

# RAML - **R**ESTful **A**PI **M**odeling **L**anguage

---

Laurent Prévost  
WEBS

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

- When you are designing and implementing a REST API, you are most often doing it for **third-party developers**:
  - Think about Twitter, Instagram or Amazon exposing services to external developers.
  - Think about an enterprise (e.g. car manufacturer) exposing services to business partners (e.g. suppliers, subcontractors, distributors).
- The documentation of your API is the first thing that third-party developers (your **customers**) will see. You want to **seduce** them.
- The documentation of your API will have a big impact on its **learnability** and **ease of use**.
- **Best practices** and **tools** have emerged. **Evaluate and apply them!**

# RAML by their authors

---



(Mulesoft but not only - OpenSource - Apache License v2)

**RESTful API Modeling Language (RAML) is a simple and succinct way of describing practically-RESTful APIs. It encourages reuse, enables discovery and pattern-sharing, and aims for merit-based emergence of best practices. The goal is to help our current API ecosystem by solving immediate problems and then encourage ever-better API patterns. RAML is built on broadly-used standards such as YAML and JSON and is a non-proprietary, vendor-neutral open spec.**

# What it looks like

```
1  ##RAML 0.8
2
3  title: World Music API
4  baseUri: http://example.api.com/{version}
5  version: v1
6  traits:
7    - paged:
8      queryParameters:
9        pages:
10         description: The number of pages to return
11         type: number
12    - secured: !include http://raml-example.com/secured.yml
13  /songs:
14    is: [ paged, secured ]
15    get:
16      queryParameters:
17        genre:
18         description: filter the songs by genre
19    post:
20    /{songId}:
21      get:
22        responses:
23          200:
24            body:
25              application/json:
26                schema: |
27                  { "$schema": "http://json-schema.org/schema",
28                    "type": "object",
29                    "description": "A canonical song",
30                    "properties": {
31                      "title": { "type": "string" },
32                      "artist": { "type": "string" }
33                    },
34                    "required": [ "title", "artist" ]
35                  }
36              application/xml:
37            delete:
38              description: |
39                This method will *delete* an **individual song**
```

# What it looks like

## Preamble

```
1  ##RAML 0.8
2
3  title: World Music API
4  baseUrl: http://example.api.com/{version}
5  version: v1
6  traits:
7    - paged:
8      queryParameters:
9        pages:
10         description: The number of pages to return
11         type: number
12    - secured: !include http://raml-example.com/secured.yml
13 /songs:
14   is: [ paged, secured ]
15   get:
16     queryParameters:
17       genre:
18         description: filter the songs by genre
19   post:
20     /{songId}:
21       get:
22         responses:
23           200:
24             body:
25               application/json:
26                 schema: |
27                   { "$schema": "http://json-schema.org/schema",
28                     "type": "object",
29                     "description": "A canonical song",
30                     "properties": {
31                       "title": { "type": "string" },
32                       "artist": { "type": "string" }
33                     },
34                     "required": [ "title", "artist" ]
35                   }
36               application/xml:
37             delete:
38               description: |
39                 This method will *delete* an **individual song**
```

# What it looks like

Preamble

Traits

(reusability)

```
1  ##RAML 0.8
2
3  title: World Music API
4  baseUrl: http://example.api.com/{version}
5  version: v1
6  traits:
7    - paged:
8        queryParameters:
9          pages:
10             description: The number of pages to return
11             type: number
12    - secured: !include http://raml-example.com/secured.yml
13 /songs:
14   is: [ paged, secured ]
15   get:
16     queryParameters:
17       genre:
18         description: filter the songs by genre
19   post:
20     /{songId}:
21       get:
22         responses:
23           200:
24             body:
25               application/json:
26                 schema: |
27                   { "$schema": "http://json-schema.org/schema",
28                     "type": "object",
29                     "description": "A canonical song",
30                     "properties": {
31                       "title": { "type": "string" },
32                       "artist": { "type": "string" }
33                     },
34                     "required": [ "title", "artist" ]
35                   }
36               application/xml:
37             delete:
38               description: |
39                 This method will *delete* an **individual song**
```

# What it looks like

Preamble

Traits

(reusability)

Resources

```
1  ##RAML 0.8
2
3  title: World Music API
4  baseUrl: http://example.api.com/{version}
5  version: v1
6  traits:
7    - paged:
8      queryParameters:
9        pages:
10         description: The number of pages to return
11         type: number
12    - secured: !include http://raml-example.com/secured.yml
13 /songs:
14   is: [ paged, secured ]
15   get:
16     queryParameters:
17       genre:
18         description: filter the songs by genre
19   post:
20     /{songId}:
21       get:
22         responses:
23           200:
24             body:
25               application/json:
26                 schema: |
27                   { "$schema": "http://json-schema.org/schema",
28                     "type": "object",
29                     "description": "A canonical song",
30                     "properties": {
31                       "title": { "type": "string" },
32                       "artist": { "type": "string" }
33                     },
34                     "required": [ "title", "artist" ]
35                   }
36               application/xml:
37             delete:
38               description: |
39                 This method will *delete* an **individual song**
```

# What it looks like

Preamble

Traits

(reusability)

Resources

Responses

(different format)

```
1  ##RAML 0.8
2
3  title: World Music API
4  baseUrl: http://example.api.com/{version}
5  version: v1
6  traits:
7    - paged:
8      queryParameters:
9        pages:
10         description: The number of pages to return
11         type: number
12    - secured: !include http://raml-example.com/secured.yml
13 /songs:
14   is: [ paged, secured ]
15   get:
16     queryParameters:
17       genre:
18         description: filter the songs by genre
19   post:
20     /{songId}:
21       get:
22         responses:
23           200:
24             body:
25               application/json:
26                 schema: |
27                   { "$schema": "http://json-schema.org/schema",
28                     "type": "object",
29                     "description": "A canonical song",
30                     "properties": {
31                       "title": { "type": "string" },
32                       "artist": { "type": "string" }
33                     },
34                     "required": [ "title", "artist" ]
35                   }
36               application/xml:
37             delete:
38               description: |
39                 This method will *delete* an **individual song**
```



# What it looks like

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

Preamble

Traits

(reusability)

Resources

Responses

(different format)

```
1  ##RAML 0.8
2
3  title: World Music API
4  baseUri: http://example.api.com/{version}
5  version: v1
6  traits:
7    - paged:
8      queryParameters:
9        pages:
10         description: The number of pages to return
11         type: number
12    - secured: !include http://raml-example.com/secured.yml
13 /songs:
14   is: [ paged, secured ]
15   get:
16     queryParameters:
17       genre:
18         description: filter the songs by genre
19   post:
20     /{songId}:
21       get:
22         responses:
23           200:
24             body:
25               application/json:
26                 schema: |
27                   { "$schema": "http://json-schema.org/schema",
28                     "type": "object",
29                     "description": "A canonical song",
30                     "properties": {
31                       "title": { "type": "string" },
32                       "artist": { "type": "string" }
33                     },
34                     "required": [ "title", "artist" ]
35                   }
36               application/xml:
37             delete:
38               description: |
39                 This method will *delete* an **individual song**
```

Query params

# What it looks like

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

Preamble

Traits

(reusability)

Resources

Responses

(different format)

```
1  ##RAML 0.8
2
3  title: World Music API
4  baseUri: http://example.api.com/{version}
5  version: v1
6  traits:
7    - paged:
8      queryParameters:
9        pages:
10         description: The number of pages to return
11         type: number
12    - secured: !include http://raml-example.com/secured.yml
13 /songs:
14   is: [ paged, secured ]
15   get:
16     queryParameters:
17       genre:
18         description: filter the songs by genre
19   post:
20     /{songId}:
21       get:
22         responses:
23           200:
24             body:
25               application/json:
26                 schema: |
27                   { "$schema": "http://json-schema.org/schema",
28                     "type": "object",
29                     "description": "A canonical song",
30                     "properties": {
31                       "title": { "type": "string" },
32                       "artist": { "type": "string" }
33                     },
34                     "required": [ "title", "artist" ]
35                   }
36               application/xml:
37             delete:
38               description: |
39                 This method will *delete* an **individual song**
```

Query params

Markdown

- 
- RAML is pretty straightforward to use:
    - You describe the list of resources managed by your application, document the support HTTP verbs, enlist the query parameters, etc.
    - Most of modern IDE and text editors supports YAML syntax highlighting. As RAML is based on YAML.
    - If you use **Sublime Text**, you can take advantage of an extension that provides **syntax highlighting** specifically designed for RAML.
    - See **RAML 100 tutorial** (<http://raml.org/docs.html>)
  - RAML has advanced features that can make your specifications less verbose (by abstracting and reusing common elements):
    - includes
    - resource types and schemas
    - traits
    - See **RAML 200 tutorial** (<http://raml.org/docs-200.html>)

# RAML Alternatives



- Swagger (Reverb - Open Source - Apache License v2 )
  - Swagger is a simple yet powerful representation of your RESTful API. With the largest ecosystem of API tooling on the planet, thousands of developers are supporting Swagger in almost every modern programming language and deployment environment. With a Swagger-enabled API, you get interactive documentation, client SDK generation and discoverability.



- API Blueprint (APIary - Open Source - MIT)
  - Connecting the dots in API development
    - Web API Language - Pure Markdown - Designed for Humans - Understandable by Machines - Powerful Tooling - Easy Lifecycle

API Doc Home Blog API Reference

## Welcome

## Smart City

## Citizen Engagement

Welcome on board! You are about to discover the wonderful world where people are participating in the city life everyday.

Each citizen is able to submit any issue he discovers in the city and then a staff member can take care of this issue to solve it.

You will discover on this website the marvelous [API documentation](#) and the [epic stories](#)

[API Doc](#) [Home](#) [Blog](#) [API Reference](#)

## Overview

This pages contains general documentation about the API. Use the links on the right to navigate to specific resources.

### Content-type

The API uses the JSON format. Unless specified otherwise, all requests and response should have the `Content-Type: application/json` header.

### HTTP verbs

The API uses the standard HTTP verbs to perform CRUD operations (**C**reate, **R**etrieve, **U**ppdate, **D**eleate) on resources, following standard RESTful API practices.

Find below a quick summary of how HTTP verbs are used in the API:

Verb	Description
<code>GET</code>	Used for retrieving a resource or a collection of resources.
<code>POST</code>	Used for creating a new resource or performing a non-CRUD operation on a resource.
<code>PUT</code>	Used to perform a full update of a resource (replacing the resource by the JSON data provided in the request).
<code>DELETE</code>	Used for deleting resources.

`HEAD` and `PATCH` are not currently used.

## Resources

[/users](#)

[/data](#)

## List of resources

Endpoint: <https://localhost/api/>

/users

User resource.

/users

POST

GET

/users/{id}

DELETE

PUT

GET

Back to top

/data

Data resource.

/data/populate

POST

Back to top

## Resources

[/users](#)

[/data](#)

**Endpoint:** https://localhost/api/

**/users**  
User resource.

**/users**

GET  
Retrive the list of the users

POST  
Create a new user

**/users/{id}**

GET  
Retrieve the details of a specific user

PUT  
Update the details of a specific user

DELETE  
Delete a specific user

Back to top



# apidoc-seed

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

Endpoint: <https://localhost/api/>

**/users**

User resource.

**/users**

**GET**  
Retrive the list of the users

**POST**  
Create a new user

**/users/{id}**

**GET**  
Retrieve the details of a specific user

**PUT**  
Update the details of a specific user

**DELETE**  
Delete a specific user

**GET** **/users**

Description Request Response

HTTP status code **200**

Body

Type: application/json

Example:

```
[
  {
    "id": "54d8ae163fd30364605c81a4",
    "firstname": "Romain",
    "lastname": "Terieur",
    "phone": "+6624582",
    "roles": [
      "citizen"
    ]
  },
  {
    "id": "54d8ae163fd30364605c81a9",
    "firstname": "Romain",
    "lastname": "Terieur",
    "phone": "+8211332",
    "roles": [
      "citizen",
      "staff"
    ]
  },
  ...
]
```

**PUT** **/users/{id}**

Description Request Response

URI Parameters

- id string

Headers

- Authentication header (string)  
Use this header to specify the authentication.

roles required: **staff**

Example: **x-user-id 54d8ae183fd30364605c8401**

Body

Type: application/json

Example:

```
{
  "firstname": "Romain",
  "lastname": "Terieur",
  "phone": "+6624582",
  "roles": [
    "citizen"
  ]
}
```

- 
- **apidoc-seed** is an open source tool, which makes it easy to generate a complete HTML site for documenting your REST API.
  - To use the tool, **clone the GitHub repo** (<https://github.com/lotaris/apidoc-seed>) and follow the readme instructions.
    - **edit/create jade templates and markdown documents** to provide documentation for your service (general service information, usage guides, support information, etc.).
    - **edit/create RAML files to document your REST APIs**. Depending on the complexity of your APIs, you can **split** the documentation into several files.
  - The tool supports a notion of “**private**” API elements. This is used if your API has resources, methods or parameters that you don't want to publicly expose yet (note that this is documentation level only, nothing will prevent a user to send a request!).

# Individual work

---

- Clone the **repo**, build and test it in your browser.
  - <https://github.com/lotaris/apidoc-seed>
- Follow the instructions in the **RAML 100 tutorial** (<http://raml.org/docs.html>), but do it in the `/src/api/raml/index.raml` file (keep a copy of the original file).
- Modify the **Welcome page** in the documentation site (when you deliver your project, I expect to find a description of your platform here).
- Add an article in the blog that describe, if any, your difficulties to setup the project.
- Customize the **page footer**.

# Resources

---

- [http://control.laneworks.net/admin/project/jobs/api-design/white-paper.php?Project\\_ID=17](http://control.laneworks.net/admin/project/jobs/api-design/white-paper.php?Project_ID=17)
- <http://swagger.io/>
- <https://apibluetooth.org/>
- <http://raml.org/licensing.html>
- [https://github.com/SoftEng-HEIGVD/Teaching-HEIGVD-CM\\_WEBS-2015-Labo-Doc](https://github.com/SoftEng-HEIGVD/Teaching-HEIGVD-CM_WEBS-2015-Labo-Doc)
- <https://github.com/lotaris/apidoc-seed>