whitehorses

member of the human network

WHITEBOOK

RESTful API Modeling Language (RAML) als API ontwerptool

Februari 2018

Auteur:
Peter Holtland
INTEGRATIE SPECIALIST

Inleiding

Wanneer we denken aan een taal die een interface beschrijft voor een API (Application Programmable Interface) denken we snel aan WSDL of WADL. Deze twee talen worden voornamelijk gebruikt door ontwikkelaars. Deze talen zijn niet makkelijk leesbaar en dus niet echt geschikt voor, bijvoorbeeld, informatieanalisten. RAML biedt mogelijk een alternatief. RAML staat voor Restful Api Modeling Language.

De bedoeling van RAML is het bieden van een open, eenvoudige en bondige specificatie voor het beschrijven van API's. RAML heeft faciliteiten die gestructureerde bestanden en overerving ondersteunen. Deze functies staan centraal in dit artikel. REST of beter RESTful, staat voor 'Representational State Transfer' en is een architectuur stijl met een zestal basisprincipes die bepalen of een api RESTful is. Wikipedia geeft hier een heldere omschrijving welke principes dat zijn.

Is RAML bruikbaar als ontwerptaal voor non-technici om een beschrijving te geven van hoe een REST applicatie interface er uit moet zien? Aan de hand van een eenvoudig voorbeeld illustreren we het gebruik van RAML als API ontwerptool. Dit artikel beschrijft de opbouw van een eenvoudige API in RAML en is bedoeld als een voorbeeld van het gebruik van RAML om een eenvoudige API te ontwerpen en het gebruik van enkele functies van RAML te illustreren.

API-voorbeeld

De functionaliteit die we als voorbeeld gebruiken, is een API voor het registreren van fouten in een errorlog om foutmonitoring te ondersteunen. Deze API heeft de volgende specificaties:

- andere API's moeten de API kunnen aanroepen om een fout te registreren;
- fouten hebben een bekend patroon en schema;
- ondersteuning voor het ophalen van een lijst van fouten;
- zoekfuncties binnen een lijst;
- ophalen van een specifieke fout met behulp van een ID.

We beginnen met een rechttoe rechtaan manier om te beschrijven wat we willen bereiken. REST API-specificaties kunnen behoorlijk tekst-intensief zijn, met veel herhaling en veel boilerplate code (developer-slang voor soort code die in een of andere vorm veel voorkomt). Later zullen we code netter maken door gebruik te maken van eenvoudige modulariteit die RAML aanbiedt.

Errorlogging is standaard in elke ontwikkeling. Deze API is een soort utility service waarvan andere services gebruik kunnen maken.Voor de logging hebben we een POST-methode nodig om een fout te loggen en een GET-methode om de gelogde fouten op te halen. Elke afzonderlijke fout kan opgehaald worden met de naam van het log en ID van de fout. De RAML-code voor deze specificatie staat hieronder.

```
#%RAML 1.0
#  Basis raml met get and post.
title: ErrorLogger API
version: 1.0
baseUri: http://errorlogger.test.org/{version}
/errorLog/{Name}:
  displayName: An ErrorLog
  description: A logger is a collection of errorlogs. User provides name.
Post to create, Get to retrieve
  get:
  description: Get a list of errors in this log
  post:
    description: Create an entry into ErrorLog.
  /{ErrorId}:
   get:
     description: Get a log entry by Id.
```

De baseUri geeft het domein en versienummer. URI-parameters worden aangeduid met accolades. De logger heeft een URI relatief ten opzichte van de basis van "/ errorLog/ {Name}", verder zijn er de normale HTTP calls GET en POST voor het aanmaken en ophalen van de gegevens. Ook is er een ErrorId resource die als template meegegeven kan worden in de relative-URI met zijn eigen HTTP-get.

Schema

De standaard POST heeft nog geen enkele restrictie wat betreft de inhoud. RAML biedt aanvullende opties om deze informatie te beschrijven: schema's en voorbeelden. RAML staat toe dat de beschrijving van schema's in zowel Json als XML wordt gedaan. Schema's kunnen inline worden gespecificeerd, inline in de RAML-header of opgenomen via een include. Omdat schema's vrij groot kunnen zijn, is het gebruikelijk om externe schema's op te nemen. Inline schema's geven mindere leesbaarheid van de code.

```
#%RAML 1.0

title: Errorlogger API
version: 1.0
baseUri: http://errorlogger.test.org/{version}
schemas:
  - errorLog: !include errorLog.json
    errorLogLijst: !include errorLogLijst.json
```

Dit voorbeeld bevat een Json-schemabeschrijving van een extern bestand. RAML kan zowel lokale bestanden als remote schema's bevatten. Resources kunnen ook beschreven worden met behulp van example tags in plaats van schema's. Dit wordt geïllustreerd in het volgende codefragment met behulp van de example tag. Een example is eenvoudigweg een blok code dat de definitie weergeeft. Deze examples worden ook meteen in een eventuele mockservice gebruikt. Twee vliegen in 1 klap, verheldering van de definitie en geen extra werk bij het opzetten van een mockservice. Hieronder volgt een mogelijke uitwerking van de content op de POST berichten met gebruikelijke responses:

```
/ErrorLog/{Name}:
  displayName: An ErrorLog
description: Get a list of errors in this log
post:
    description: Create a new errorLog.
    body:
      application/json:
        example: |
                { "name": "Validation",
                  "apiName": "PostCodeCheck",
                  "Tijd": "2017-01-01T16:56:54",
                  "apiURI": "http://test.org/PostcodeCheck/99999",
                  "errorData": {
                    "code": "VAL001",
                    "omschrijving": " Wrong format "
```

```
                }
            }
    responses:
      201:
        description: A new errorLog was created.
        headers:
          location:
            type: string
            required: true
            example: /errorLog/PostcodeCheck/00001
        body: null
```

Het codevoorbeeld toont de RAML-beschrijving van een POST en de data die de foutinformatie bevat. Het response element bevat header informatie en heeft een lege body. De header bevat de relative/URI voor het ophalen van de aangemaakte log. Andere standaard responsen zijn bv HTTP/404 en HTTP/500. Het belangrijke aspect voor onze API is dat gebruikers een 201 antwoord kunnen verwachten, samen met een header met de relatieve URL van het nieuw gemaakte log. Als de gebruiker de errorlog opnieuw wil inzien, kunnen ze de opgegeven URL gebruiken. De GET-methode retourneert een 404 (als de log nog niet is aangemaakt) of een 201-response met errordata. Zie de code hierna.

```
get:
    description: Get a list of errors.
    responses:
      404:
        description: Unknown Logger
      200:
        description: Returns a list Errors.
        body:
          application/json:
            example:
              [
                { "id":"00001",
                  "name": "Validation",
                  "apiName": "PostCodeCheck",
                  "Time": "2017-01-01T16:56:54",
                  "apiURI": "http://test.org/PostcodeCheck/99999",
                  "errorData": {
                    "code`: "VAL001",
                    "omschrijving": "Wrong format"
                  }
                }
```

```
    { "id":"00002",
      "name": "Security",
      "apiName": "PostCodeCheck",
      "Time": "2017-01-01T16:12:41",
      "apiURI": "http://test.org/PostcodeCheck/99999",
      "errorData": {
        "code": "SEC001",
        "Omschrijving": "Unauthorised"
      }
    }
  ]
```

We zien nu eigenlijk al dat de RAML-code enigszins lastig te lezen is. We gaan resource types gebruiken om de base code op te schonen.

Resource types

Er is een design pattern voor collections waarbij een POST samen met de resourcelocatie een 201-respons geeft en een GET een lijst met logs retourneert. We kunnen dit pattern dus isoleren in een resource type dat hergebruikt en eventueel onderdeel kan worden van andere resources.

```
resourceTypes:
  - collection:
      post:
        responses:
          201:
            headers:
              location:
                description: A new errorLog was created.
                type: string
                required: true
                example: /errorLog/PostCodeCheck/99999
      get:

/errorLog/{Name}:
  type: collection
  displayName: An ErrorLog
  description: Get a list of errors in this log
  post:
    description: Create an ErrorLog
```

```
  body:
    application/json:
     [
              { "id":"00001",
                "name": "Validation",
                "apiName": "PostCodeCheck",
                "Time": "2017-01-01T16:56:54",
                "apiURI": "http://test.org/PostcodeCheck/99999",
                "errorData": {
                  "code": "VAL001",
                  "omschrijving": "Wrong format"
                }
              }

get:
  description: Get a list of errors.
  responses:
    200:
      body:
        application/json:
          example:
              { "id":"00001",
                "name": "Validation",
                "apiName": "PostCodeCheck",
                "Time": "2017-01-01T16:56:54",
                "apiURI": "http://test.org/PostcodeCheck/99999",
                "errorData": {
                  "code": "val-001",
                  "errormsg": "wrong format"
                }
              }
```

Traits

Het zou fijn zijn als we kunnen zoeken in de ErrorLog, of in ieder geval informatie kunnen ophalen die we daadwerkelijk willen hebben. We kunnen hiervoor filters aanmaken. Hiervoor gaan we twee verschillende methodes introduceren. De bySoureAPI en de byGroupName parameters. De eerste filtert op de API naam van de eigenaar van de error, de tweede op het soort fout dat gelogd is.

```
traits:
  - bySourceAPI:
    description:  parameter for retrieving errorLogs originating from the
specified API.
      queryParameters:
        apiName:
          description: Name of the source API
          type: string
          example: PostCodeCheck
  -byGroupName:
   description: retrieve ErrorLogs belong to group across all sources
     errorGroupName:
       description: name of the errorgroup.
          type: string
          example: Security
Hier onder is de syntax vermeld, deze moet in de definitie van het
resourceType komen te staan:
get:
     is: [bySourceAPI, byGroupName]
```

Conclusie

Dit was een korte rondleiding door een RAML-specificatie van een zeer eenvoudige API. De nadruk lag op het onderzoeken van functies in RAML die codehergebruik voor API-beschrijvingen ondersteunt. RAML dwingt niet tot een nette notatie. Er zijn geen 'best practices' die inherent zijn aan de taal of de tooling. RAML biedt geen oplossing voor de implementatie van een API. RAML is een eenvoudig te lezen specificatietaal die de structuur beschrijft, pattern based design promoot en hergebruik voorstaat. Door de versimpeling van de structuur is het eenvoudiger geworden om API's te modelleren in de ontwerpfase. Wellicht kan RAML de communicatie tussen ontwerper en ontwikkelaar verhelderen. Documentatie en tutorials voor RAML inclusief de volledige RAML-specificatie zijn beschikbaar op de raml.org-website.