Practical Implementation of 10 Rules for Writing REST APIs

Jiri Hradil, Vilém Sklenak
University of Economics in Prague
Faculty of Informatics and Statistics
jiri@hradil.cz, sklenak@vse.cz

DOI: 10.20470/jsi.v8i1.290

Abstract: *This paper shows a practical implementation of "10 Rules for Writing REST APIs introduced in the article" (Hradil, 2016). The application is done in Invoice Home (Wikilane, 2016), an invoicing web application for small business and entrepreneurs available world-wide. The API is implemented in JSON hypermedia format (ECMA International, 2016) and with Ruby on Rails framework (Hansson, 2016). The main purpose of the API is to allow connection of Invoice Home with external systems and offer Invoice Home data in simpler format compared to the current HTML format of the full-stack web application. The paper could be also used as a basic template or pattern for any other implementation of the JSON API in any web-based application.*

Keywords: API, REST, JSON, rules, recommendation, Invoice Home, pattern, template, web application.

1. Introduction

Most current software applications use Web as a default presentation layer. The original purpose of the Web for information sharing, accessibility, collaboration and access all the data from any device connected to Internet is obvious, as well as a need to interconnect different systems and send data between them. Due to many systems, application domains, and user's needs, data is different but the mechanism how to expose and access it could be standardized via API (Application Program Interface). This way, author of a API can offer any data in way that others can understand and use in their own systems.

API as itself can be designed in many ways and technologies but in the context of web applications we need to work over HTTP protocol and under a mechanism which is easy to implement, maintain and standardized. This is where REST comes. REST (Representational State Transfer) is a set of design constraints for hypermedia-distributed systems (Fielding, 2000). API can be think as a one of these "distributed systems". So, we have API and the standard for the API (REST) but we also need a format of the data which will be used in API.

As has been mentioned (Richardson and others., 2013, pp. 719; Verborgh and others., 2015, pp. 239), the standard for data description and transmission comes with two main and popular flavors – XML and JSON. Both formats have some advantages as well as limitations – for example XML is better structured but difficult to write/parse and JSON's missing data semantics but the format is more efficient and shorter. For simplicity, we use JSON data format.

For now, we have discussed three main terms/technologies: API, REST and JSON. With these, we can implement API for *Invoice Home* following the "*10 Rules for Writing REST APIs"* (Hradil, 2016).

2. Summary of 10 Rules for Writing REST APIs

The "*10 Rules for Writing REST APIs*" has been first introduced in (Hradil, 2016). The rules have been collected and written based on the searches by (Verborgh and others, 2015; Richardson and others, 2013; Robillard a Chhetri, 2015; Myers and others, 2010; Businge and others, 2015; Rama, 2015) and on practical experience of the author of this paper. The rules have been described as follows.

Table. 1: 10 Rules for Writing REST APIs

Rule Id	Rule Description	Rule Resource
API-1	Use a standard for medium and format of transmitted data (XML or JSON for example).	(Verborgh and others, 2015; Richardson and others, 2013).
API-2	Use a vocabulary for domain description (schema.org or another).	(Richardson and others, 2013).
API-3	Provide a starting URL.	(Richardson and others, 2013).
API-4	Don't mention a technology in URLs.	(Verborgh and others, 2015; Richardson and others, 2013).
API-5	Create self-described URLs (resource must be obvious from URL and it's clear how the request changes the resource).	(Richardson and others, 2013).
API-6	Provide next steps in the server request.	(Verborgh and others, 2015; Richardson and others, 2013).
API-7	Create API documentation.	(Robillard a Chhetri, 2015; Myers and others, 2010).
API-8	Create code examples of the API's implementation.	(Myers and others, 2010).
API-9	Map HTTP methods by REST conventions (GET=get, POST=create, PUT/PATCH changes, DELETE=remove).	(Richardson and others, 2013).
API-10	Use OAuth for authentication/authorization.	(Richardson and others, 2013).
API-xx	*(Free for custom rules/extensions).*	*N/A*

3. Invoice Home API description

The main purpose of *Invoice Home* is to allow its users to issue invoices, send them out to their customers and help them to be paid. Typical user of *Invoice Home* is a small business or individual entrepreneur. The service is provided world-wide, has users from more than 150 countries but more than half of the users are from the United States.

The described API will be a part of a new release of *Invoice Home* in 2017.

Object schema of the *Invoice Home* has tens of objects but the main objects which are exposed via API are the following:

1. **Invoice** – an invoice is issued by a user and send to a customer who pays it.
2. **Item** – an invoice has one or many items. Items have description, amount and quantity. The sum of items is an invoice's total and it's paid by a customer (the recipient of the invoice).
3. **Customer** – a customer is a person or organization who receives an invoice from user. It's a payer of the invoice.

These objects can be accessed via new *Invoice Home* API under the constraints of *10 Rules for Writing REST APIs* as follows.

Please notice that for brevity not all the object's attributes nor complete API implementation is shown in the following examples. Invoice object has about 30 attributes and if we would describe them entirely, the paper readiness would get worse.

When exposing an implementation, only particular fragments of JSON code related to the current API-rule are described and if it makes sense, they are marked in bold. This way, the intention is more clear compared to the complete output. Also, for brevity, only HTTP responses are shown in examples because the original HTTP request is a part of the response and it's easy to determine.

API-1

Rule ID: API-1 (Table 1: 10 Rules for Writing REST APIs).

Rule Description: Use a standard for medium and format of transmitted data (XML or JSON for example).

Comment: As a medium format for *Invoice Home* and per (Richardson and others, 2013, loc. 719; Verborgh and others, 2015, pp. 239; Amundsen, 2011), JSON+Collection (*application/vnd.collection*+json) has been chosen.

Pros: All the advantages of simple JSON data format and additional JSON semantics (which is missing for pure JSON but added with *JSON+Collection*) are the main pros. *JSON+Collection* schema can be easily extended in future so the format of the attributes doesn't need to be locked in the very first version. Of course, with every future API change, versioning is mandatory and hopefully, *JSON+Collection* supports direct versioning with attribute "*version*".

Cons: Because of putting all custom (*Invoice Home's* domain specific) attributes under the "*data*" key, the original meaning of default *JSON+Collection* attributes (like a *collection* or *items*) might not be obvious. Key naming scheme of the application domain (under the „*data*" key) is important too – it needs to be as self-describing as possible to deduce and understand semantics.

Example 1: JSON HTTP response to GET all Invoices, following API-1 rule (Use a standard for medium and format of transmitted data (XML or JSON for example)).

```
{
  "collection": {
    "version": "1.0",
    "href": "https://api.invoicehome.com/invoices",
    "items": [
      {
        "href": "https://api.invoicehome.com/invoices/1",
        "data": [
        { "name": "invoiceId", "value": "1", "prompt": "Identifier" },
        { "name": "number", "value": "100", "prompt": "Invoice Number" },
        ]
      }
    ]
  }
}
```

Example 2: JSON HTTP response to GET an Invoice (including Items) and following API-1 rule (Use a standard for medium and format of transmitted data (XML or JSON for example)).


```
{
  "collection": {
    "version": "1.0",
    "href": "https://api.invoicehome.com/invoices/1",
    "items": [
      {
        "href": "https://api.invoicehome.com/invoices/1/items",
        "data": [
        { "name": "itemId", "value": "1", "prompt": "Identifier" },
        { "name": "quantity", "value": "2", "prompt": "Quantity" },
        { "name": "amount", "value": "1000", "prompt": "Amount" },
        { "name": "description", "value": "Guitars", "prompt": "Description" },

        ]
      }
    ]
  }
}
```


Example 3: JSON HTTP response to GET all customers, following API-1 rule (Use a standard for medium and format of transmitted data (XML or JSON for example)).


```
{
  "collection": {
    "version": "1.0",
    "href": "https://api.invoicehome.com/customers",
    "items": [
      {
        "href": "https://api.invoicehome.com/customers/1 ",
        "data": [
        { "name": "customerId", "value": "1", "prompt": "Identifier" },
        { "name": "name", "value": "John Doe", "prompt": "Name or Organization" },
        { "name": "billing", "value": "3651 Lindell Rd., Las Vegas", "prompt": "Billing Address" },
        { "name": "shipping", "value": "1609 Star Way Rd., New York", "prompt": "Shipping Address" },

        ]
      }
    ]
  }
}
```


API-2

Rule ID: API-2 (Table 1: 10 Rules for Writing REST APIs).

Rule Description: Use a vocabulary for domain description (schema.org or another).

Comment: *Invoice Home* API uses Schema.org vocabulary (Schema.org, 2016). Existing standard or vocabulary means that the attributes of an object have been already given. The object is also placed in some context, for example *Thing/Intangible/Invoice* in case of used Schema.org vocabulary (Schema.org Invoice, 2016).

Pros: Involving a standard vocabulary means that any external system using the API would understand *Invoice Home's* data in the clear way. API's implementation is also easier because the attributes are well explained and described so their purpose is clear. In general, if anyone would use such standards/vocabularies only, inter-connection among systems would be faster and simpler to implement.

Cons: Unfortunately, not all the used attributes are present in the vocabulary. This means that each API's author has stay within borders given by the standard. However, the internal database model of the application doesn't need to follow a vocabulary schema because the schema is related to the

exposed API only. In most cases, some mappings from an internal database structure to standard/vocabulary need to be created.

Example 4: Invoice JSON HTTP response to GET an Invoice, following rule API-2 (Use a vocabulary for domain description (schema.org or another)).

```
{
  "collection": {
    "version": "1.0",
    "href": "https://api.invoicehome.com/invoices",
    "items": [
      {
        "href": "https://api.invoicehome.com/invoices/1",
        "data": [
        { "name": "customer", "value": "Jane Doe", "prompt": "Customer Name" },
        ],
        "links": [
        { "rel": "pdf", "href": "https://api.invoicehome.com/invoices/1.pdf", "prompt": "Invoice PDF" },
        { "rel": "profile", "href": "http://schema.org/Invoice" },
        ]
      }
    ]
  }
}
```

API-3

Rule ID: API-3 (Table 1: 10 Rules for Writing REST APIs).

Rule Description: Provide a starting URL.

Comment: When starting work with an external API, providing a guide or help where to begin could be useful. This is good for robots too – API could be easily indexed outside or becoming a part of auto-generated documentation, be a part of a search for API's etc.

The starting URL for *Invoice Home* API is https://api.invoicehome.com and accepts requests with HTTP GET and HTTP OPTIONS methods. **The API provides a list of URL's for two main objects – Invoice and Customer (Item is a sub-part of Invoice and Invoice ID is needed so Item doesn't have a starting URL).**

Pros: Easy to implement and understand. No glitches here.

Cons: When extending an API, a consideration what new objects expose with a starting URL is needed. Not all objects need to be exposed, only frequently used and the most important ones. General advice would be to expose as few objects as possible because it decreases the number of combinations. More combinations means more probability of issues.

Example 5: JSON HTTP response for rule API-3 (Provide a starting URL).

```
{
  "collection": {
    "version": "1.0",
    "href": "https://api.invoicehome.com",
    "items": [
      {
        "href": "https://api.invoicehome.com",
        "links": [
        { "rel": "invoices", "href": "https://api.invoicehome.com/invoices", "prompt": "Invoice List" },
        { "rel": "reports", "href": "https://api.invoicehome.com/customers", "prompt": "Customer List" },
        ]
      }
    ]
  }
}
```

API-4

Rule ID: API-4 (Table 1: 10 Rules for Writing REST APIs).

Rule Description: Don't mention a technology in URLs.

Comment: Not mention a technology in any exposed URL is a crucial for long term API usage. Exposing a technology outside (as being a part of URL) is a commitment which binds the author to support current technology forever. When a technology switch is needed, the new technology still must support old (and probably already unrelated) URL's. *Invoice Home* doesn't mention any internal technology in their exposed URLs.

Pros: Technology could be switched without any broke of commitment (with an exception to support all the previously exposed URLs and data, of course).

Cons: Each new URL exposed needs to be checked and any mentioning of technology/internal implementation must be removed from any URL. This check could take some time and additional expenses but pros are clear.

Example 6: API-4 (Don't mention a technology in URLs).

https://api.invoicehome.com - Starting URL.
https://api.invoicehome.com/invoices - Invoice List.
https://api.invoicehome.com/invoices/1 - Invoice Detail.
https://api.invoicehome.com/invoices/1/items - Invoice Items.
https://api.invoicehome.com/items/1 - Item Detail.
https://api.invoicehome.com/customers - Customer List.
https://api.invoicehome.com/customers/1 - Customer Detail.

API-5

Rule ID: API-5 (Table 1: 10 Rules for Writing REST APIs).

Rule Description: Create self-described URLs (resource must be obvious from URL and it's clear how the request changes the resource).

Comment: Clear and self-describing URLs are easy to remember and implement. Authors of external systems working with our API don't need to study a documentation repeatedly. Number of issues related to incomprehension or by error could decrease too. *Invoice Home* uses English-naming in URL's only.

Pros: Clear URLs, easy to remember, easy to implements. URLs are intended for humans, not for computers.

Cons: Self-describing URL are related to the language. With any new language supported by application we need to consider if we would add language-related URLs to the API too. The simplest implementation would be to offer English URLs only because in IT, English is just a standard. Also, for some long names, URLs could be pretty long which could lead to mistyping.

Example 7: API-5 (Create self-described URLs (resource must be obvious from URL and it's clear how the request changes the resource)).

https://api.invoicehome.com/invoices - This URL is related to all Invoices.
https://api.invoicehome.com/invoices/1 - This URL is related to an Invoice.
https://api.invoicehome.com/invoices/1/items - This URL is related to items under an Invoice.
https://api.invoicehome.com/items/1 - This URL is related to an Item.
https://api.invoicehome.com/customers - This URL is related to all Customers.
https://api.invoicehome.com/customers/1 - This URL is related to a Customer.

API-6

Rule ID: API-6 (Table 1: 10 Rules for Writing REST APIs).

Rule Description: Provide next steps in the server request.

Comment: API calls are completely unrelated to each other. There is simply 1 HTTP request/response and this cycle doesn't know about past / future calls. If we think about API calls in time, there would be useful that each response suggests the most probably next steps. For example, when viewing an invoice (and receive HTTP response with invoice details), API can provide a link to download invoice's PDF so the caller doesn't have to look at external documentation.

Invoice Home API suggests next steps via "*links*" attribute in the API response (in the following example we offer a link to download a PDF while providing details about an invoice).

Pros: Decreases a need for read an external documentation. It's self-describing for robots too. Makes API more clear because the author needs to think about connection (flow) within the API.

Cons: More time-consuming to implement. More data transferred / stored / read. Responses couldn't be as simple compared to the version without any next steps mentioning.

Example 8: JSON HTTP response to GET invoices, following API-7 (Provide next steps in the server request).

```
{
  "collection": {
    "version": "1.0",
    "href": "https://api.invoicehome.com/invoices",
    "items": [
      {
        "href": "https://api.invoicehome.com/invoices/1",
        "data": [
        { "name": "invoiceId", "value": "1", "prompt": "Identifier" },
        ],
        "links": [
        { "rel": "pdf", "href": "https://api.invoicehome.com/invoices/1.pdf", "prompt": "Invoice PDF" },
        ]
      }
    ]
  }
}
```

API-7

Rule ID: API-7 (Table 1: 10 Rules for Writing REST APIs).

Rule Description: Create API documentation.

Comment: Creating API documentation is a need in case when self-describing URL's are insufficient or just by following the good manners. *Invoice Home* API has just minimal external HTML documentation which is generated automatically from directly from the API sources. The documentation is available at

https://api.invoice.com (point out that the URL is same for HTML and API JSON calls). Providing starting point, self-describing URL's, next steps and attribute description is a documentation itself so we believe that a need for additional HTML documentation wouldn't be demanding.

Pros: Providing completed guide for any external API consumer (and for author too).

Cons: Very time consuming. The biggest issue with an external documentation is to keep it sync with current version of API. Every little change in API leads to the need for updating the external documentation too. As has been mentioned this could be solved with API's starting points/self-describing URLs and next steps.

Also, beware of a different URL's. When providing API at https://api.invoicehome.com which reacts to JSON calls it would be desired to provide any other format of documentation (HTML, PDF) at the same URL too. Any exception from this rule leads to issues.

API-8

Rule ID: API-8 (Table 1: 10 Rules for Writing REST APIs).

Rule Description: Create code examples of the API's implementation.

Comment: Code examples are the part of documentation created at API-7. Currently, Ruby, PHP and pure HTTP request/response examples (as a fallback for any other languages/platforms) are available.

Pros: Allows API users (consumers) to connect their services faster and increases credibility of the API. Based on previous experience, such examples are the most searched part of documentation.

Cons: Time consuming. Complexity of examples is also multiplied by number of languages/platforms we support. There is also external dependency towards to languages so the author of documentation must be sure that the examples still work when a new version of language or platform is released. It means that a continuous verification of these dependencies is required.

API-9

Rule ID: API-9 (Table 1: 10 Rules for Writing REST APIs).

Rule Description: Map HTTP methods by REST conventions (GET for get, POST for create, PUT/PATCH for update, DELETE for remove).

Comment: One advantage of HTTP methods is that they are self-describing. It's obvious that GET methods means get "something" and PATCH is a request for "change" or "fix". Following this concept, API doesn't need long or detailed explanation what a request means. *Invoice Home* follows this concept entirely, for example:

- **GET for invoice detail**
- **POST for creating a new invoice**
- **PATCH for invoice update**
- **DELETE for invoice delete**
- **OPTIONS for a list of all URL's related to invoice**

Pros: Using HTTP methods means "speaking" with an API in a natural, human-oriented way.

Cons: Even though HTTP methods are clear for English-speaking parts of the world, the rest could suffer. However, English is a standard for IT industry so this is just a remark.

Example 9: JSON HTTP response to Invoice's OPTIONS, following API-9 (Map HTTP methods by REST conventions (GET for get, POST for create, PUT/PATCH for update, DELETE for remove)).


```
{
  "collection": {
    "version": "1.0",
    "href": "https://api.invoicehome.com/invoices",
    "items": [
      {
        "href": "https://api.invoicehome.com/invoices",
        "links": [
        { "rel": "get", "href": "https://api.invoicehome.com/invoices/1", "prompt": "Invoice detail" },
        { "rel": "post", "href": "https://api.invoicehome.com/invoices", "prompt": "Create a new invoice" },
        { "rel": "put", "href": "https://api.invoicehome.com/invoices/1", "prompt": "Update invoice" },
        { "rel": "delete", "href": "https://api.invoicehome.com/invoices/1", "prompt": "Delete invoice" },
        { "rel": "options", "href": "https://api.invoicehome.com/invoices", "prompt": "Options for invoice" },
        ]
      }
    ]
  }
}
```


API-10

Rule ID: API-10 (Table 1: 10 Rules for Writing REST APIs).

Rule Description: Use OAuth for authentication/authorization.

Comment: *Invoice Home* API doesn't support OAuth (OAuth 2.0, 2016)
authentication/authorization in version 1.0. Only basic HTTP authentication/authorization is

used. We have decided to support OAuth in next releases due to more difficult implementation compared to pure HTTP auth.

Pros: HTTP authentication/authorization is the easiest way how to implement basic security. The solution is proven to be usable and stable as it's a part of HTTP protocol itself.

Cons: HTTP authentication/authorization doesn't support token revocation, multiple device-tokens, long-term tokens and all the features of OAuth.

Comments:

Example 10: Part of HTTP header for authentication/authorization. Currently, OAuth is not supported.

```
GET / HTTP/1.1
Host: https://api.invoicehome.com
Authorization: Basic bmVqYWt5aGVzbG8=
```

4. Conclusion

The most rules from "*The 10 Rules for Writing REST APIs*" have been considered as easy to follow and implement. First version of *Invoice Home* API follows all the rules except the last API-10 (Use OAuth for authentication/authorization). OAuth (OAuth 2.0, 2016) is a very robust security concept but it's implementation is not trivial. If we would have used an external library to implement it, maybe the implementation could be available since the first version but we've decided to postpone it to future releases. API needs to be run in its easiest form first and anything above the basic scope could cause more issues because the complexity of system increases.

Involvement of an external vocabulary (API-2 - Use a vocabulary for domain description (schema.org or another)) has been understood as an obstacle, particularly the attributes for Schema.org Invoice (Schema.org Invoice, 2016) were almost incompatible with *Invoice Home*'s Invoice object. We understand the advantage of the same schema across systems but when we realize that we need to bend our API just because of something "external" which could cause our API less readiness or even hard to use, we would break this rule by design.

Implementation of *Invoice Home* API proven that "*The 10 Rules for Writing REST APIs*" are well designed, described and usable. Author believe that any similar system or, in general, any web application could benefit from it too. Author would be also grateful for any comments and suggestions.

Resources

Amundsen, Mike., 2011: *Collection+JSON - Hypermedia Type*. [online]. [cit. 1.12. 2016]. Retrieved from http://amundsen.com/media-types/collection/

Businge, J., Serebrenik, A., van den Brand, Mark G & J, 2015: Eclipse API usage: the good and the bad., *Software Quality Journal*, vol. 23, no. 1, pp. 107-141.

ECMA International, 2016: *Standard ECMA-404 - The JSON Data Interchange Format* [online]. [cit. 1.12.2016]. Retrieved from: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

Fielding, Roy Thomas, 2000: Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine. [online]. [cit. 1.12.2016]. Retrieved from https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

ISO/EIC 42010, 2007: Systems and software engineering – Recommended practice for architectural description of software-intense systems. ISO/IEC

Hansson, David Heinemeier, 2016: *Ruby on Rails,* version 4.2.7.1. [online]. Retrieved from: http://rubyonrails.org/

Hradil, Jiří, 2016: Pravidla pro tvorbu RESTových API. *Systemova integrace.* 2016, vol.24, no.3-4: 97-112: ISSN 1214-6242

Myers, BA, Jeong, SY, Xie, YY, Beaton, J, Stylos, J, Ehret, R, Karstens, J, Efeoglu, A,Busse, DK, 2010: Studying the Documentation of an API for Enterprise Service-Oriented Architecture, Journal of Organizational and End User Computing, vol. 22, no. 1, pp. 23-51

OAuth 2.0, 2016: Internet Engineering Task Force (IETF). [online]. [cit. 1.12. 2016]. Retrieved from: https://tools.ietf.org/html/rfc6749

Rama, GM,Kak, A., 2015: Some structural measures of API usability, SOFTWARE-PRACTICE & EXPERIENCE, vol. 45, no. 1, pp. 75-110

RFC 5988, 2010: RFC 5988 – Web Linking. Internet Engineering Task Force (IETF). [online]. [cit. 1.12. 2016]. Retrieved from: https://tools.ietf.org/html/rfc5988

Richardson, Leonard. Amundsen, Mike. Ruby, Sam. 2013. RESTful Web APIs. O'Reilly Media. 1. vydání. ISBN: 978-1449358068

Robillard, MP, Chhetri, YB, 2015: Recommending reference API documentation, *Empirical Software Engineering,* vol. 20, no. 6, pp. 1558-1586

Schema.org, 2016: Community project. [online]. [cit. 1.12.2016]. Retrieved from: http://schema.org/

Schema.org Invoice, 2016: Community project. [online]. [cit. 1.12.2016]. Retrieved from: http://schema.org/Invoice

Scrum, 2016: Scrum Guides, ScrumGuides.org. [online]. [cit. 1.12.2016]. Retrieved from: http://www.scrumguides.org/

United Nations, 2016: ISIC Rev.4 - International Standard Industrial Classification of All Economic Activities. [online]. [cit. 17.6.2016]. Retrieved from: http://unstats.un.org/unsd/cr/registry/regcst.asp?Cl=27

Verborgh, R., van Hooland, S., Cope, A.S., Chan, S., Mannens, E. & Van, d.W, 2015: The fallacy of the multi-API culture, *Journal of Documentation,* vol. 71, no. 2, pp. 233-252.

Voříšek, J. and others, 2015. *Principy a modely řízení podnikové informatiky*. OECONOMIA. 2. vydání. ISBN: 978-80-245-2086-5

Wikilane Inc., 2016: Invoice Home. [online]. [cit. 1.12.2016]. Retrieved from: https://invoicehome.com

JEL Classification: L86