Universiteit Gent

Vakoverschrijdend project

Project Mobiliteitsbedrijf Gent

Uitbreidingsdocument

Groep 7:
Trésor Akimana
Nick De Smedt
Tom Hoet
Dean Parmentier
Simon Scheerlynck
Xavier Seyssens
Jonas Van Wilder
Lennart Vermeir

Lesverantwoordelijken:
Jan Goedgebeur
Annick Van Daele
Prof. Dr. Marko Van Dooren

2015-2016

Inhoudsopgave

1 Een nieuwe bron voor gebeurtenissen

Voorlopig is de enige bron voor gebeurtenissen Waze zoals de opgave het heeft gespecificeerd. Om een nieuwe bron van gebeurtenissen aan te maken van moet een nieuwe klasse aangemaakt worden die de interface *DataSource* in de package *vop.groep7.vop7backend.Datasources* implementeert. Aangezien de package *vop.groep7.vop7backend.Datasources* speciaal aangemaakt werd voor entiteiten die te maken hebben met bronnen van gebeurtenissen is het ook best dat de nieuwe klasse ook in diezelfde package aangemaakt wordt.
Om de interface *DataSource* te kunnen implementeren moet een methode *checkDataSource* aangemaakt worden met de volgende signatuur: *public void checkDataSource()* (zonder return-waarde en zonder argumenten). Deze is de methode die wordt opgeroepen wanneer een databron gecontroleerd moet worden. Het is dan ook in die methode dat de gebeurtenissen opgehaald worden en elk van hen verstuurd wordt naar de *EventController* [1] voor verdere bewerking (opslaan in databank en meldingen sturen naar de genteresseerde gebruikers).
Om het systeem deftig te kunnen laten reageren op een gebeurtenis die net binnen is gekomen is het belangerijk dat er een onderscheid wordt gemaakt tussen gebeurtenissen die het aanmaken van een nieuwe gebeurtenis impliceren of gebeurtenissen die het wijzigen van een al bestaand gebeurtenis impliceren. In de *EventController* mogen ze niet door dezelfde methode afgehandeld worden: de nieuwe gebeurtenissen worden meegegeven aan de methode *modifyAPIEvent(int eventId, APIEvent event)* ($eventId$ is het id van de gebeurtenis die overschreven moet worden) en nieuwe gebeurtenissen worden meegegeven aan de methode *createAPIEvent(APIEvent event)*. Beide methodes maken enkel gebruik van *APIEvent*-objecten waardoor elke gebeurtenis omgevormd moet worden in een *APIEvent*-object.

Indien nodig en/of gewenst kan een nieuwe subklasse aangemaakt worden van de klasse *Event* uit de domeinlaag in de package *package vop.groep7.vop7backend.Models.Domain.Events* om een nieuw type javaclassEvent-objecten te definiëren met een ander gedrag dan degene die al bestaan (nl. *JamEvent* en *AlertEvent*).

Als laatst kan ervoor gekozen worden om het ophalen van de gebeurtenissen met de nieuwe bron als een geplande taak(scheduled task) toe te voegen in het bean-bestand van Spring (bestand $beans.xml$ in de resources-directory van het Spring-project). Daarvoor moet men eerst een instantie van de bron-klasse registreren als een bean en een scheduled-task toevoegen bij de al reeds gedefiniëerde scheduled-task(s) die dan de methode *checkDataSource()* gaat oproepen. Als we stellen dat we onze nieuwe bron-klasse *EventSource* hebben genoemd en we bvb om de 24 uren data moeten controleren als data beschikbaar is dan doen we dat als volgt:

```
...
<bean id="EventSourceConfig" class="vop.groep7.vop7backend.
    Datasources.EventSource" />
...

<task:scheduled-tasks>
        ...
        <task:scheduled ref="EventSourceConfig" method="
            checkDataSource" cron="0 0 0 * * *"></task:
            scheduled>
        ...
</task:scheduled-tasks>
```

Voor het gemak en meer structuur zou de *cron*-eigenschap van de scheduled task in het bestand application.properties in de resources-map mogen gaan. Daar staat ook die van de scheduled task voor Waze gedefinieerd.

[1] De *EventController* is bereikbaar via de klasse *AppConfig* in de package *vop.groep7.vop7backend* met de methode *getEventController()*

In het kort zijn dus de volgende stappen nodig om een nieuwe bron van gebeurtenissen te gebruiken:

1. In de package *vop.groep7.vop7backend.Datasources* een klasse met een default constructor die de interface *DataSource*(in dezelfde package) implementeert aanmaken die de bron gaat voorstellen.
2. Indien gewenst een nieuwe subklasse definiëren van de klasse *Event* uit de domeinlaag in de *package vop.groep7.vop7backend.Models.Domain.Events*-package.
3. De methode *public void checkDataSource()* implementeren. Deze moet:
   (a) de gebeurtenissen afhalen en omzetten naar *APIEvent*-objecten
   (b) de gebeurtenissen één voor één doorgeven aan de *EventController* via de methodes *modifyAPIEvent(int eventId, APIEvent event)* voor degene die een reeds opgeslagen gebeurtenis overschrijven en *createAPIEvent(APIEvent event)* voor nieuwe gebeurtenissen.
4. Als een geplande taak(scheduled task) nodig is die de *checkDataSource*-methode moet oproepen:
   (a) een object van de nieuwe bron-klasse registreren als bean in bean.xml in de resources-map
   (b) een nieuwe scheduled task die de methode van de bean gaat oproepen toevoegen in de lijst van scheduled tasks.

2 Meertalige website

Op dit moment is de website slechts in taal beschikbaar, namelijk Nederlands. Maar er gebeuren hier en daar al vertalingen in de achtergrond vermits we sommige data in het Engels binnenkrijgen. Hierdoor moeten er slechts een paar simpele uitbreidingen gebeuren om de website in een nieuwe taal te verkrijgen.

De volgende stappen moeten echter ondernomen worden:

1. Voor alle teksten een vertaling voorzien in een bijhoordend json bestand.
2. Alle teksten in het volgende formaat onderbrengen {{"tekst die vertaald moet worden" | translate}}.
3. Knop toevoegen in het bestand index.html die toelaat om tussen de talen te wisselen.

Een andere mogelijkheid voor de laatste optie is de taal automatisch af te leiden uit de instellingen van de browser en/of de locatie van ip-adres.

3 Toevoeging interessante punten manipulatie in de app

Eerst moet er een *Activity* en een *View* bijgemaakt worden. De *Activity* zit vast aan de respectievelijk view en moet bovendien overerven van de *BaseActivity* klasse. Deze gaat dan elke actie van de gebruiker op het scherm interpreteren en zo de gepaste acties doen via de bijhorende *POICommunication*. Deze moet ook zelf aangemaakt worden met overerving van *Communication* en implementeerd voor elke gewenste handeling een functie. De zelfgedefinieerde *POICommunication* moet aan een singleton eigenschap voldoen. De singleton eigenschap stipuleert dat in de hele werking van de applicatie (en op elk moment in de tijd) slechts 1 en enkel 1 instantie van dit object mag bestaan en bevraagd worden.

4 Toevoeging Trajecten manipulatie in de app

Eerst moet er een *Activity* en een *View* bijgemaakt worden. De *Activity* zit vast aan de respectievelijk view en moet bovendien overerven van de *BaseActivity* klasse. Deze gaat dan elke actie van de gebruiker op het scherm interpreteren en zo de gepaste acties doen via de bijhorende *TrajectCommunication*. Deze moet ook zelf aangemaakt worden met overerving van *Communication* en implementeerd voor elke gewenste handeling een functie. De zelfgedefinieerde *TrajectCommunication* moet aan een singleton eigenschap voldoen. De singleton eigenschap stipuleert dat in de hele werking van de applicatie (en op elk moment in de tijd) slechts 1 en enkel 1 instantie van dit object mag bestaan en bevraagd worden.