objecten op te vragen via naam. Voor dit project werd de link naar de bronbestanden, de link naar de JDBC Connection Pool en de link naar alle beans opgenomen in JNDI.

JDBC Connection Pools

Een Connection Pool houdt een *cache* van connecties naar een welbepaalde databank bij en maakt deze beschikbaar aan de applicaties van de applicatieserver. Dit alles zorgt voor een hogere efficiëntie want de connecties worden behouden en herbruikt. Hiernaast wordt er ook een hogere veiligheid verondersteld, de connectieparameters zijn namelijk niet langer in de applicatie zelf aanwezig.

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 drie startpunten en drie eindpunten worden meegegeven, zal het resultaat een 3X3-matrix zijn. Dit komt dan ook overeen met negen 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 twee 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 vijf 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 geeft info terug uit de database, hier heb je de keuze uit drie opties: 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 optie, 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. Indien je geen parameters meegeeft zullen naam, id en tussenpunten teruggegeven worden. In de toekomst zal het ook mogelijk zijn om routes toe te voegen via de API.

Vraag

Patroon

GET http://verkeer-2.bp.tiwi.be/api/v2/routes/id

Voorbeeld

GET http://verkeer-2.bp.tiwi.be/api/v2/routes/1,2,3

Antwoord

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. Het is ook mogelijk om in plaats van twee tijdstippen enkel *current* op te geven, in dat geval wordt de actuele informatie getoond.

Vraag

Patroon

GET http://verkeer-2.bp.tiwi.be/api/v2/routes/id/data/timeStart/timeEnd

Voorbeeld

GET http://verkeer-2.bp.tiwi.be/api/v2/routes/1,2,3/data/1456761535931/huidig
tijdstip/

Voorbeeld (Actueel)

GET <http://verkeer-2.bp.tiwi.be/api/v2/routes/1,2,3/data/current/>

Antwoord *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 <http://verkeer-2.bp.tiwi.be/api/v2/providers>

Voorbeeld

GET <http://verkeer-2.bp.tiwi.be/api/v2/providers>

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

Parameters

Er zijn 2 types parameters die je kan meegeven op het einde van je URL: velden en providers. In velden kan je beslissen wat weergegeven moet worden in het antwoord en bij provider kan je ervoor kiezen om enkel data van bepaalde providers weer te geven.

Voorbeeld

?fields=route.name,route.id,route.geolocations&provider=GoogleMaps,Here

2.3.3 Databank

De database bestaat uit drie 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.

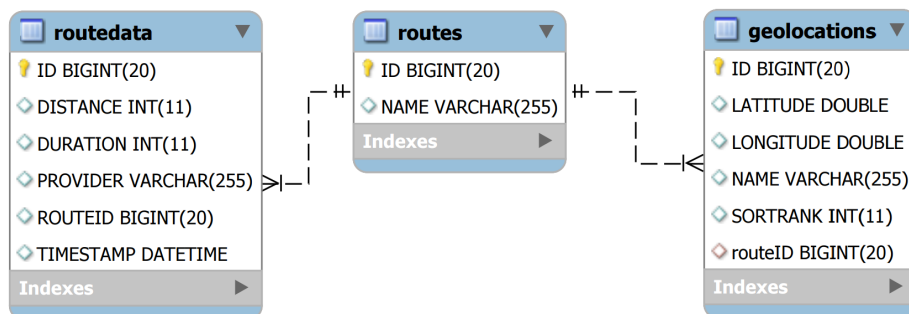


Diagram 2 Databank

2.3.4 Klassendiagram

In dit onderdeel vindt u drie 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 één klasse die de klassen uit diagram 3 zal beheren.

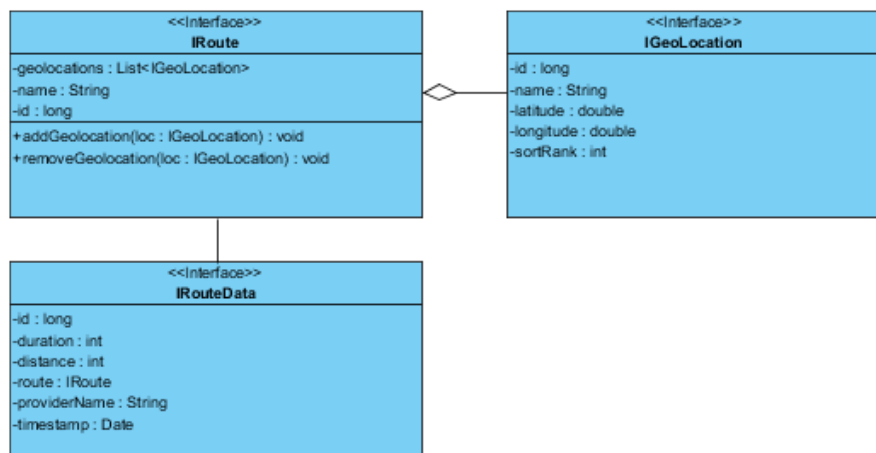


Diagram 3 Basiscomponenten/Database

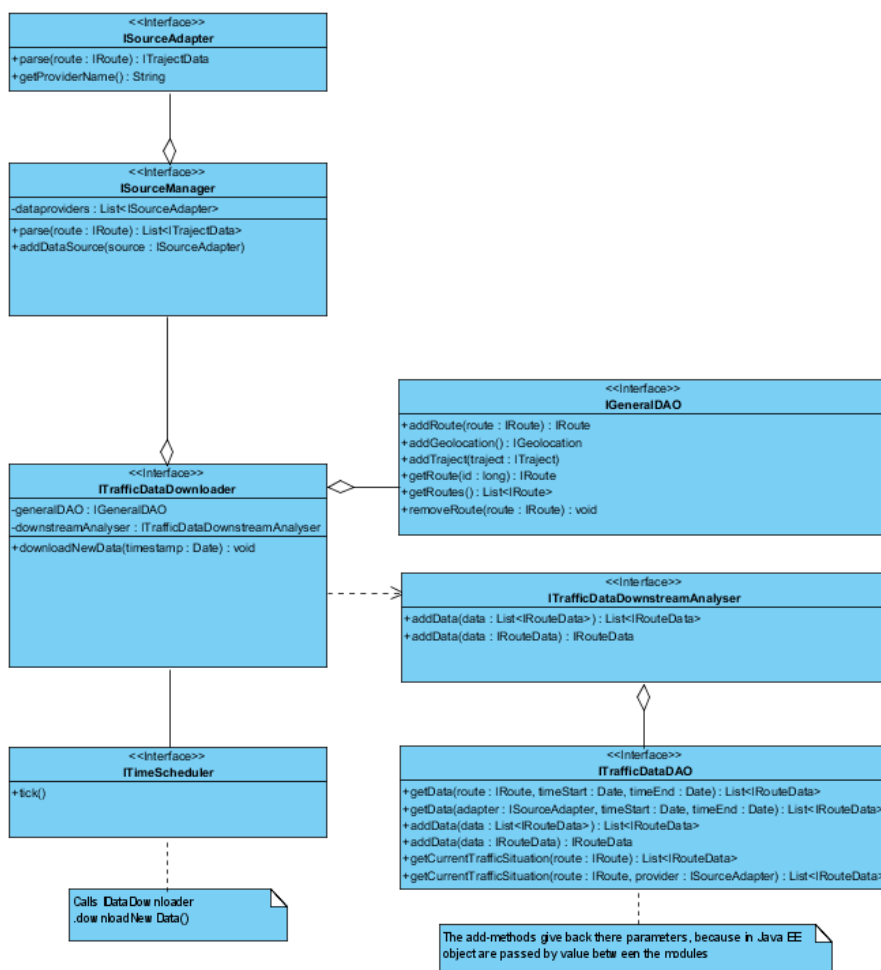


Diagram 4 Gegevensbeheer

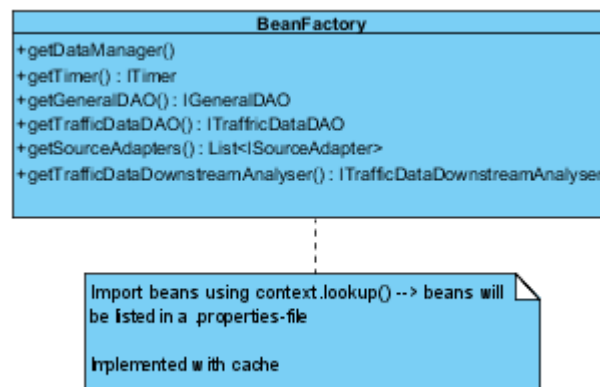


Diagram 5 BeanFactory

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 worden 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 gecreëerd en worden toegevoegd.

Onderhoudbaarheid

Dit analysedocument bevat alle nodige informatie over de klassen 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.

Parallele 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

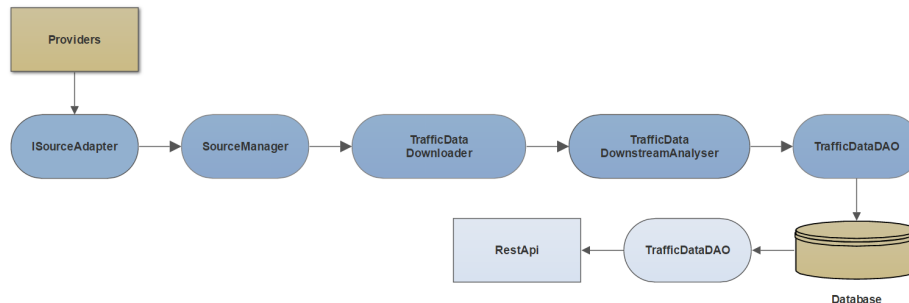


Diagram 6 Gegevensstroomdiagram

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)

Het project is opgedeeld in verschillende beans die we hieronder behandelen.

Logger

Deze bean start tegelijk met de server en maakt logging naar een bestand (log.txt) mogelijk.

TimerScheduler

Deze bean start tegelijk met de server en stuurt triggers volgens het patroon dat opgegeven wordt aan de bean in een *properties*-bestand. Deze timer wordt gecreëerd via Java EE TimerServices.

TrafficDataDownloader

De downloader staat in voor de connectie tussen de adapters en de DAO. Deze bean wordt getriggerd door de TimerScheduler.

TrafficDataDownstreamAnalyser

Deze analyzer staat in voor generatie van meldingen bij opmerkelijke data.

GoogleMapsSourceAdapter/HereSourceAdapter

Providers leveren nieuwe RouteData-objecten aan de applicatie. Deze data wordt verkregen via API-aanroepen.

GeneralDAO

De GeneralDAO houdt de routes bij die door de applicatie in de gaten worden gehouden. De verkregen data uit het programma wordt omhult in GeoLocationEntity en RouteEntity om deze compatibel te maken voor de achterliggende databank.

TrafficDataDAO

De TrafficDataDAO houdt alle data bij van de routes verkregen door de applicatie. De RouteData uit het programma wordt omhult in RouteDataEntity om deze compatibel te maken voor de achterliggende databank.

GeneralDAONoDB/TrafficDataDAONoDB

Dit zijn twee dummy-objecten die gebruikt kunnen worden voor tests zonder de echte databank te gebruiken.

RESTAPI

Deze bean bevat de API-service van het project.

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.