API CONFERENCE

The Liferay case

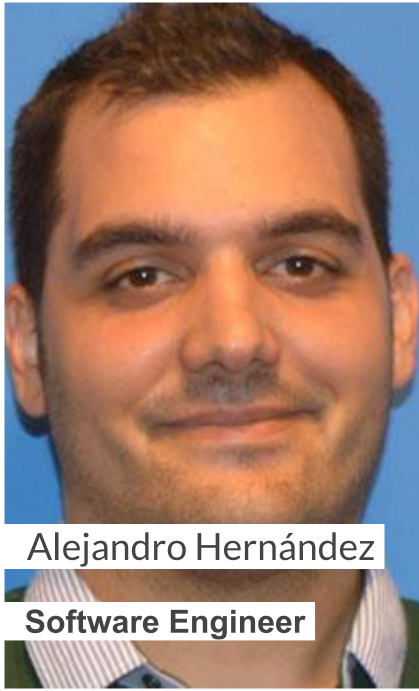
Lessons learned evolving from RPC to Hypermedia REST APIs

This slides are already available at

bit.ly/liferay-hypermedia-api

Who are we?

We work for Liferay Inc



Why do we need APIs?

Liferay is a software provider

Digital Experiences

Platform

Web, Mobile, ...

Open Source

On-Premise + Cloud

APIs

Key usages of APIs in Liferay

1. **Integration** (Cloud services, Legacy Apps, ...)
2. Omni-channel consumers
3. Web Applications

The beginnings: SOAP

Conclusions - The Good

- ✓ Enabled the possibility of **integration with external systems**
- ✓ Easy to build APIs thanks to **code generation** from Java APIs

Conclusions - The Ugly

- ✘ Compatibility problems
- ✘ Hard to consume APIs
- ✘ Strong dependency on tooling

⇒ Poor adoption

Next step: REST-API

“REST”-API

mmm….

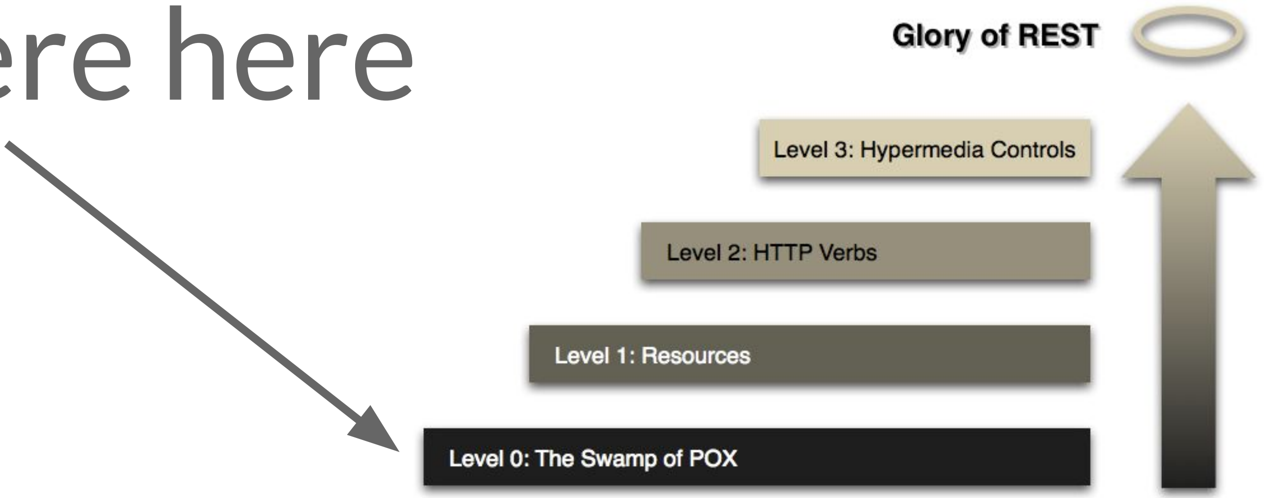
We mean RPC over HTTP

“REST”-API: JSON Web Services

- Automatic generation of an HTTP+JSON Web API from a Java API
- Auto-generated interactive documentation
- Batch operations

We were here

Is that bad?



Richardson Maturity Model - Martin Fowler

Conclusions - The Good

✓ **Very comprehensive** , 90+% of the platform's functionalities

✓ **More developer friendly**

✓ Interactive docs, batch operations, ... were highly appreciated ⇒ **More adoption**

Conclusions - The Ugly (1/2)

- ✘ Certain APIs were very difficult to consume
  - “Java-focused” objects did not match paradigms of all consumers

- ✘ **Custom** technology. Requires learning just for Liferay

Conclusions - The Ugly (2/2)

- ✘ Internal **changes auto-propagated** ⇒ Consumers were broken in every release
  - Unfeasible for public/partner APIs
- ✘ Increasingly **perceived as bad/old** API in comparison
  - “It’s not REST”

We also tried a “competing” approach!

- AtomPub (With Shindig)
  - Fully RESTful
  - Atom XML
- Mapping Layer
  - Manual Coding

It failed to gain any traction

Lessons!

Lessons

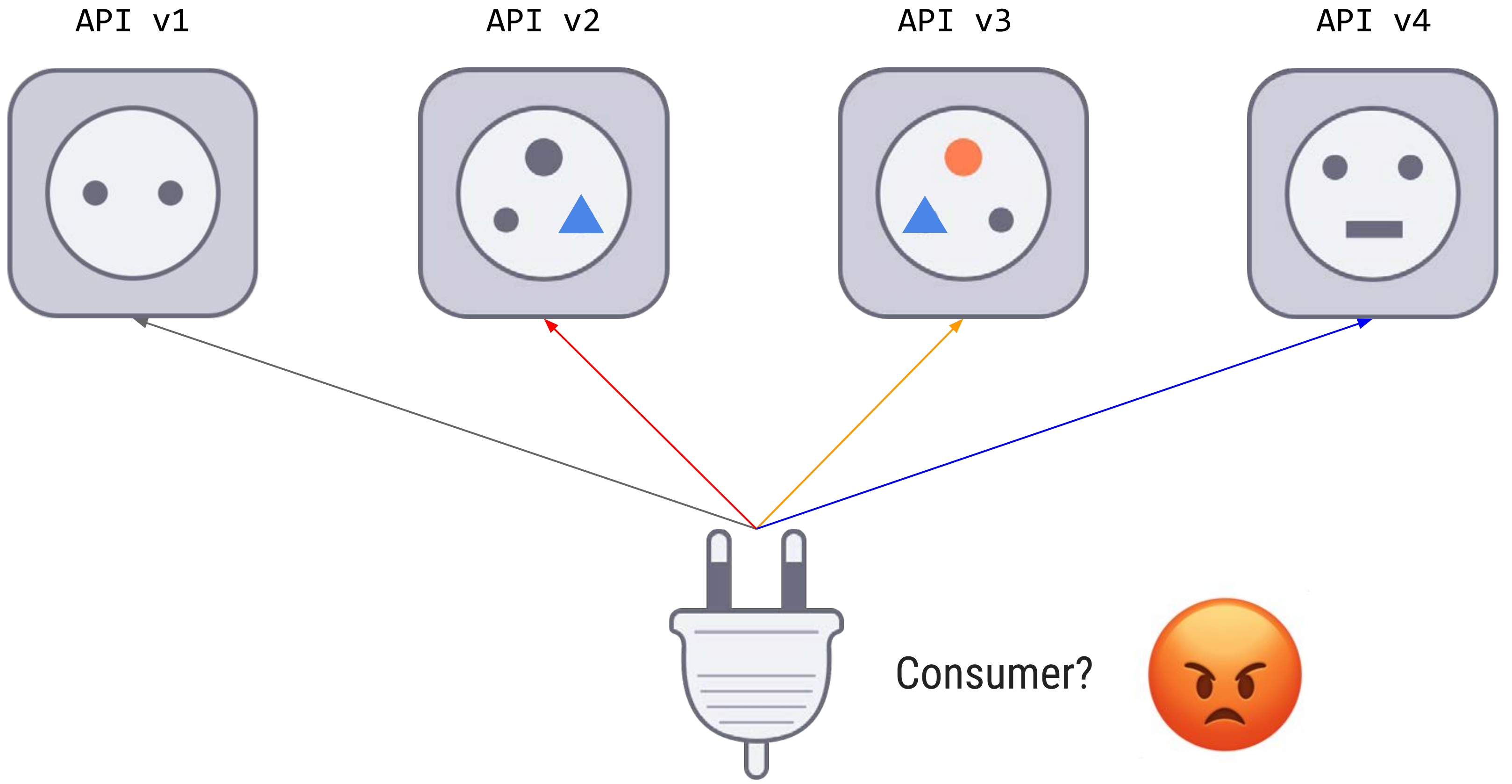
1. API generation means
   - ✓ Less work and more comprehensiveness
   - ✗ **Deep coupling**
2. Importance of **features for consumer devs**

In search of a better solution

Our two key challenges

Developer Experience

Evolution ~~Change~~ Management


API v1
API v2
API v3
API v4
Consumer?

The cost of breaking changes

For **consumer devs**

- Being forced to change code with each new version

For **API devs**

- Visible: Keep several API versions alive
- Hidden: Avoid change to reduce visible cost

Are we really the only ones with this problem?

How should APIs be versioned?

Is hypermedia really feasible or is it a utopia?

What is the *best* format for the API responses?

JSON or XML?

Or should it be binary?

HAL, JSON-LD, Siren, JSON-API, …?

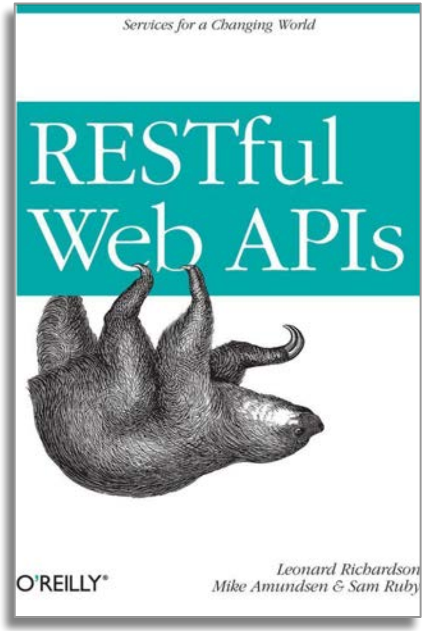
Is REST dead and should we go with GraphQL?

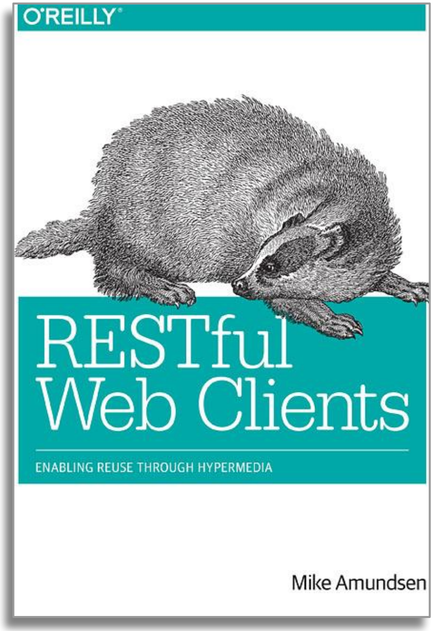
Learning from the best

1. The most popular “API Guidelines”
2. Tons of articles and several books.

Books that made a difference for us

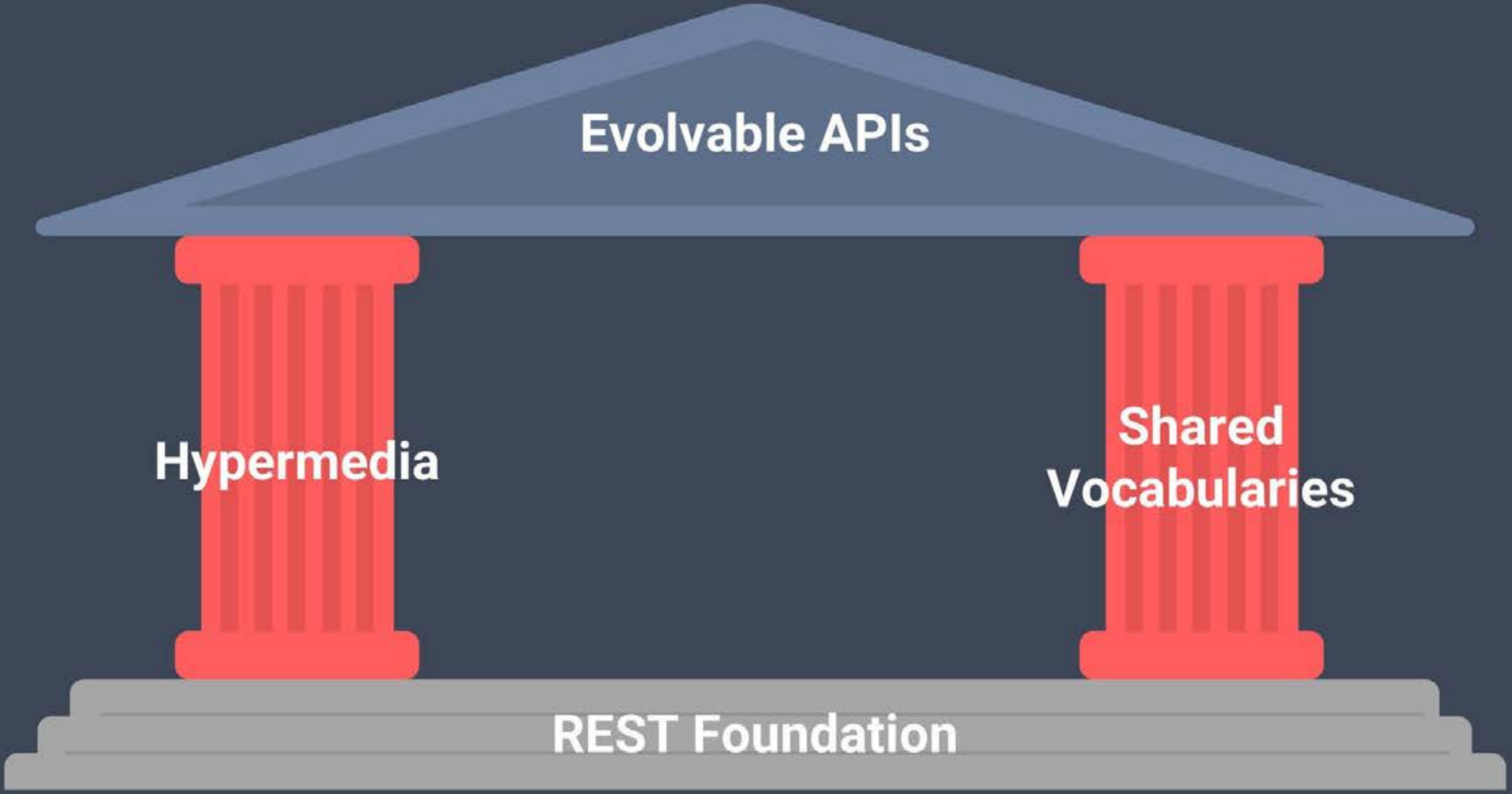
Beware of the many “bad” articles and books out there





API CONFERENCE

Our solution


Evolvable APIs
Hypermedia
Shared Vocabularies
REST Foundation

APIs designed to evolve

How we are solving each of the challenges

1. Hypermedia Controls

Home URL

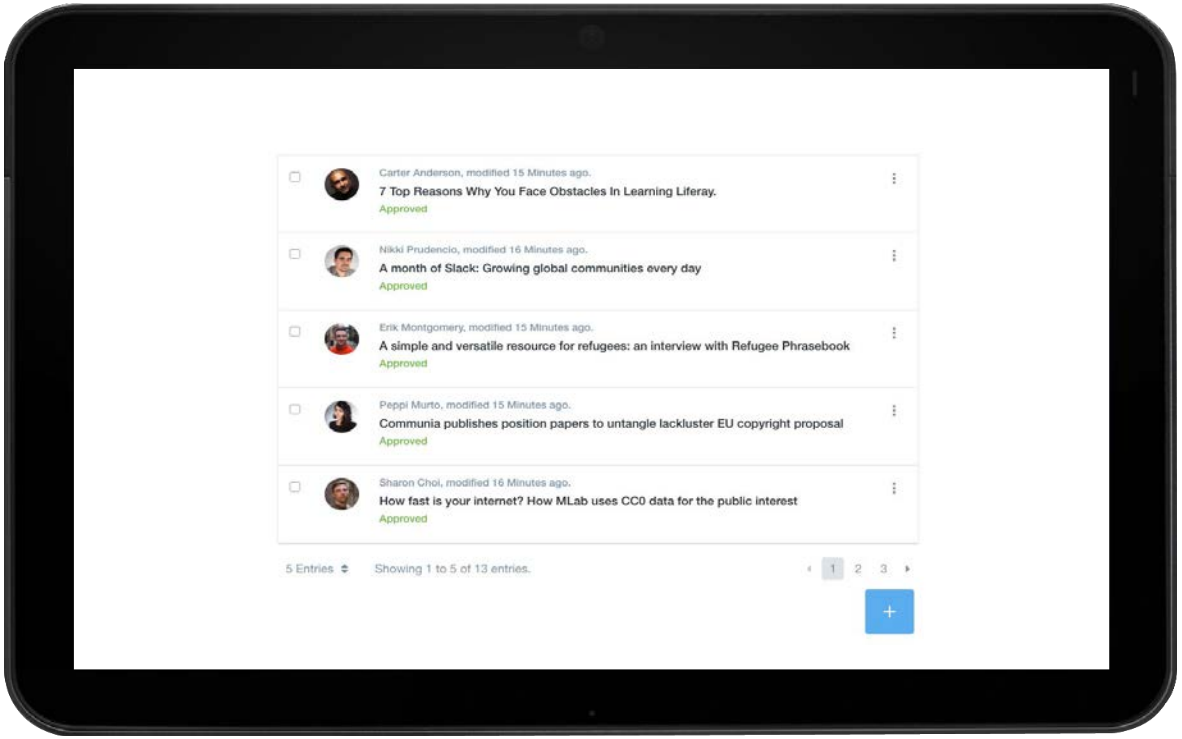
Consumers must only know
ONE URL

And how to navigate from it

Affordance Types

Contract with consumer defines
affordance types
(relations, actions, …)
Start with IANA’s [80 relation types]()

Pagination



Carter Anderson, modified 15 Minutes ago.
7 Top Reasons Why You Face Obstacles In Learning Liferay.
Approved

Nikki Prudencio, modified 16 Minutes ago.
A month of Slack: Growing global communities every day
Approved

Erik Montgomery, modified 15 Minutes ago.
A simple and versatile resource for refugees: an interview with Refugee Phrasebook
Approved

Peppi Murto, modified 15 Minutes ago.
Communia publishes position papers to untangle lackluster EU copyright proposal
Approved

Sharon Choi, modified 16 Minutes ago.
How fast is your internet? How MLab uses CC0 data for the public interest
Approved

5 Entries

Showing 1 to 5 of 13 entries.

Pagination

```
{
   "_embedded": {...},
   "total": 43,
   "count": 30,
   "_links": {
       "first": {
           "href": "http://localhost:8080/o/api/p/groups?page=1&per_page=30"
       },
       "next": {
           "href": "http://localhost:8080/o/api/p/groups?page=2&per_page=30"
       },
       "last": {
           "href": "http://localhost:8080/o/api/p/groups?page=2&per_page=30"
       }
   }
}
```

Defined by
IANA Link Relations

HAL

API CONFERENCE

Actions

```
{
   “properties”: {
      “title”: “Hypermedia is awesome”,
      …
   }
   "actions": [
       {
           "name": "delete-item",
           "title": "Delete Blog Posting",
           "method": "DELETE",
           "href": "http://localhost:8080/o/p/blogs/abcdef",
       }
       {
           "name": "publish",
           "title": "Publish Blog Posting",
           "method": "POST",
           "href": "http://localhost:8080/o/p/123URLs4123AREabcdeOPAQUEf41231",
       }
 …
```

SIREN

Forms

```
{
   ..
   "actions": [
       {
           "name": "add-blog-posting",
           "title": "Add Blog Posting",
           "method": "POST",
           "href": "http://localhost:8080/o/p/blogs",
           "type": "application/json",
           "fields": [
               { "name": "headline", "type": "text" },
               { "name": "author", "type": "Person" },
           ]
       }
   …
```

SIREN

2. Shared Vocabularies

Standard types

schema.org: 597 types y 867 properties

ActivityStreams, microformats, …

Well defined custom types

Never expose internal models

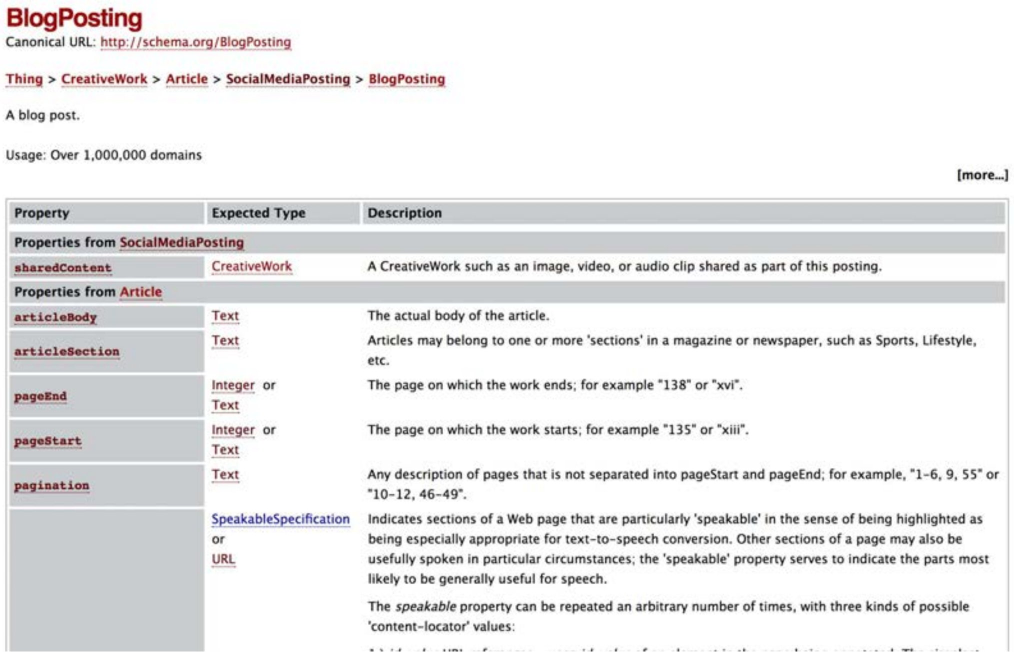
Custom types must be consumer focused

API CONFERENCE

Internal models should never be exposed in an API

Schema.org type

Inheritance-based model



BlogPosting

Canonical URL: http://schema.org/BlogPosting

Thing > CreativeWork > Article > SocialMediaPosting > BlogPosting

A blog post.

Usage: Over 1,000,000 domains

[more...]

Property	Expected Type	Description
Properties from SocialMediaPosting		
sharedContent	CreativeWork	A CreativeWork such as an image, video, or audio clip shared as part of this posting.
Properties from Article		
articleBody	Text	The actual body of the article.
articleSection	Text	Articles may belong to one or more 'sections' in a magazine or newspaper, such as Sports, Lifestyle, etc.
pageEnd	Integer or Text	The page on which the work ends; for example "138" or "xvi".
pageStart	Integer or Text	The page on which the work starts; for example "135" or "xiii".
pagination	Text	Any description of pages that is not separated into pageStart and pageEnd; for example, "1-6, 9, 55" or "10-12, 46-49".
	SpeakableSpecification or URL	Indicates sections of a Web page that are particularly 'speakable' in the sense of being highlighted as being especially appropriate for text-to-speech conversion. Other sections of a page may also be usefully spoken in particular circumstances; the 'speakable' property serves to indicate the parts most likely to be generally useful for speech. The *speakable* property can be repeated an arbitrary number of times, with three kinds of possible 'content-locator' values:

All attributes are optional

Defining types and their mapping to internal models and actions is the **most important API design activity**

And the most difficult

Communicating the types

OpenAPI	JSON Schema	ALPS
Format agnostic Widely adopted Focused on resources Not yet hypermedia friendly	Tied to JSON Focused on field types not semantics	Format agnostic API **Profile** ⇒ Focus on semantics

Goal: The smallest contract possible

- **One single URL**
- **Message types** instead of specific resources
- **Affordance types** instead of actions per resource

Focus on types!

Shared vocabularies make Hypermedia feasible

API CONFERENCE

Sure but, how are consumers built?

Web

ESB

Microservices

Mobile

IoT?

Building Consumers

1. Robustness principle
2. Code to types
3. Mindset change to “Game loop” (leads to big reusability)

How should APIs be versioned?

Do not version upfront

Design APIs to avoid breaking compatibility

Is hypermedia really feasible or is it a utopia?

It is feasible, and recent progress on standards and tools has made it much easier

What is the *best* format for the API responses?

It depends on the consumer.

Ideally, support “all” and let them decide

Is REST dead and should we go with GraphQL?

Nope

Does this work for real?

3 projects were we are applying this

1

Project: Microservice APIs

API stack: Java with Spring
Consumers: Java Microservice, Mobile App

Home URL

```
{
        name: "pulpo-api",
        description: "API for consuming PULPO Services",
        _links: {
                self: { href: "http://localhost:8084/" },
                accounts: {
                        href: "localhost/{projectId}/accounts{?filter,page,size,sort*}",
                        templated: true
                },
                account: {
                        href: "localhost/{projectId}/accounts/{identifier}",
                        templated: true
                },
                fields: {
                        href: "localhost/{projectId}/fields{?filter,page,size,sort*}",
                        templated: true
                },
                field: {
                        href: "localhost/{projectId}/fields/{identifier}",
                        templated: true
                },
        }
}
```

HAL

Links among resources

Affordance Types

```
{
        "dateCreated":"2017-11-15T16:23:35Z",
        "dateModified":"2017-11-15T16:23:35Z",
        "identifier":"AV_Afi6-Y3UMLZEdmkBE",
        "name":"Friends",
        "segmentType":"STATIC",
        "status":"ACTIVE",
        "_links":{
                "self":{
                        "href":"http://localhost:8084/my-project/individual-segments/AV_Afi6-Y3UMLZEdmkBE"
                },
                "individual-segments":{
                        "href":"http://localhost:8084/my-project/individual-segments{?filter}",
                        "templated":true
                }
        }
}
```

HAL

CONFERENCE

Hiding **internal** models

Affordance Types

```
@GetMapping(
     produces = {MediaType.APPLICATION_JSON_VALUE, "application/hal+json"},
     value = "/{identifier}"
)
public @ResponseBody Resource<Individual> findOne(
     @PathVariable String projectId, @PathVariable String identifier) {

     IndividualEntity individualEntity = _individualService.findOneByUUID(
          projectId, identifier);

     if (individualEntity == null) {
          throw new NotFoundException(
               "Unable to find Individual with individualUUID " + identifier);
     }

     return _individualResourceAssembler.toResource(individualEntity);
}
```

```
[
   {
      "title": “We are in APIConference!”,
      "subtitle": “APIConference”,
      "user": “localhost:8080/o/p/30325”
   },
   {
      "title": “5 amazing things!”,
      "subtitle": “Get english!”,
      "user": “localhost:8080/o/p/30325”
   }
]
```

```
[
   {
      "headline": “We are in APIConference!”,
      "alternativeHeadline”: “APIConference”,
      "author": “localhost:8080/o/p/30325”
   },
   {
      "headline": “5 amazing things!”,
      "alternativeHeadline": “Get english!”,
      "author": “localhost:8080/o/0/65443”
   }
]
```

localhost:8080/o/api/blogs?start=25&end=27

Hypermedia controls for pagination

```
[
    {
        "title": “We are in APIConference!”,
        "subtitle": “THE conference for APIs”,
        "user": “localhost:8080/o/p/30325”
    },
    {
        "title": “5 amazing things to do in
London!”,
        "subtitle": “Get english!”,
        "user": “localhost:8080/o/0/65443”
    }
]
```

```
{
    “count”: 2,
    “totalItems”: 30,
    “members”: [
        {
            "headline": “We are in APIConference!”,
            "alternativeHeadline": “APIConference”,
            "author": “localhost:8080/o/p/30325”
        },
        {
            "headline": “5 amazing things!”,
            "alternativeHeadline": “Get english!”,
            "author": “localhost:8080/o/0/65443”
        }
    ],
    “view”: {
        “next”: “localhost:8080/blogs?p=7&p_p=2”
    }
}
```

localhost:8080/o/api/blogs?page=6&per_page=2

How do I add support for queries?

Adopt **OData's** query language

Document that **all** collections support queries

This becomes part of our contract!

API CONFERENCE

We used several standards

HAL, IANA Link relations, OData queries

Consumer developers can reuse existing libraries

Project: Platform APIs

API stack: Java with OSGi and JAX-RS
Consumers: Mobile Apps, Think Web clients, ESBs, Legacy Apps, ...

Home URL

```
{
	"resources": {
		"blog-postings": {
			"href": "http://localhost:8080/p/blog-postings"
		},
		"web-sites": {
			"href": "http://localhost:8080/p/web-sites"
		},
		"documents": {
			"href": "http://localhost:8080/p/documents"
		},
		"organizations": {
			"href": "http://localhost:8080/p/organizations"
		},
		"people": {
			"href": "http://localhost:8080/p/people"
		}
	}
}
```

JSON-HOME

Support for several response formats

HAL, JSON-LD and Plain JSON

Affordance Types

Resource Links

```
{
    "@context": [
        { "creator": { "@type": "@id" } },
        { "@vocab": "http://schema.org/" },
        "https://www.w3.org/ns/hydra/core#"
    ],
    "@id": "http://localhost:8080/p/blog-postings/0",
    "@type": "BlogPosting",
    "alternativeHeadline": "Et eaque quod.",
    "articleBody": "Sunt adipisci eligendi dolorem ducimus placeat.",
    "creator": "http://localhost:8080/p/people/9",
    "dateCreated": "2017-07-11T11:06Z",
    "dateModified": "2017-07-11T11:06Z",
    "headline": "Alone on a Wide, Wide Sea"
}
```

JSON-LD + HYDRA

Affordance Types

```
{
    "@id": "http://localhost:8080/p/blog-postings/0",
    "@type": "BlogPosting",
    "creator": "http://localhost:8080/p/people/9",
    "headline": "Alone on a Wide, Wide Sea",
    "operation": [
        {
            "@id": "_:blog-postings/delete",
            "@type": "Operation",
            "method": "DELETE"
        },
        {
            "@id": "_:blog-postings/update",
            "@type": "Operation",
            "expects": "http://localhost:8080/f/u/blog-postings",
            "method": "PUT"
        }
    ]
}
```

Actions

JSON-LD + HYDRA

Affordance Types

```
{
    "@id": "http://localhost:8080/f/u/blog-postings",
    "@type": "Class",
    "description": "This can be used to create or update a blog posting",
    "supportedProperty": [
        {
            "@type": "SupportedProperty",
            "property": "creator",
            "required": false,
        },
        {
            "@type": "SupportedProperty",
            "property": "headline",
            "required": true,
        }
    ],
    "title": "The blog posting form"
}
```

Forms

JSON-LD + HYDRA

API CONFERENCE

Representor pattern

Let the consumer decide
what’s the best format for their needs

We created our own thin framework to add Hypermedia capabilities and Representor

Apio

Well defined custom types

```java
public Representor<BlogPostingModel, Long> representor(
  Builder<BlogPostingModel, Long> builder) {

  return builder.types(
      "BlogPosting"
  ).identifier(
      BlogPostingModel::getId
  ).addDate(
      "dateModified", BlogPostingModel::getModifiedDate
  ).addLinkedModel(
      "creator", PersonId.class, BlogPostingModel::getCreatorId
  ).addRelatedCollection(
      "comment", BlogPostingCommentId.class
  ).addString(
      "alternativeHeadline", BlogPostingModel::getSubtitle
  ).addString(
      "articleBody", BlogPostingModel::getContent
  ).addString(
      "headline", BlogPostingModel::getTitle
  ).build();
}
```

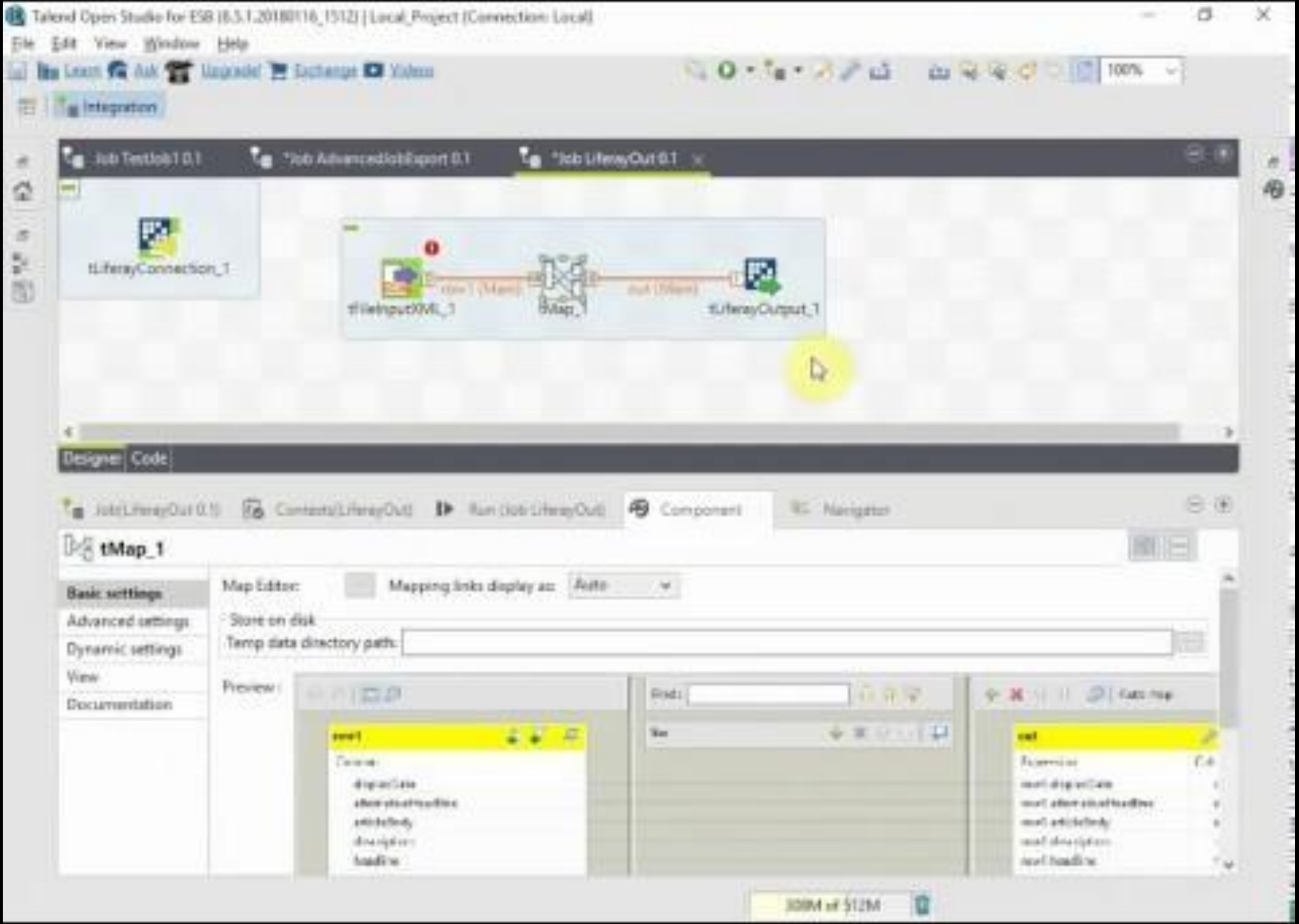
Schema.org's types can be a good start, but ultimately you will need to define your own types

API Discoverability in action!

3

Project: Data Integration through ETL/ESB

Consumer: Talend Plugin



Talend Open Studio for ESB (6.5.1.20180116_1512) | Local_Project (Connection: Local)

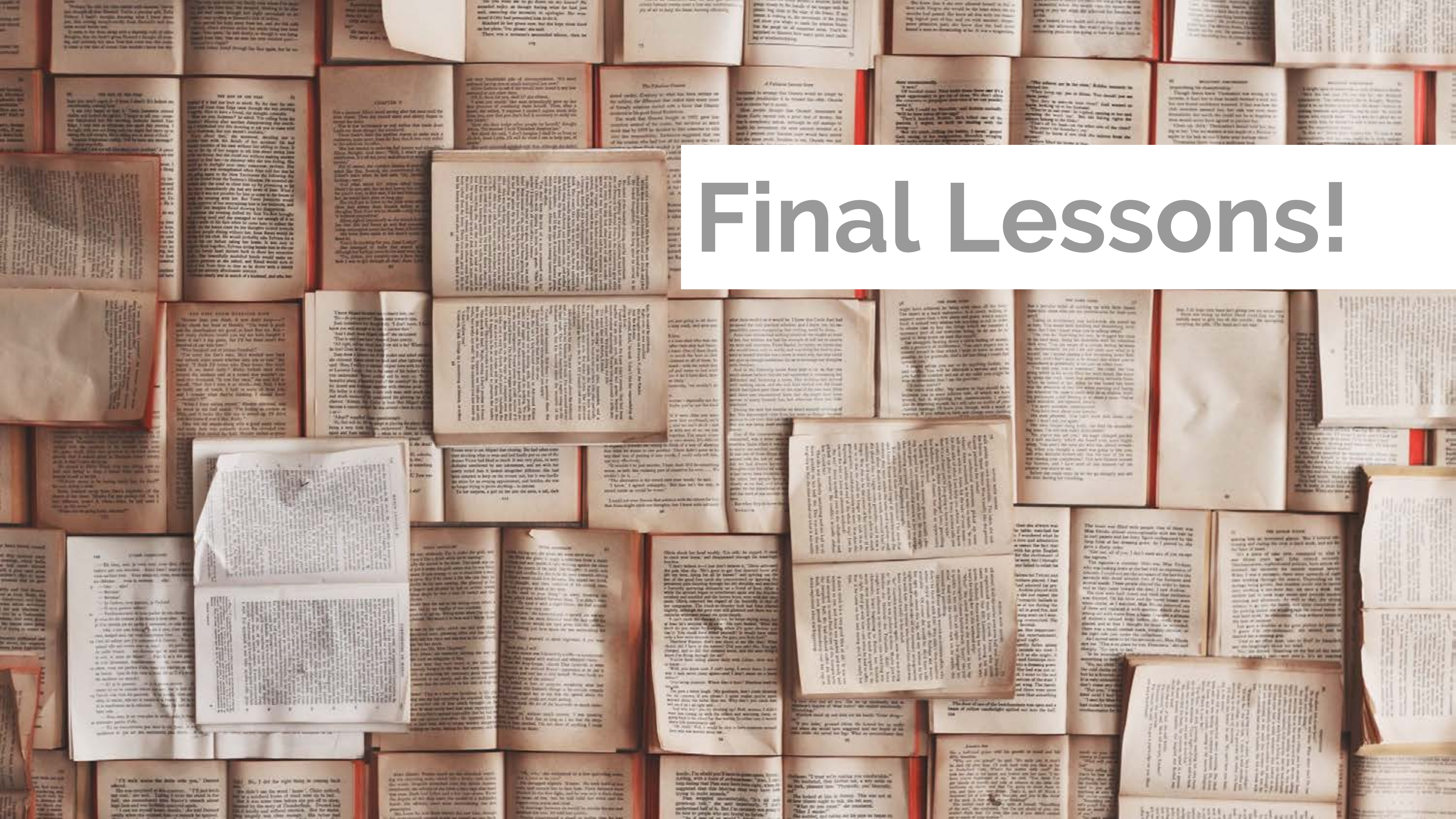
tMap_1

Map Editor: Mapping links display as: Auto

Store on disk

Temp data directory path:

Preview:



Final Lessons!

Your needs > Any specific solution

API CONFERENCE

REST

(with Hypermedia)

+

Shared Vocabularies

is the best solution for Evolvability

Spend time defining your vocabulary

It is the most important design activity for an API

Make consumers & their developers the focus of your API design strategy

- Provide features that make their job easier
- APIs should speak their language, not yours

Giving Back

Apio: An Open Source Project

Apio Architect

- Forces mapping layer
- Hypermedia by design

Initially for JAX-RS.

Community effort to port it to .NET, Python, Node,...

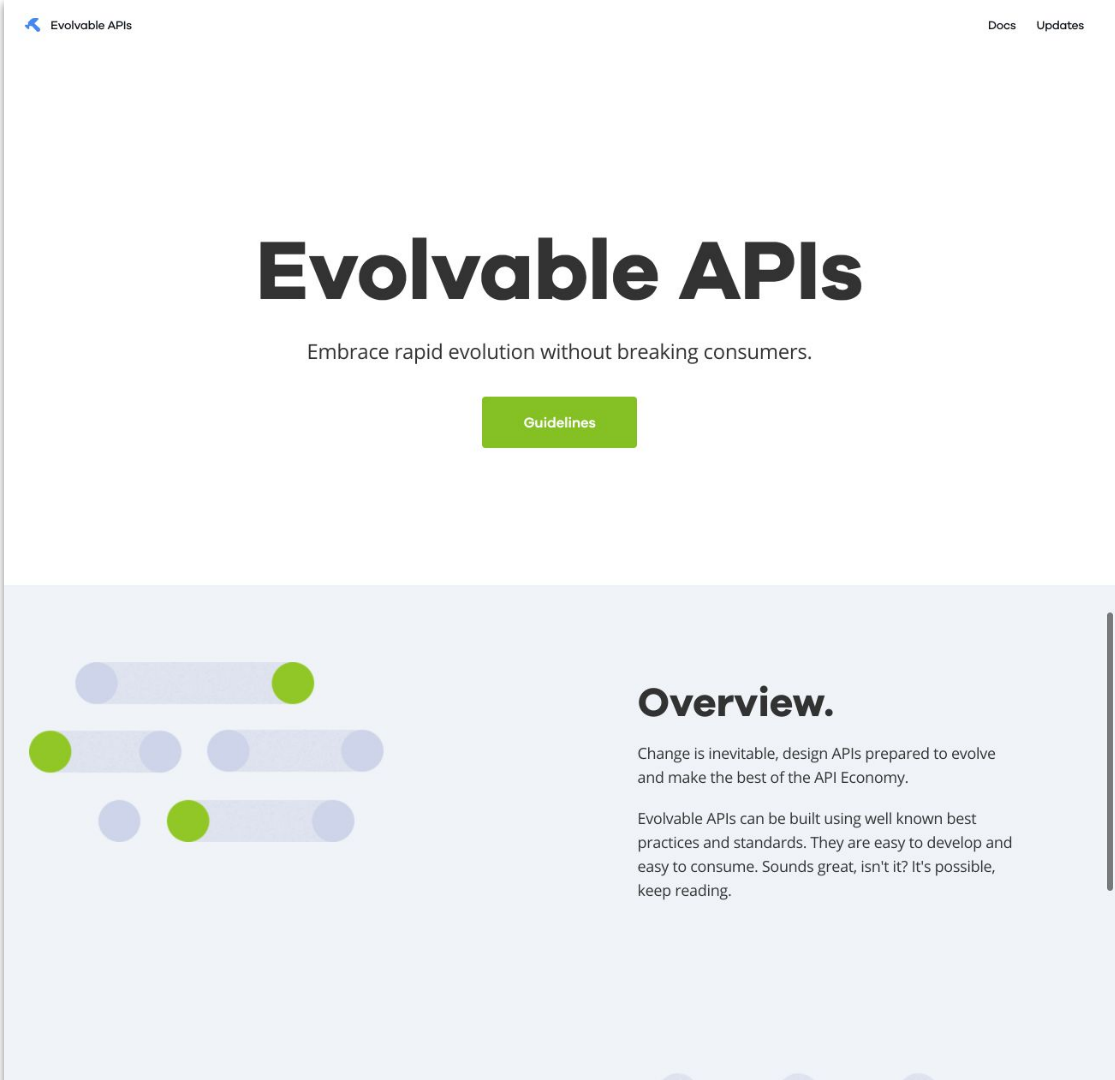
Apio Consumer

- Leverage high reusability made possible by Hypermedia
- Features: Retries, Offline support, ...

Web
Android
iOS

API CONFERENCE

Evolvable-apis.org
(Beta)



Evolvable APIs

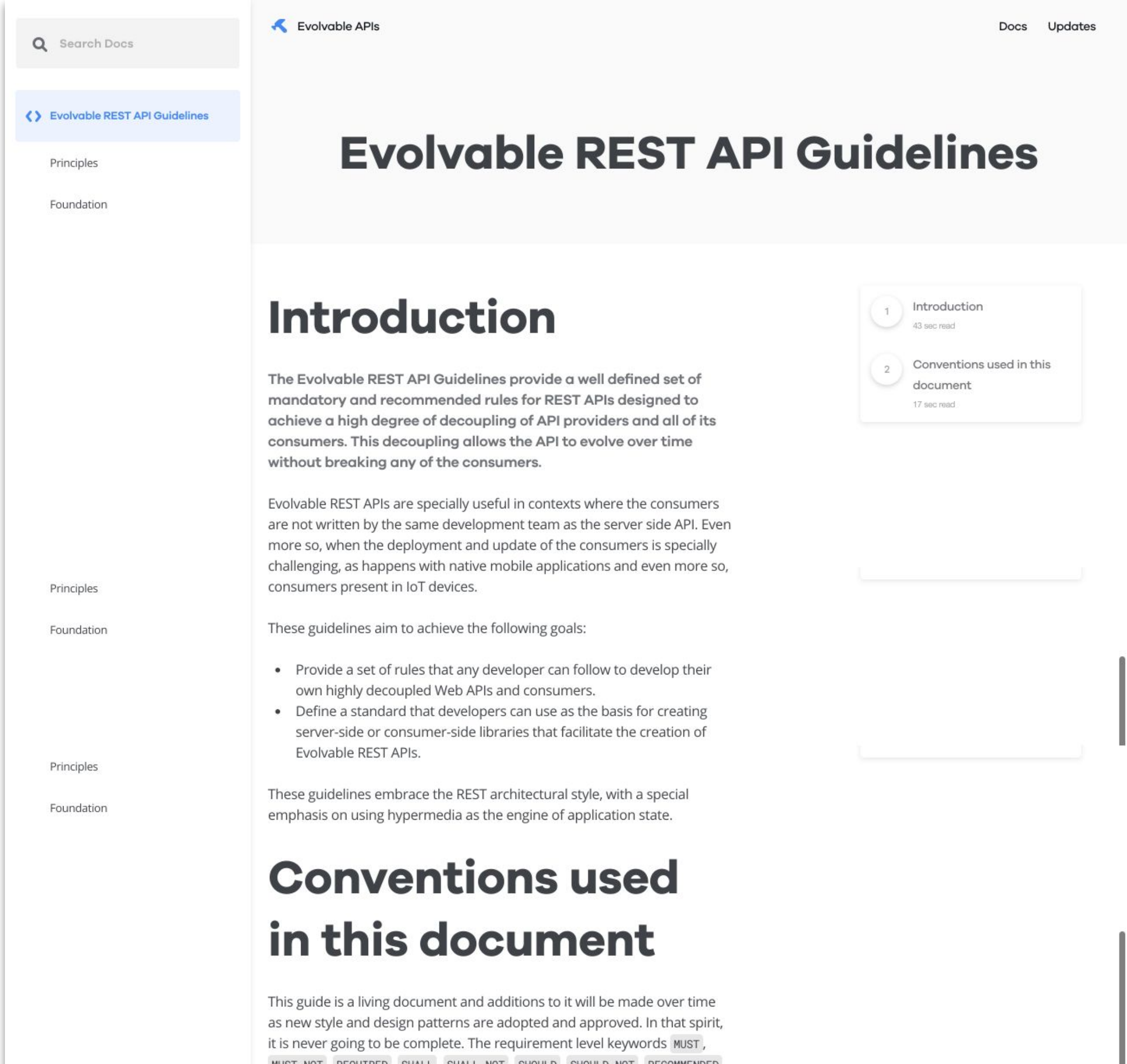
Embrace rapid evolution without breaking consumers.

Overview.

Change is inevitable, design APIs prepared to evolve and make the best of the API Economy.

Evolvable APIs can be built using well known best practices and standards. They are easy to develop and easy to consume. Sounds great, isn't it? It's possible, keep reading.

Evolvable-apis.org
(Beta)



Search Docs

Evolvable REST API Guidelines

Principles

Foundation

Principles

Foundation

Principles

Foundation

Evolvable APIs

Docs

Updates

Evolvable REST API Guidelines

Introduction

The Evolvable REST API Guidelines provide a well defined set of mandatory and recommended rules for REST APIs designed to achieve a high degree of decoupling of API providers and all of its consumers. This decoupling allows the API to evolve over time without breaking any of the consumers.

Evolvable REST APIs are specially useful in contexts where the consumers are not written by the same development team as the server side API. Even more so, when the deployment and update of the consumers is specially challenging, as happens with native mobile applications and even more so, consumers present in IoT devices.

These guidelines aim to achieve the following goals:

- Provide a set of rules that any developer can follow to develop their own highly decoupled Web APIs and consumers.
- Define a standard that developers can use as the basis for creating server-side or consumer-side libraries that facilitate the creation of Evolvable REST APIs.

These guidelines embrace the REST architectural style, with a special emphasis on using hypermedia as the engine of application state.

Conventions used in this document

This guide is a living document and additions to it will be made over time as new style and design patterns are adopted and approved. In that spirit, it is never going to be complete. The requirement level keywords `MUST`,

@alejandrohdezma / @jorgeferrer