

# Webservice api

Andreas De Lille

February 26, 2014

## 1 Inhoud

In dit document som ik eerst een aantal mogelijkheden op voor de webservice. Bij elke mogelijkheid worden de voor- en nadelen vermeld. De derde methode zal uiteindelijk uitgewerkt worden. Deze methode wordt dus in meer detail uitgeschreven.

## 2 Eerste uitwerking

Deze eerste uitwerking heeft voor elke test een apart functie. Zo is bijvoorbeeld voor de resultaten van ping op te vragen een functie `getPing()` die de ping van een testbed terug geeft.

### 2.1 Functies

Hierbij zijn er veel verschillende functies nodig.

- `getPing`
- `getDuration`
- `getTestbedName`
- `getTestDescription`
- ..

### 2.2 Besluit

Hierbij heeft men per functie een pagina die specifiek voor dat deel informatie kan teruggeven. Het nadeel is wel dat er veel verschillende functies zijn die bijna hetzelfde doen. Verder hebben veel van die functies dezelfde parameters. het zou handiger zijn om deze functie meer generisch op te stellen. Een gebruiker vraagt Ook moeten uiteindelijke gebruikers altijd een andere functie aanroepen. Dit is dus allicht niet het beste idee. Bijgevolg zal ik hier niet verder op ingaan.

## 3 Tweede uitwerking

De tweede mogelijkheid is om meer generisch te werken. Met een functie getResult() om het resultaat van een test op te halen. We moeten deze functie natuurlijk wel meegeven van welke test we een resultaat willen. Daarnaast zou ik nog 2 functies voor zien getTest en getTestbed die info geven over respectievelijk een test en een testbed.

### 3.1 Functies

Deze uitwerking heeft maar 3 functies

- get data van een test opvragen.
- getTest Test opvragen.
- getTestbed Testbed opvragen.

#### 3.1.1 Besluit


We zien hier nog altijd een verderling van functies die eigenlijk niet noodzakelijk is. Waarom zou het opvragen van een commando van een test anders moeten verlopen dan het opvragen van een resultaat? Uiteindelijk is dat ook data die verbonden is met een test. De derde methode werkt dit verder uit.

## 4 Derde uitwerking

Een derde mogelijkheid is een generische functie. Daaraan moeten we meegeven wat we willen krijgen. Deze methode lijkt de beste uitwerking te zijn en zal dus gebruikt worden.

### 4.1 Functies

Er zijn een aantal crud acties. Deze komen overeen met de verbs van de request method in de http-header. Bijgevolg moet er dus ook geen get/put/create/delete vermeld worden in de url om te melden wat men wil doen. Aangezien we over http werken, wordt dit doorgegeven via het request method veld van de http header. Het spreekt uiteraard voor zich dat get dient om opvragingen te doen uit de databank. Put om dingen aan te passen, create om dingen aan te maken in de databank en delete om dingen te verwijderen. Het belangrijkste zullen de get en put zijn. Waarbij get geraadpleegd wordt vanuit jFed om te kijken wat de status van een testbed is. Put zal gebruikt worden om vanuit de testbeds zelf de database actueel te houden.

- get
- put
- create 
- delete

## 4.2 Parameters

Hier worden de parameters besproken parameternamen tussen () zijn optioneel.

- objectType ( test — testbed ) Deze parameter geeft aan of men informatie wil opvragen over een test of over een testbed.
- objectName : ( 'testnaam' — 'testbednaam' ) Dit geeft aan om welk test/testbed het gaat.
- property : ( result — config — list ) Rest zal het volledige resultaat van een test terug geven. Config zal de configuratie van de test terug geven zoals de frequentie en het commando. List daarin tegen zal een lijst met resultaten teruggeven. Dit is nodig als we een detail overzicht van een test willen op een testbed.
- testbedid : ( 'testbedid' ) Dit kan een string of een getal zijn om het testbed te identificeren. Dit is niet altijd nodig, als men bijvoorbeeld de beschrijving van een test wil opvragen moet men geen testbedid meegeven omdat deze verondersteld wordt van dezelfde te zijn op verschillende testbeds. Hetzelfde kan gezegd worden voor de frequentie, tenzij we willen dat een ping test niet op elk testbed met dezelfde frequentie uitgevoerd wordt.
- (from) : Als from of till is opgegeven wordt er normaliter een gemiddelde teruggegeven. Als enkel from opgegeven is zal het gemiddelde berekend worden vanaf de fromwaarde tot het huidige waarde. Behalve als we een history opvragen dan wordt er geen gemiddelde berekend. In plaats daarvan wordt een array van waardes terug gegeven zodat men snel kan zien wanneer een fout nog is opgetreden.
- (till) : gelijkaardig aan hierboven, maar nu tot een bepaalde waarde. Als beide opgegeven zijn zal het gemiddelde van die periode gegeven worden.

### 4.3 Return waarden

Er kan gekozen worden tussen xml en json. Het verschil tussen xml en json is dat xml een boomstructuur beschrijft, waar json meer een hashmap voorstelt. Xml is meer geschikt voor grote geavanceerde structuren. Aangezien het hier om eenvoudige monitoring informatie gaat, is xml overkill. Verder dient te worden opgemerkt dat een xml-notatie van een object langer is dan de overeenkomstige json. Ook is het parsen van json eenvoudiger. Bijgevolg zal hier dus geen xml, maar json gebruikt worden. Door de mvc opbouw die ik hier zal gebruiken, zal het echter niet moeilijk zijn om xml functionaliteiten te voorzien. Dit kan achteraf eventueel toegevoegd worden.

Merk op dat de huidige situatie niet voorziet in een webservice. De website maakt rechtstreeks verbinding met de databank. Het tussenvoegen van een webservice kan overhead veroorzaken. Door calls te bundelen, zodat een call direct alle resultaten van een testbed teruggeeft, wordt deze overhead tot een minimum beperkt. Het is ook mogelijk om de webservice rechtstreeks te laten antwoorden in html, waarbij de webservice tegelijk ook de site host.

Sommige opvragen geven langere antwoorden terug. Zo is het opvragen van de status van een stitching test multivalued. Een stitching test zal proberen om connectie te maken met een testbed. Vervolgens zal hij aanmelden en een testnetwerk opzetten. Eenmaal het testnetwerk opgezet is zal hij proberen om te pingen tussen de verschillende nodes. Aangezien een stitching test bij elke stap kan mislopen, zal er in de status duidelijk vermeld moeten zijn welke stappen gelukt zijn en welke niet. Zo kan het zijn dat een stitching test niet kan aanmelden, wat niet verwonderlijk is als hij bv. ook niet kan pingen naar een testbed.

## 4.4 Voorbeelden

Hieronder zal ik een aantal voorbeelden van calls en bijhorende antwoorden geven. De GET of PUT duiden op de http-methode die gebruikt wordt. Deze methode is ingevuld in de http header en zal dus niet zichtbaar zijn in de link.

### 4.4.1 geen parameters

Hiervoor zijn er een aantal mogelijkheden.

1. De testbed/call (hieronder besproken) het meeste gebruikt zal worden om de site op te bouwen, zou ik default het antwoord van deze call terugsturen. Dat houdt in dat er een overzicht gegeven wordt per testbed van de status van elke test.
2. Er kan gebruikersinfo teruggegeven worden. Deze info zou dan beschrijven hoe de api werkt en welke calls er gedaan kunnen worden. Het voordeel hiervan is dat de documentatie geïntegreerd is met de api.
3. Deze call kan ook als ongeldig beschouwd worden, eventueel via een foutboodschap. Toch lijken bovenstaande instellingen deze call beter in te vullen.

De optimale keuze zal vooral afhangen of we de site regelen door de webservice die html antwoord. Indien we dat doen lijkt oplossing 1 het beste. De site is dan gewoon een algemeen overzicht. Indien niet is de documentatie de beste keuze. Uiteraard kunnen beide voorzien worden door bijvoorbeeld een objectType help toe te voegen of een objectType site om respectievelijk de documenten of het overzicht weer te geven.

#### 4.4.2 testbed

Deze call zal voor elk testbed een overzicht van alle resultaten van elke test teruggeven. Doordat deze call snel de actuele stand van zaken teruggeeft, is het makkelijk om via website deze info in een keer op te halen. Dit verkleint de overhead die komt van meerdere calls te moeten afwerken. Indien een test niet bestaat op een testbed zal deze waarde ingevuld worden met een nullpointer. Er zou ook een lijst van alle testbeds teruggegeven kunnen worden. Een voorbeeld van een return waarde zou zijn:

```
[ {  
  "Testbed": "virtualWall2",  
  "results" : [  
    "TestResult" : {  
      "testName" : "Ping",  
      "TestbedName" : "VirtualWall",  
      "Value" : "56",  
      "Status" : "Good",  
      "Last Check" : "2014-02-26 07:35:03"  
    },  
    "TestResult" : {  
      "testName" : "LoginTest",  
      "testbedName" : "testbed1",  
      "Value" : "",  
      "Status" : "Good",  
      "Last Check" : "2014-02-26 06:57:34"  
    },  
    ...  
  ],  
  }, {  
    "Testbed": "testBed",  
    ...  
  }  
]
```

Merk op dat Testbed zowel in de lijst al in Test vermeldt wordt. Dit is om een wildgroei aan klassen te vermijden. Een TestResult zal altijd de naam van het testbed en de naam van de test bevatten.

#### 4.4.3 test

Deze call zal, gelijkaardig aan hierboven, voor elke test de status van elk testbed geven. Het verschil is dat de resultaten nu gegroepeerd zullen zitten per test. Indien een test niet aangemaakt is voor dat testbed (bijvoorbeeld een testbed ondersteunt geen aggregate manager version 3), zal een null pointer teruggegeven worden.

```
[ {
  "Test": "ping",
  "Results" : [
    "testResult" : {
      "testbedName" : "VirtualWall",
      "testname" : "ping",
      "Value" : "56",
      "Status" : "Good",
      "Last Check" : "2014-02-26 07:35:03"
    },
    "testResult" : {
      "testbedName" : "testbed1",
      "testname" : "ping",
      "Value" : "63",
      "Status" : "Good",
      "Last Check" : "2014-02-26 06:57:34"
    },
    ...
  ],
  "Test" : "LoginTest",
  ...
},
...
]
```

#### 4.4.4 test/Ping

- GET test/ping

Deze aanvraag zal voor elk testbed de resultaten van de ping test teruggeven. Waarbij de status good, fail of warn kan zijn. Warn zal duiden op een hoge ping waarde. Het antwoord in json zal bijvoorbeeld zijn:

```
[ {  
  [  
    "TestResult" : {  
      "TestbedName" : "virtualwall",  
      "TestName" : "ping",  
      "Value": "56",  
      "status" : "Good"  
      "Last Check" : "2014-02-26 07:35:03"  
    },  
    "TestResult" : {  
      "TestbedName" : "testbed1",  
      "TestName" , "ping",  
      "Value": "56",  
      "status" : "Good"  
      "Last Check" : "2014-02-26 07:35:03"  
    }  
  ]  
}
```

- GET test/ping/value

Hier zijn 2 mogelijkheden (zie ook opmerkingen):

1. Het stricte minimum teruggeven, waarbij er dus mogelijk extra klassen nodig zijn in jFed. De output zou er dan zo uitzien:

```
[ {  
  "Testbed": "virtualWall2",  
  "Value": "475"  
},  
{  
  "Testbed": "testBed",  
  "Value": "56"  
} ,  
...  
]
```



2. Value beschouwen als alle waarden behorend bij een specifieke pingtest. Hierbij wordt gedacht aan de waarde, de status, en het testbed waarop de ping uitgevoerd wordt. Er worden dan testresults teruggegeven. De uitvoer zou er dan zo uitzien:


```
[ {
  [
    "TestResult" : {
      "TestbedName" : "virtualwall",
      "TestName" : "ping",
      "Value": "56",
      "status" : "Good",
      "Last Check" : "2014-02-26 07:35:03"
    }, "TestResult" : {
      "TestbedName" : "testbed1",
      "TestName" , "ping",
      "Value": "56",
      "status" : "Good",
      "Last Check" : "2014-02-26 07:35:03"
    }
  ]
}
```

Een tweede aanroep zou dan de config zijn. Deze zou de configuratie van een test weergeven. Hierbij zullen ook alle properties van de config gebundelt en in een aanvraag verwerkt.

```
[ {
  "TestConfig" : {
    "testname" : "ping",
    "Frequency" : "00:05:00", //weergave frequency nog niet definitief
    "command" : "ping $host",
    ...
  }
} ]
```

Een derde aanroep zou list zijn die een lijst met waarden teruggeeft, eventueel tussen 2 opgegeven data.

Het voordeel hiervan is dat we direct alle waarden hebben. De overeenkomstige klasse is dan ook direct volledig ingevuld. Het nadeel is dat we iets meer data doorsturen dan eigenlijk gevraagd.

 Tweede optie is hier duidelijk beter. Het is beter om direct alle belangrijke waarden te bundelen. De overhead om alles te bundelen is veel kleiner dan de overhead die zou veroorzaakt worden door alle calls apart uit te voeren. Hieronder zal ik dus verder werken in de veronderstelling dat de tweede methode gebruikt wordt. **Daarom zal ik vanaf nu het objectType value vervangen door result.**

- GET test/ping/result/virtualwall

Deze aanvraag geeft de huidige ping terug van de virtualwall. Met huidige ping wordt de laatst gemeten ping bedoeld. Een antwoord ziet er zo uit:

```
[ {
  "TestResult" : {
    "TestbedName" : "testbed1",
    "TestName" , "ping",
    "Value": "56",
    "status" : "Good"
    "Last Check" : "2014-02-26 07:35:03"
  }
} ]
```

- GET test/ping/result/virtualwall/2013-02-05 18:00:00/2014-02-25 00:00:00 Deze aanvraag geeft de gemiddelde ping terug vanaf de opgegeven datum tot de tweede opgegeven datum. Indien de tweede datum niet opgegeven is, wordt het gemiddelde berekend tot de huidige datum. We kunnen op 2 manieren aanduiden dat het om een gemiddelde gaat.

1. via de naam.

```
[ {
  "TestResult" : {
    "TestbedName" : "testbed1",
    "TestName" , "avgping",
    "Value": "56"
  }
} ]
```

2. Via een boolean.

```
[ {
  "TestResult" : {
    "TestbedName" : "testbed1",
    "TestName" , "avgping",
    "Value": "56",
    "isAverage" : true;
  }
} ]
```

De tweede mogelijkheid vermijdt het onnodig parsen van strings. Dit kan eventueel gecombineerd worden met een default-false waarde. Als isAverage dan niet opgegeven is, kan aangenomen worden dat het om een enkele waarde gaat.

- GET test/stitching/result/virtualwall Deze test geeft een uitgebreider antwoord. Deze test bestaat uit meerdere subtests. Voor elke van deze subtests moet er dus een antwoord gegeven worden. Hiervoor zou ik een lijst van testresults teruggeven. Het antwoord zou er dan zo uitzien:

```
[ {
  "StitchingTestResult" : {
    "testName" : "Stitching",
    "TestbedName" : "VirtualWall",
    "Last Check" : "2014-02-26 07:35:03",
    "TestResults" : [{
      "value" : "...",
      "status" : "good"
    },{
      "value" : "...",
      "status" : "fail"
    }, ....
  ]}
}
```

StitchingTestResult en testResult zullen een gemeenschappelijke bovenliggende interface hebben.

## 4.5 Opmerkingen

- Wordt een type test (bijvoorbeeld een ping test) altijd met dezelfde frequentie uitgevoerd, of moet deze kunnen verschillend van testbed tot testbed? Indien deze niet moet verschillend kunnen we de frequentie bij de test opslaan. Als dit wel moet kunnen moeten we de frequentie per testbed bijhouden wat aanpassingen vereist. Dit komt omdat we dan per testbed een frequentie moeten opslaan en niet meer per test.
- Zoals eerder vermeld, is het ook mogelijk om de webservice en website te bundelen. Door te detecteren of de aanvraag van een browser komt of niet. Indien de aanvraag van een browser komt kan dan html terug gegeven worden. Dit kan bijvoorbeeld door als men rechtstreeks naar de webservice surft het overzicht weer te geven. Indien parameters meegegeven worden en het is een browser wordt in html een antwoord gegeven. Indien er geen browser gedetecteerd wordt, zal in json geantwoord worden.
- Het de parameter objectType die weergeeft dat het om een test of testbed gaat kan eventueel weggelaten worden. Al lijkt mij dit niet aangeraden. Het is namelijk duidelijker om daar een onderscheid te maken of men opvragingen wil doen over een test of een testbed.

- Merk op dat de status telkens opgeslagen wordt in de databank. Voor een simpele pingtest is dit eigenlijk overbodig. Er kan ook op cliënt-side beslist worden dat een ping van 200ms overeen komt met een Warn status. Me zou dus denken dat de status overbodig is. Er zijn echter test die geen value hebben zoals ene login test. Deze test is gelukt of niet. Deze informatie zit dus in de status. Vandaar dat er met 2 velden gewerkt wordt. Men zou deze velden evenwel kunnen samensteken door bv negatieve waarden te gebruiken voor een foute status. Al zorgt dit voor overbodige complexiteit. Verder is het moeilijker om consequent te blijven. Als de status volledig op server-side bepaald wordt, is deze overal dezelfde bij dezelfde waarde.

#### 4.6 Besluit

De laatste uitwerking lijkt mij de meest logische. Het is zeer intuïtief om de ping van de virtualwall op te halen met het commando: `test/ping/value/virtualWall/` of als we de eerste parameter weglaten `ping/value/virtualWall`.

### 5 Conclusie

Er zijn 3 uitwerkingen. De eerste heeft per test een aparte functie. De tweede is een hybride uitwerking waarbij het opvragen van testresultaten al gegroepeerd wordt. De derde is volledig generisch. De derde lijkt mij het eenvoudigste omdat gebruikers dan geen onderscheid van `getTest`, `getTestbed` en `getResult` moeten onthouden. Verder kan er dan eenvoudig gelinkt worden `ping/frequency` geeft dan de frequentie weer terwijl `ping/value` de waardes bevat. Op deze manier kunnen we onze webservice beschouwen als een verzameling objecten waarvan we waarden opvragen. De eerste parameter `objectType` kan behouden worden om te vermijden dat de webservice onoverzichtelijk wordt door de generische opbouw.