FACULTEIT
WETENSCHAPPEN

VAKOVERSCHRIJDEND PROJECT

UITBREIDINGEN

Thomas De Pontieu, Billie Devolder, Jarre Knockaert, Sam Persoon, Jorg Van Rentergem, Tomas Van Roose, Freek Verschelden

20 mei 2017

UNIVERSITEIT
GENT

1 Vloten samenvoegen

De eerste uitbreiding die we zullen bespreken is het samenvloegen van vloten. Indien 2 vestigingen van een bedrijf fuseren en de vloten moeten worden samengevoegd, zou het handig zijn dit met een simpele druk op een knop te kunnen doen. In principe moet er voor deze uitbreiding niets worden veranderd in de backend. De frontend kan alle voertuigen van een bepaalde vloot opvragen en dan via een PUT request de vloot veranderen van alle voertuigen in de vloot.

De aanpassingen staan hier zeker in verhouding tot de uitbreiding. In de backend en database moet er namelijk niets aangepast worden.

2 Ondersteunen van boten

Misschien wil Solvas, in een verre toekomst, zich ook gaan bezighouden met het verzekeren van boten. Hier nemen we aan dat het verzekeren van boten gebeurt op een gelijkaardige manier.

In het domeinmodel zou er een nieuwe klasse, `WaterCraft`, moeten toegevoegd worden. Deze klasse zou een boot voorstellen en sterk lijken op de huidige klasse `Vehicle`. Deze 2 klassen zouden grote gelijkenissen vertonen dus zou hier best een gemeenschappelijke superklasse voor gemaakt worden. In dit document zal deze superklasse `AbstractVehicle` worden genoemd. De klasse `Fleet` zou dan ook een collectie `AbstractVehicle` objecten moeten hebben in plaats van `Vehicle` objecten.

In de andere lagen moeten volgende zaken worden toegevoegd/veranderd:

1. Er moet een hibernate mapping gemaakt worden voor de nieuwe klasse.
2. Er moet een nieuw `DataAccessObject` aangemaakt worden. Deze moet overerven van de abstracte klasse `ProductionDAO`. In deze klasse moeten geen nieuwe methodes worden geïmplementeerd.
3. Het nieuwe `DataAccessObject` moet toegevoegd worden aan de `DAOProvider`.
4. Er moet een nieuwe `Controller` worden aangemaakt die overerft van `AbstractController`. In de nieuwe `Controller` moet de isOwner function worden geïmplementeerd.
5. De nieuwe `Controller` moet worden toegevoegd aan de `ControllerManager`.
6. Er moet een `RESTModel` worden aangemaakt voor de nieuwe klasse. Deze erft over van `RESTAbstractModel`. In de klasse moet de constructor en de translate methode worden geïmplementeerd.
7. Er moet een nieuwe `RESTController` worden aangemaakt die overerft van `RESTAbstractController`. Hier moet de getController methode worden geïmplementeerd een een methode die alle objecten, al dan niet gefilterd, teruggeeft.

We denken dat de aanpassingen in verhouding staan tot de uitbreiding. Het meeste werk zou kruipen in het doorgeven van de waarden van de velden en deze te setten in de objecten. Dit is repetitief werk maar niet iets dat gemakkelijk te veralgemenen is. De meeste veranderingen in het domeinmodel zijn eenmalig. Indien er nog een ander soort voertuig zou bijkomen met andere velden, zou er enkel een klasse moeten worden toegevoegd die overerft van `AbstractVehicle`.

3 E-mails verzenden naar klanten bij bepaalde acties

Voor het versturen van e-mails zouden we gebruik maken van het observer patroon. Als tussenklasse zouden we een `EventBroker` gebruiken die events ontvangt en ze doorstuurt naar geregistreerde listeners. Dit is een singleton klasse. Onze controllers zouden bij elke actie een event genereren en doorgeven aan de `EventBroker`. Hierdoor blijven de controller module en e-mail module volledig ontkoppeld. De controllers kunnen eenvoudig aangepast geworden. Enkel `AbstractController` zal moeten worden aangepast. De e-mail module zou een listener worden. Aan de hand van de informatie die wordt doorgestuurd bij een event kan er dan beslist worden of er al dan niet een e-mail moet verstuurd worden.

4 Algemene log tonen

Er zou in de applicatie ook een algemene log kunnen getoond worden. Bij het ontwerpen van het logsysteem hebben we hier rekening mee gehouden dus kan dit relatief gemakkelijk worden toegevoegd. Bij de `LogEntryController` moet een methode worden aangemaakt die alle logs ophaalt van de `LogEntryDAO`. Bij `RESTLogEntryController` moet een methode worden aangemaakt die de methode van `LogEntryController` oproept.

5 Ondersteunen van andere soorten correcties

Indien we andere soorten correcties willen ondersteunen, moet er bij het veranderen van een `VehicleInsurance`, naast het object zelf, ook de datum waarop de verandering moet plaatsvinden worden meegegeven. Voor de API call wordt er best een wrapper object aangemaakt die de datum en de `RESTVehicleInsurance` bevat. Bij de put methode van `VehicleInsuranceController` zal nu ook de datum moeten worden meegegeven. De code die moet gebruikt worden voor het genereren van correcties kan grotendeels hergebruikt worden van de andere soorten correcties (verzekering toevoegen en verwijderen).

6 Exporteren van voertuigen

Voor deze uitbreiding moeten we de bestaande code niet aanpassen maar gewoon code toevoegen. Ten eerste hebben we hier een nieuwe API call voor nodig. In de methode die de API call implementeert, zullen de voertuigen moeten worden opgehaald aan de hand van de `VehicleController`. We zullen ook een nieuwe module moeten implementeren die een xls bestand kan aanmaken. Deze module zal ergens een methode nodig hebben waar de lijst van voertuigen aan moet worden meegegeven en die een bestand teruggeeft. De methode die de API call implementeert, zal dan deze nieuwe methode oproepen en het bestand teruggeven. Hier staan de aanpassingen in verhouding tot de uitbreiding: aan de bestaande code moet niets aangepast worden.