Open Source Frameworks (OSF)
Designing Your REST API

Open Source Frameworks (OSF)
Master of Science in Engineering (MSE)
Olivier Liechti
olivier.liechti@heig-vd.ch

MASTER OF SCIENCE
IN ENGINEERING

MASTER OF SCIENCE
IN ENGINEERING

Agenda

Quick Intro
14h20 - 14h40

Group Work
14h40 - 16h30

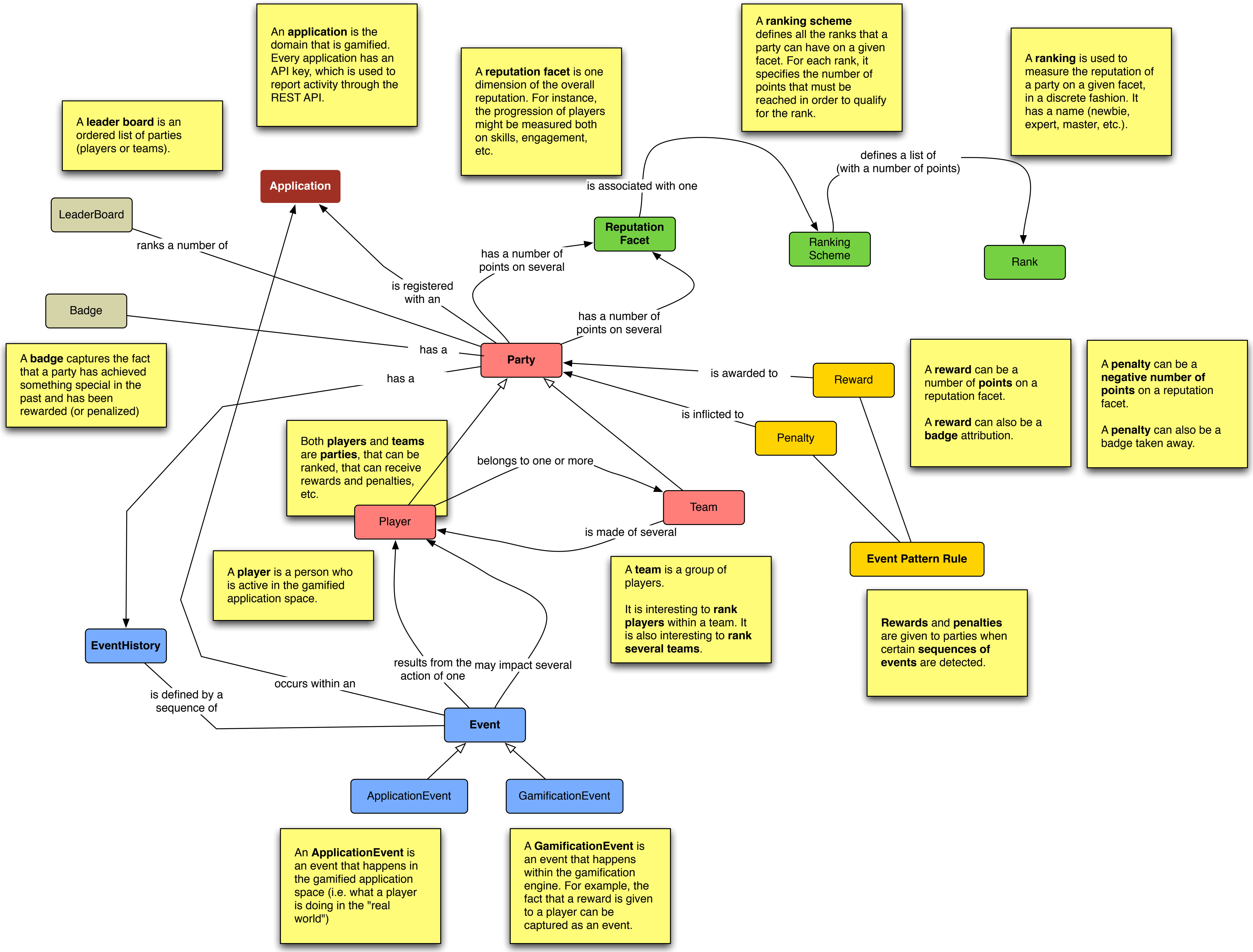
Review
16h30 ~

MASTER OF SCIENCE IN ENGINEERING

Planning

Date	Java EE Frameworks	Gamification Project
20.02.13	Overview + Business Tier (**EJB**)	Group formation + domain model analysis
27.02.13	Web Services (**JAX-RS** / **JAX-WS**)	**10' presentation** of the domain model + review
06.03.13	Design of the gamification REST API / project setup on Github	
13.03.13	**15' presentation** of the REST API + 15' discussion / documentation on Github	
20.03.13	Intro to javascript / node.js frameworks	node.js + express.js tutorial
27.03.13	**Spring Framework**	Implementation of the REST API
03.04.13	*Eastern Break*	*Eastern Break*
10.04.13	Persistence Tier (**JPA**)	NoSQL tutorial + relevance to the project
17.04.13	Project implementation	
24.04.13	Message Oriented Middleware (**JMS**)	Project implementation
01.05.13	**20' presentation** of your NoSQL back-end and its role in your project	
08.05.13	Presentation Tier (MVC / Frameworks)	javascript framework tutorial
15.05.13	Project implementation	
22.05.13	Project implementation	
29.05.13	**Final Presentations & Demos (30')**	

Gamification Domain Model (work-in-progress)



An **application** is the domain that is gamified. Every application has an API key, which is used to report activity through the REST API.

A **leader board** is an ordered list of parties (players or teams).

A **reputation facet** is one dimension of the overall reputation. For instance, the progression of players might be measured both on skills, engagement, etc.

A **ranking scheme** defines all the ranks that a party can have on a given facet. For each rank, it specifies the number of points that must be reached in order to qualify for the rank.

A **ranking** is used to measure the reputation of a party on a given facet, in a discrete fashion. It has a name (newbie, expert, master, etc.).

A **badge** captures the fact that a party has achieved something special in the past and has been rewarded (or penalized)

Both **players** and **teams** are **parties**, that can be ranked, that can receive rewards and penalties, etc.

A **reward** can be a number of **points** on a reputation facet.

A **reward** can also be a **badge** attribution.

A **penalty** can be a **negative number of points** on a reputation facet.

A **penalty** can also be a badge taken away.

A **player** is a person who is active in the gamified application space.

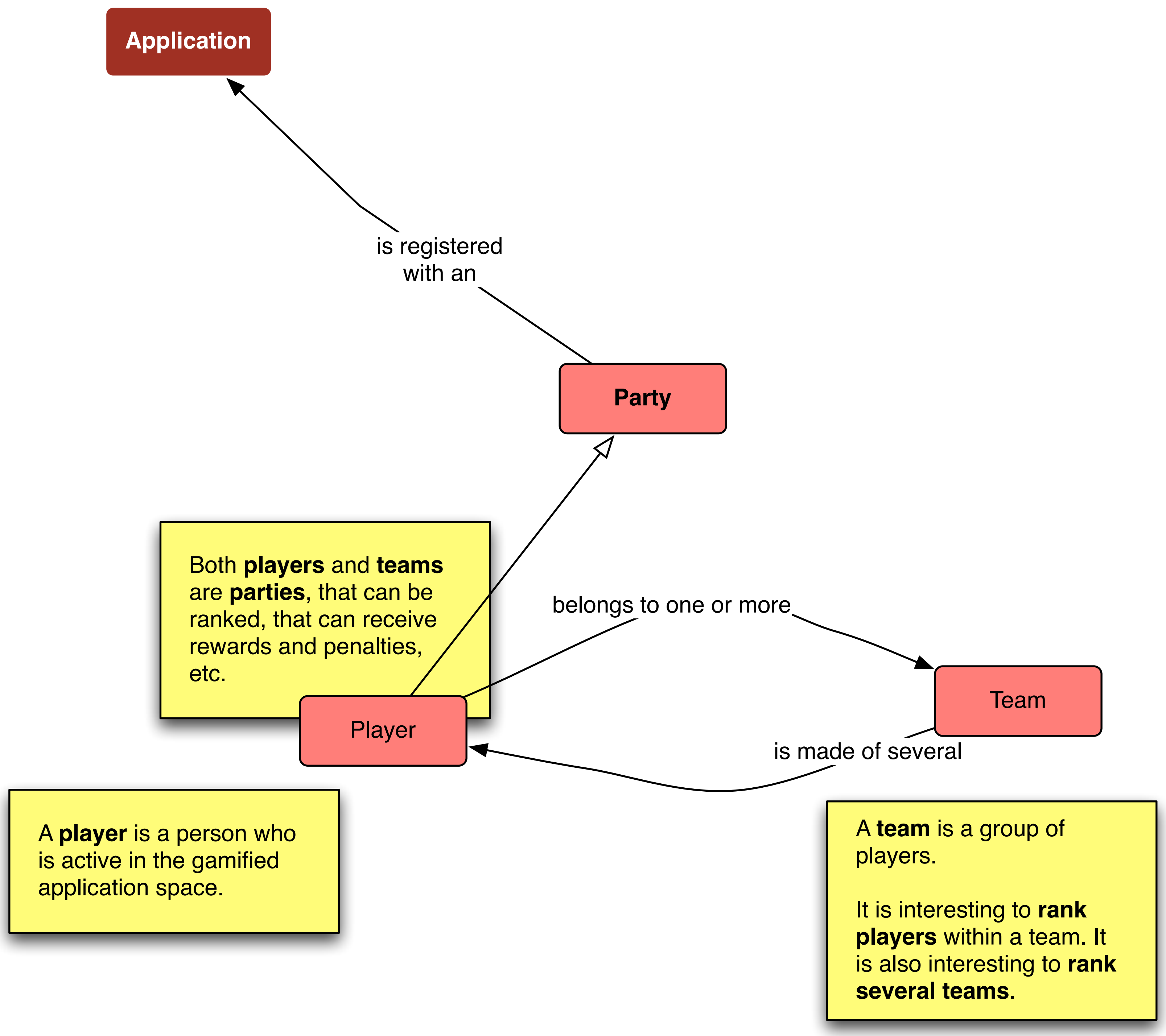
A **team** is a group of players.

It is interesting to **rank players** within a team. It is also interesting to **rank several teams**.

Rewards and **penalties** are given to parties when certain **sequences of events** are detected.

An **ApplicationEvent** is an event that happens in the gamified application space (i.e. what a player is doing in the "real world")

A **GamificationEvent** is an event that happens within the gamification engine. For example, the fact that a reward is given to a player can be captured as an event.

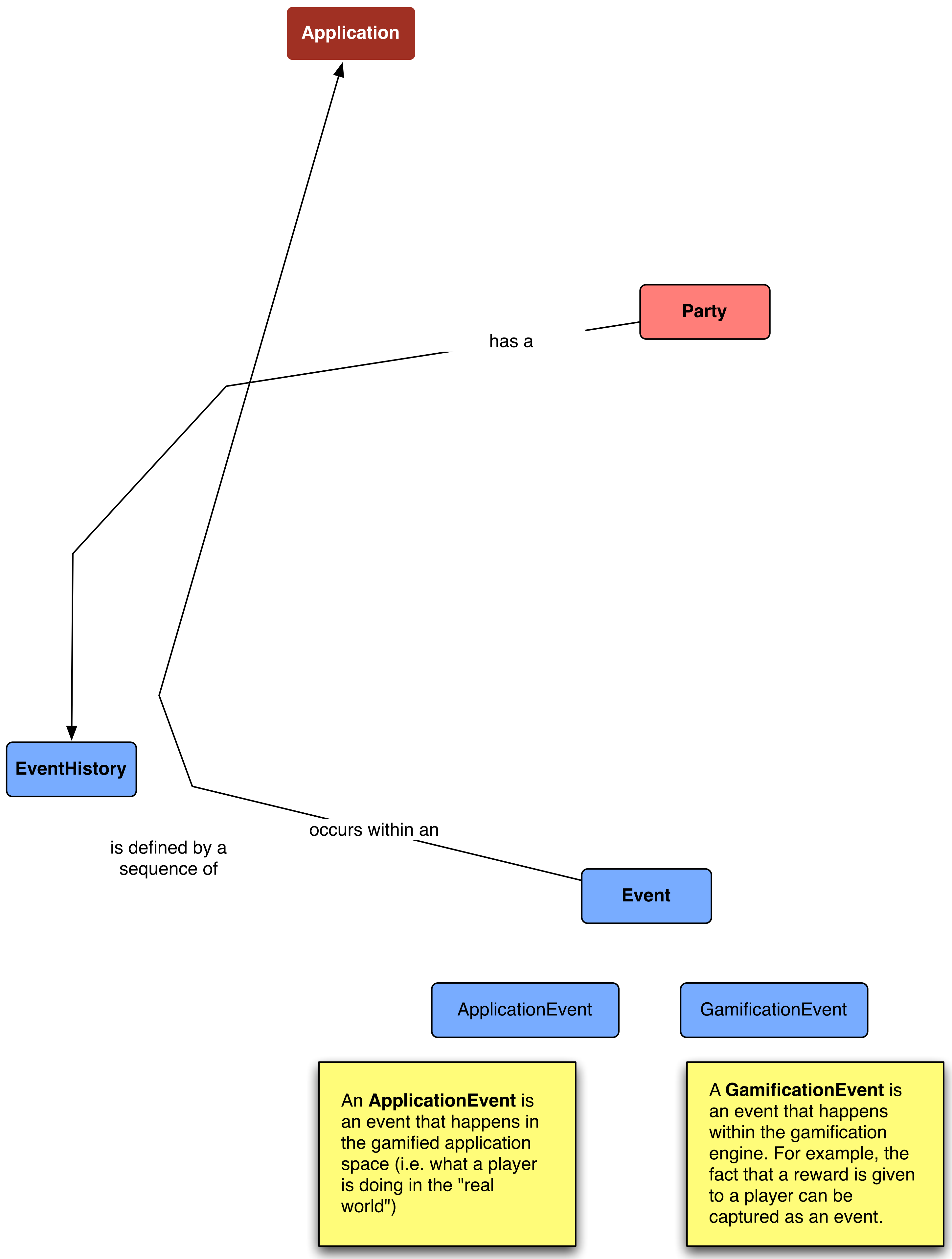


Both **players** and **teams** are **parties**, that can be ranked, that can receive rewards and penalties, etc.

A **player** is a person who is active in the gamified application space.

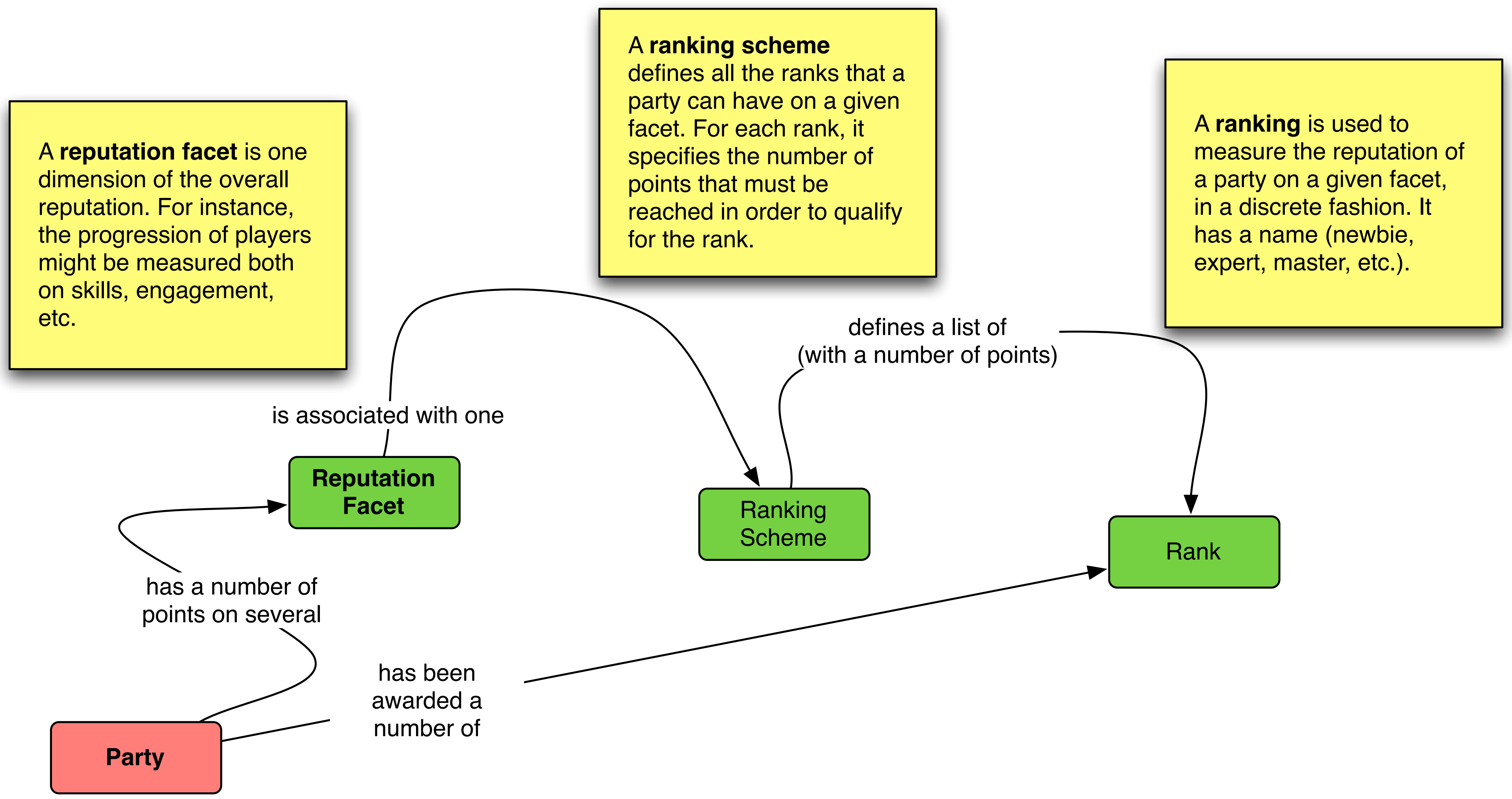
A **team** is a group of players.

It is interesting to **rank players** within a team. It is also interesting to **rank several teams**.



An **ApplicationEvent** is an event that happens in the gamified application space (i.e. what a player is doing in the "real world")

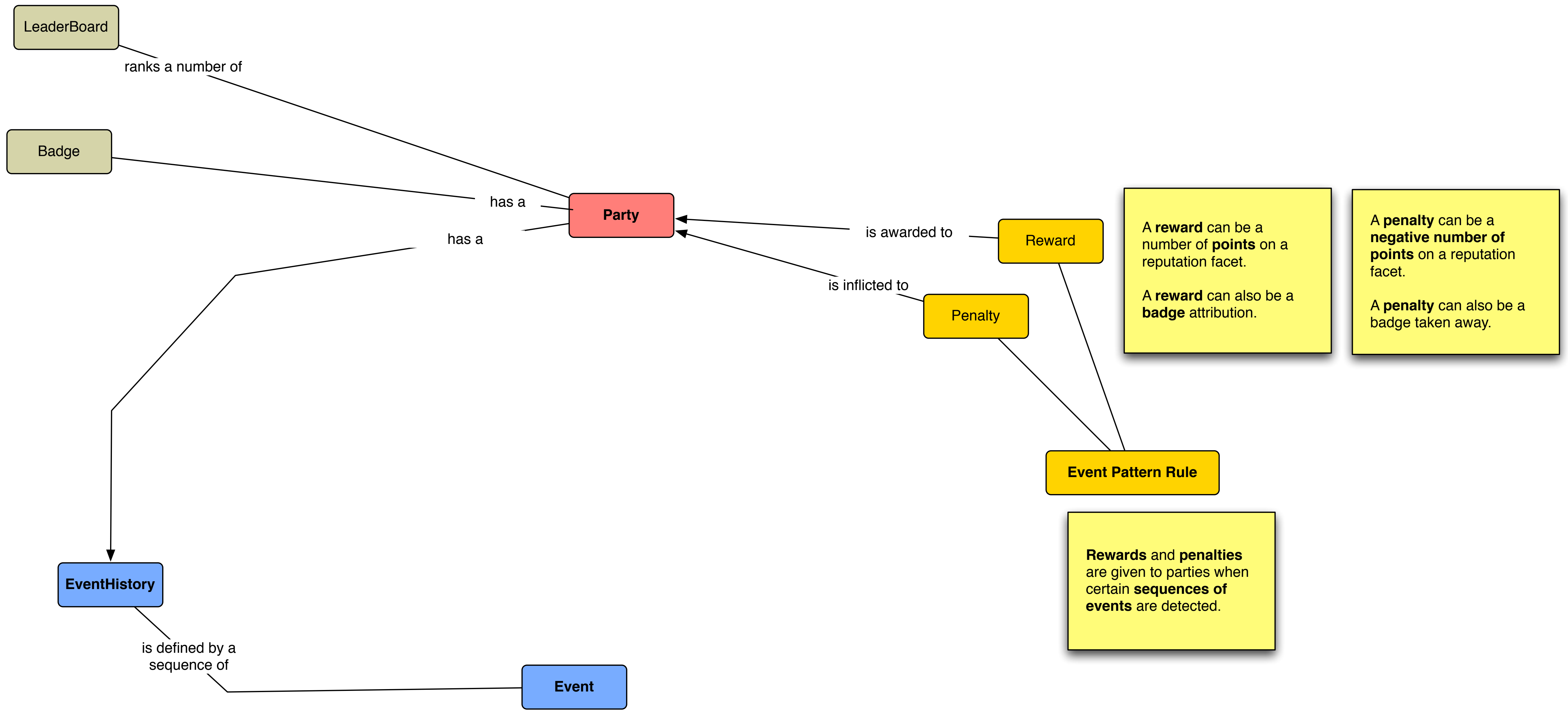
A **GamificationEvent** is an event that happens within the gamification engine. For example, the fact that a reward is given to a player can be captured as an event.


Reputation Facet
Ranking Scheme
Rank
Party
is associated with one
defines a list of (with a number of points)
has a number of points on several
has been awarded a number of

A **reputation facet** is one dimension of the overall reputation. For instance, the progression of players might be measured both on skills, engagement, etc.

A **ranking scheme** defines all the ranks that a party can have on a given facet. For each rank, it specifies the number of points that must be reached in order to qualify for the rank.

A **ranking** is used to measure the reputation of a party on a given facet, in a discrete fashion. It has a name (newbie, expert, master, etc.).



A **reward** can be a number of **points** on a reputation facet.

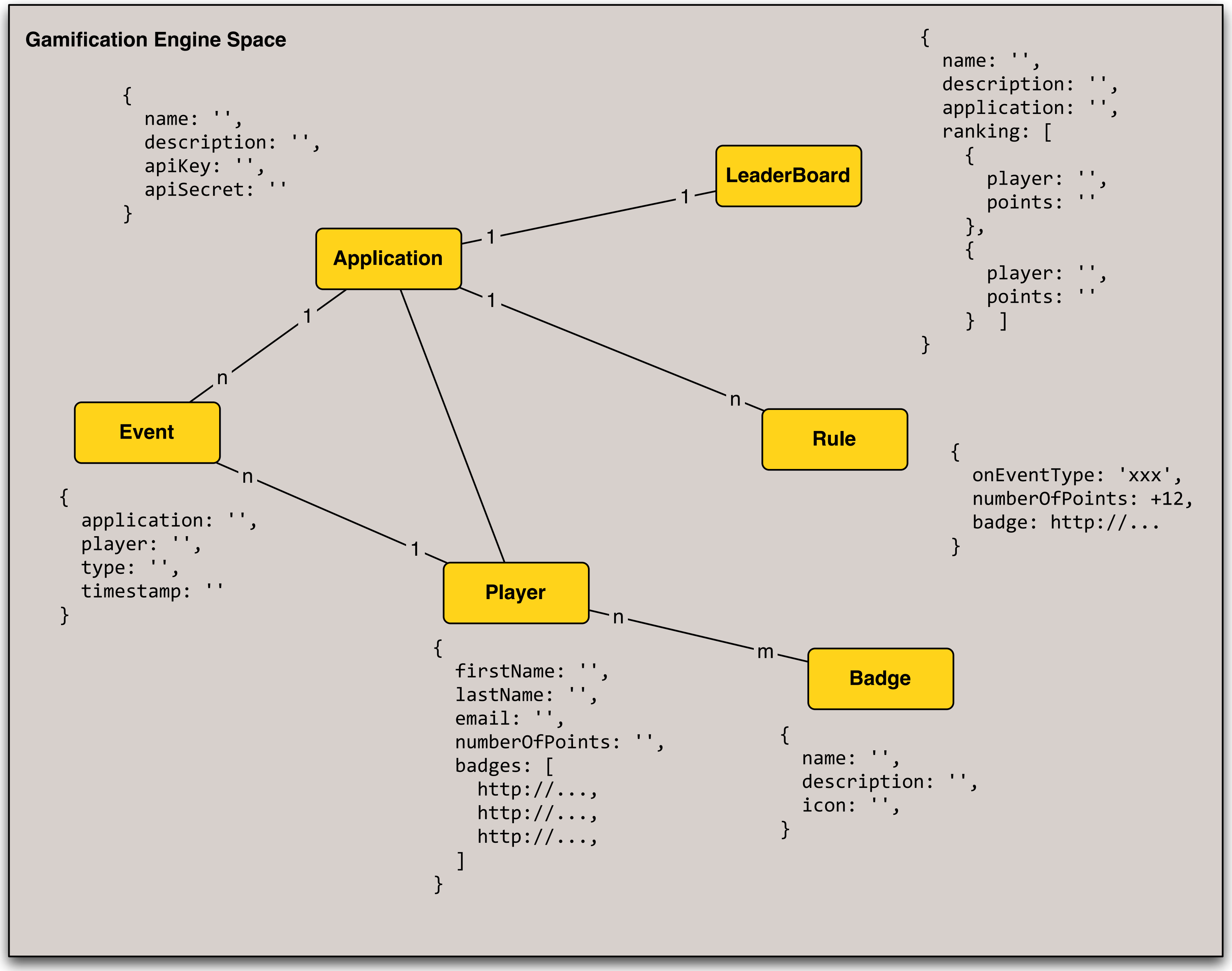
A **reward** can also be a **badge** attribution.

A **penalty** can be a **negative number of points** on a reputation facet.

A **penalty** can also be a badge taken away.

Rewards and **penalties** are given to parties when certain **sequences of events** are detected.

Gamification Domain Model
(simple version)



```
{
  name: '',
  description: '',
  apiKey: '',
  apiSecret: ''
}
```

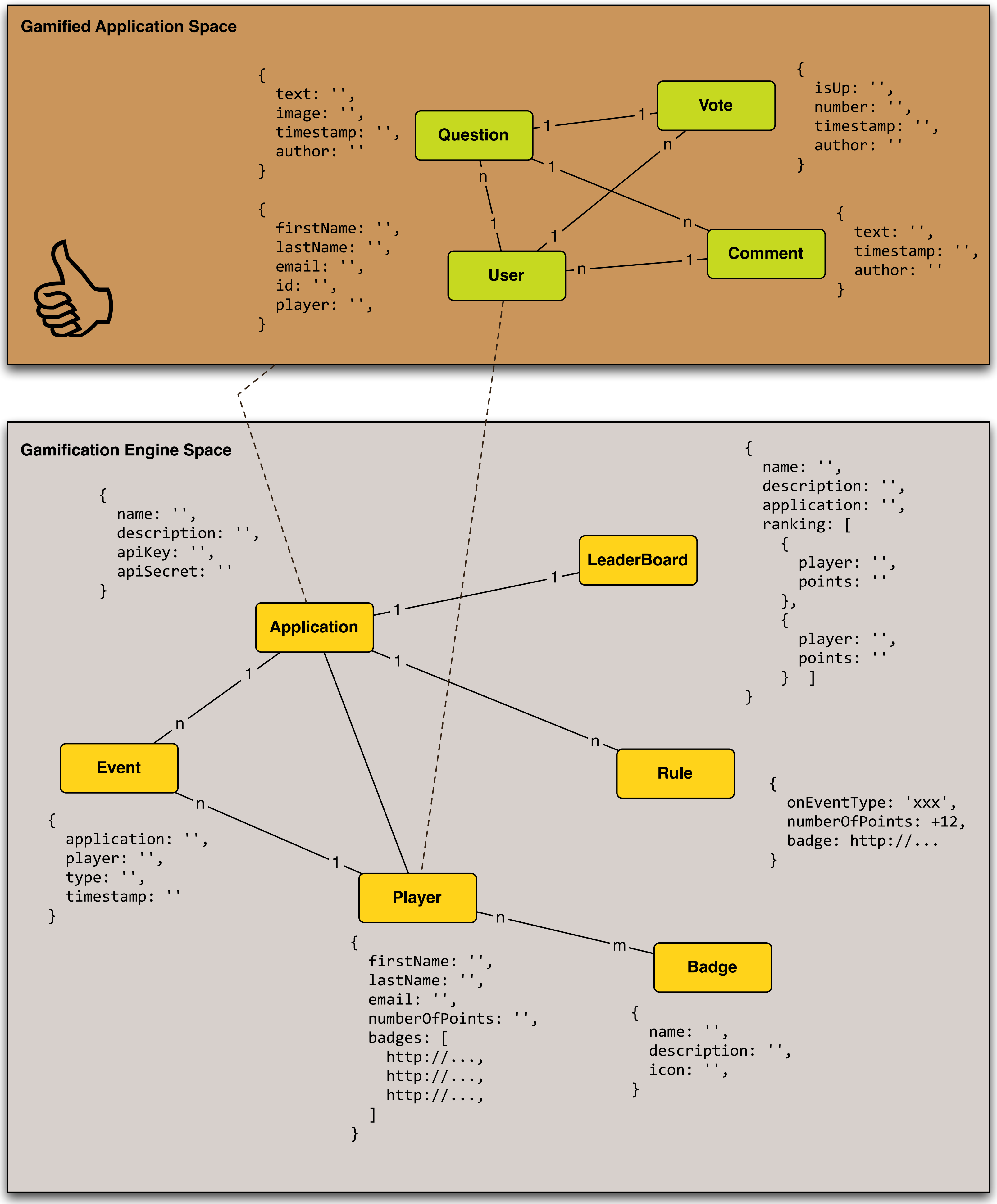
```
{
  name: '',
  description: '',
  application: '',
  ranking: [
    {
      player: '',
      points: ''
    },
    {
      player: '',
      points: ''
    }  ]
}
```

```
{
  onEventType: 'xxx',
  numberOfPoints: +12,
  badge: http://...
}
```

```
{
  application: '',
  player: '',
  type: '',
  timestamp: ''
}
```

```
{
  firstName: '',
  lastName: '',
  email: '',
  numberOfPoints: '',
  badges: [
    http://...,
    http://...,
    http://...,
  ]
}
```

```
{
  name: '',
  description: '',
  icon: '',
}
```



```
{
  text: '',
  image: '',
  timestamp: '',
  author: ''
}
```

```
{
  isUp: '',
  number: '',
  timestamp: '',
  author: ''
}
```

```
{
  firstName: '',
  lastName: '',
  email: '',
  id: '',
  player: '',
}
```

```
{
  text: '',
  timestamp: '',
  author: ''
}
```

```
{
  name: '',
  description: '',
  apiKey: '',
  apiSecret: ''
}
```

```
{
  name: '',
  description: '',
  application: '',
  ranking: [
    {
      player: '',
      points: ''
    },
    {
      player: '',
      points: ''
    }  ]
}
```

```
{
  onEventType: 'xxx',
  numberOfPoints: +12,
  badge: http://...
}
```

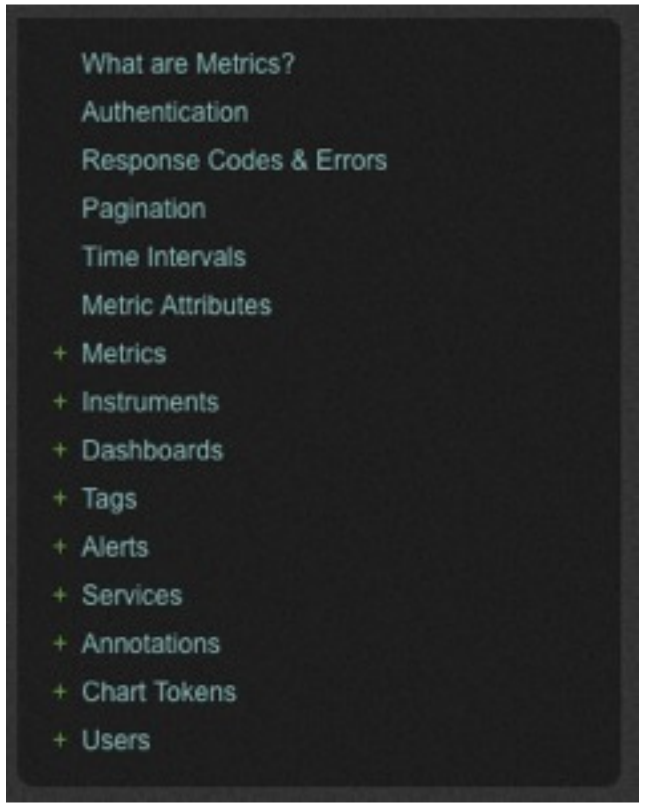
```
{
  application: '',
  player: '',
  type: '',
  timestamp: ''
}
```

```
{
  firstName: '',
  lastName: '',
  email: '',
  numberOfPoints: '',
  badges: [
    http://...,
    http://...,
    http://...,
  ]
}
```

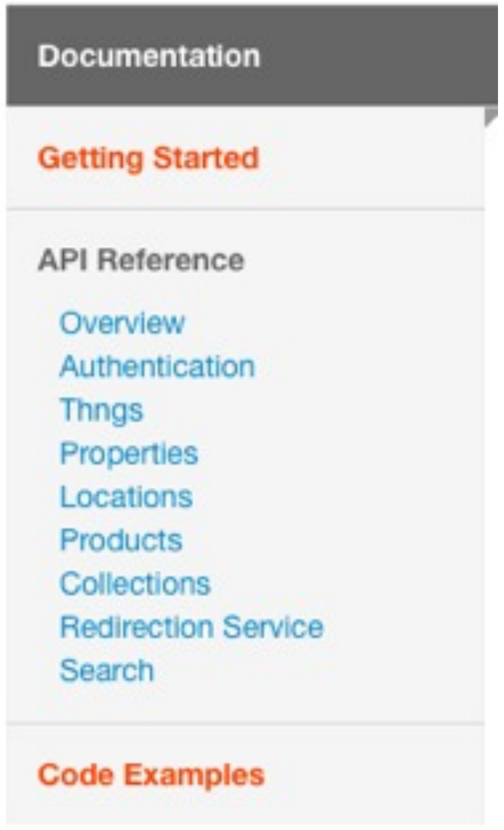
```
{
  name: '',
  description: '',
  icon: '',
}
```

How should I specify/document my REST API?

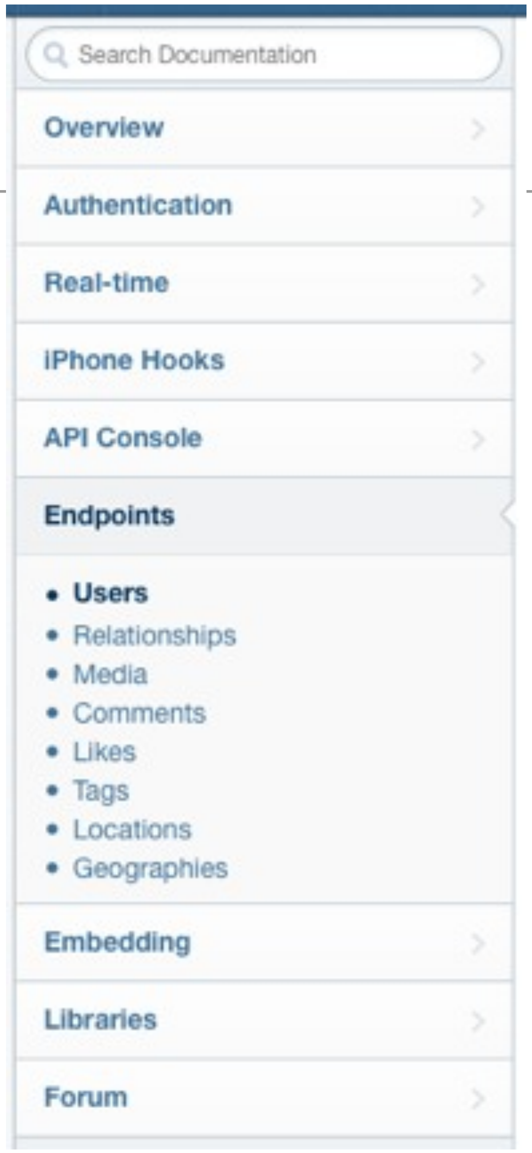
Look at Some Examples



What are Metrics?
Authentication
Response Codes & Errors
Pagination
Time Intervals
Metric Attributes
+ Metrics
+ Instruments
+ Dashboards
+ Tags
+ Alerts
+ Services
+ Annotations
+ Chart Tokens
+ Users



Documentation
Getting Started
API Reference
Overview
Authentication
Thngs
Properties
Locations
Products
Collections
Redirection Service
Search
Code Examples



Search Documentation
Overview
Authentication
Real-time
iPhone Hooks
API Console
Endpoints
- Users
- Relationships
- Media
- Comments
- Likes
- Tags
- Locations
- Geographies

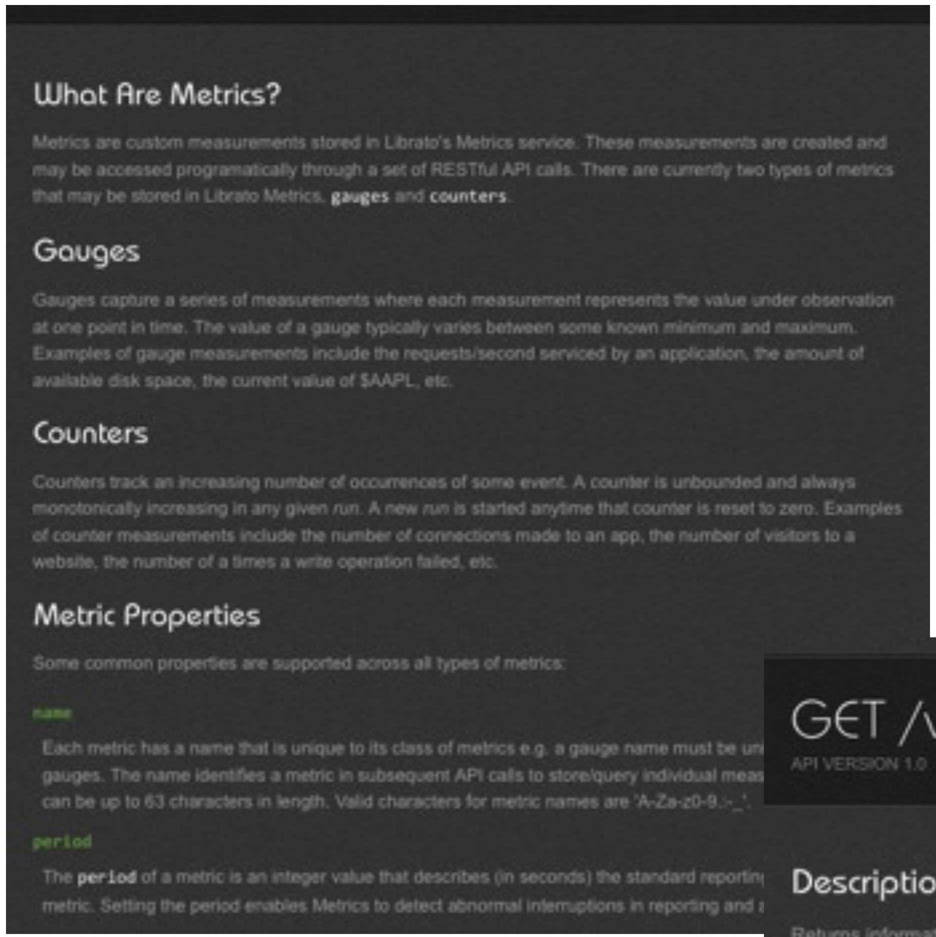
Embedding
Libraries
Forum

http://dev.librato.com/v1

https://dev.evrythng.com/documentation/api

http://instagram.com/developer/endpoints/

Short description of the resource (domain model)



What Are Metrics?

Metrics are custom measurements stored in Librato's Metrics service. These measurements are created and may be accessed programatically through a set of RESTful API calls. There are currently two types of metrics that may be stored in Librato Metrics, **gauges** and **counters**.

Gauges

Gauges capture a series of measurements where each measurement represents the value under observation at one point in time. The value of a gauge typically varies between some known minimum and maximum. Examples of gauge measurements include the requests/second serviced by an application, the amount of available disk space, the current value of $AAPL, etc.

Counters

Counters track an increasing number of occurrences of some event. A counter is unbounded and always monotonically increasing in any given *run*. A new *run* is started anytime that counter is reset to zero. Examples of counter measurements include the number of connections made to an app, the number of visitors to a website, the number of a times a write operation failed, etc.

Metric Properties

Some common properties are supported across all types of metrics:

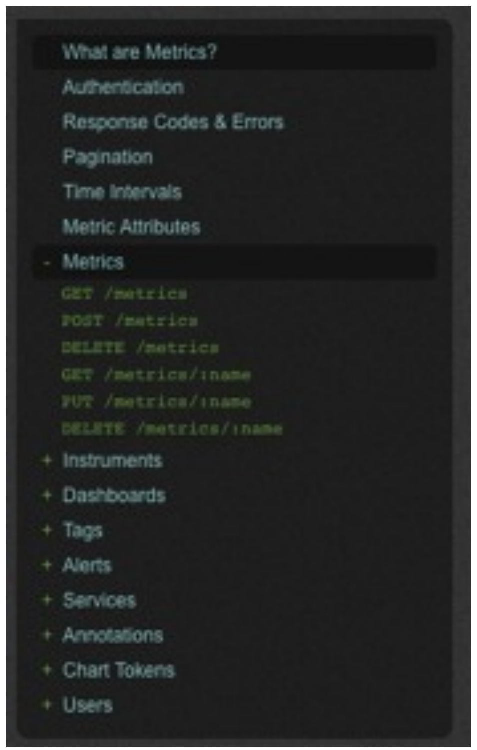
`name`

Each metric has a name that is unique to its class of metrics e.g. a gauge name must be un
gauges. The name identifies a metric in subsequent API calls to store/query individual meas
can be up to 63 characters in length. Valid characters for metric names are 'A-Za-z0-9.:-_'.

`period`

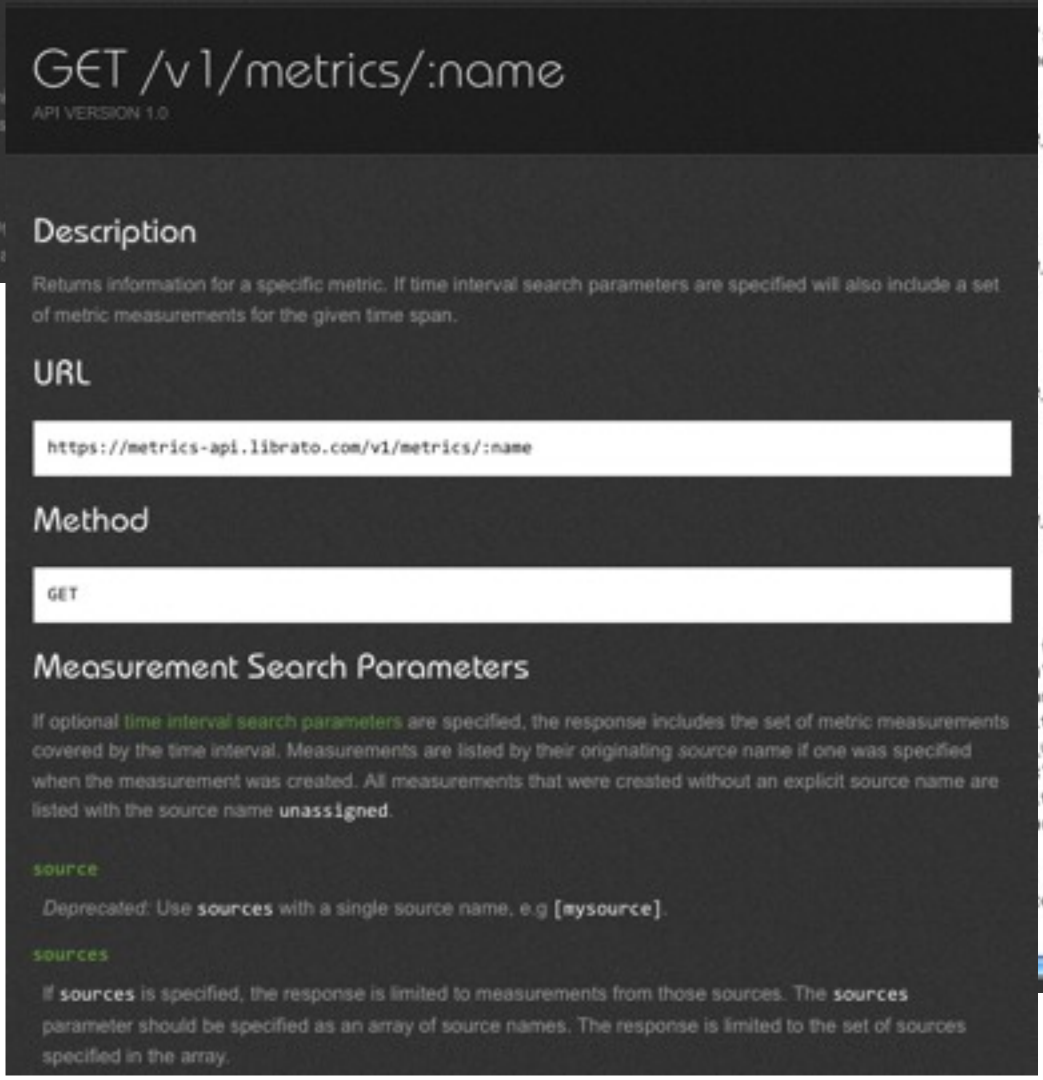
The **period** of a metric is an integer value that describes (in seconds) the standard reportin
metric. Setting the period enables Metrics to detect abnormal interruptions in reporting and a

navigator



- What are Metrics?
- Authentication
- Response Codes & Errors
- Pagination
- Time Intervals
- Metric Attributes
- - Metrics
  - `GET /metrics`
  - `POST /metrics`
  - `DELETE /metrics`
  - `GET /metrics/:name`
  - `PUT /metrics/:name`
  - `DELETE /metrics/:name`
- + Instruments
- + Dashboards
- + Tags
- + Alerts
- + Services
- + Annotations
- + Chart Tokens
- + Users

CRUD method description



GET /v1/metrics/:name

API VERSION 1.0

Description

Returns information for a specific metric. If time interval search parameters are specified will also include a set of metric measurements for the given time span.

URL

```
https://metrics-api.librato.com/v1/metrics/:name
```

Method

```
GET
```

Measurement Search Parameters

If optional time interval search parameters are specified, the response includes the set of metric measurements covered by the time interval. Measurements are listed by their originating *source* name if one was specified when the measurement was created. All measurements that were created without an explicit source name are listed with the source name **unassigned**.

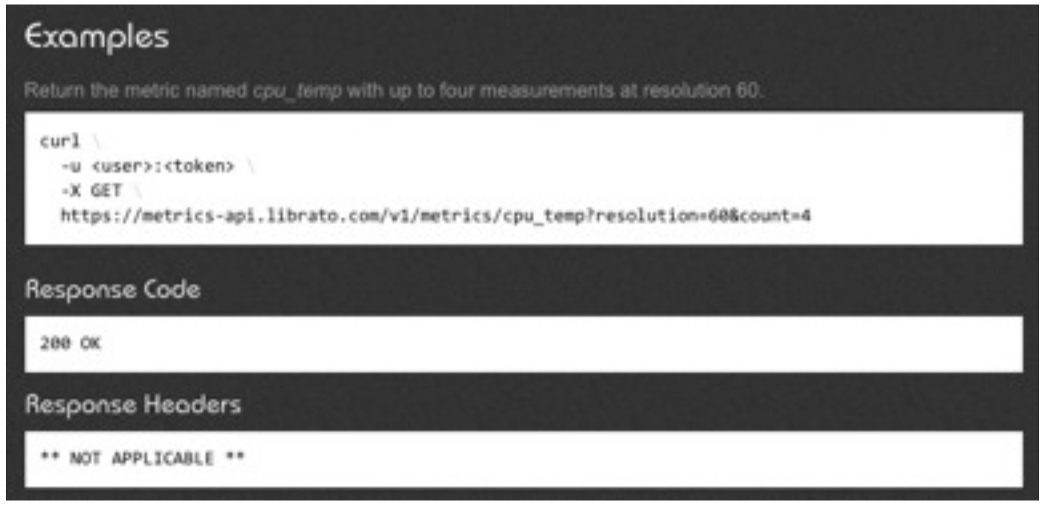
`source`

Deprecated: Use **sources** with a single source name, e.g **[mysource]**.

`sources`

If **sources** is specified, the response is limited to measurements from those sources. The **sources** parameter should be specified as an array of source names. The response is limited to the set of sources specified in the array.

Examples & payload structure



Examples

Return the metric named cpu_temp with up to four measurements at resolution 60.

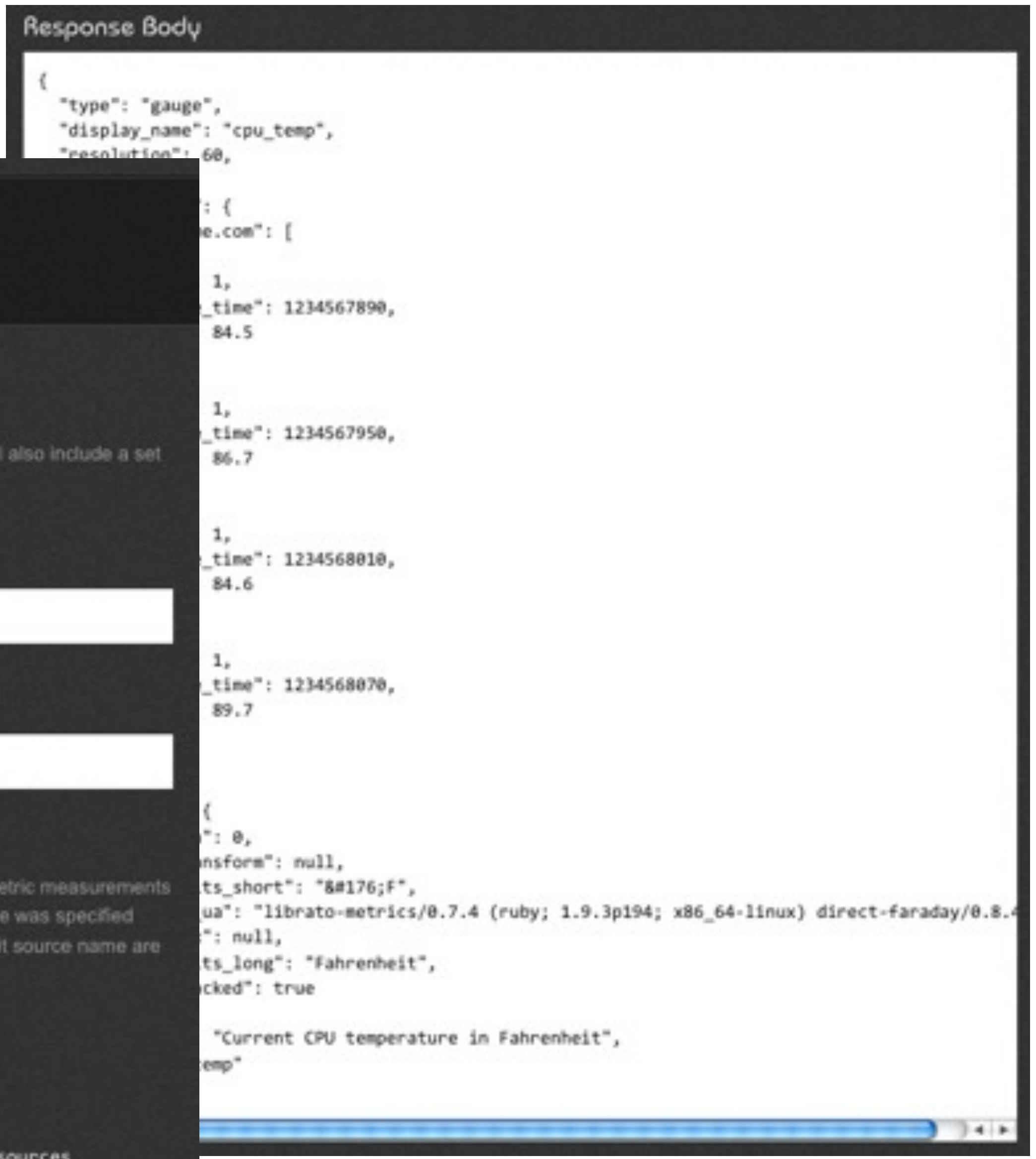
```
curl \
  -u <user>:<token> \
  -X GET \
  https://metrics-api.librato.com/v1/metrics/cpu_temp?resolution=60&count=4
```

Response Code

```
200 OK
```

Response Headers

```
** NOT APPLICABLE **
```

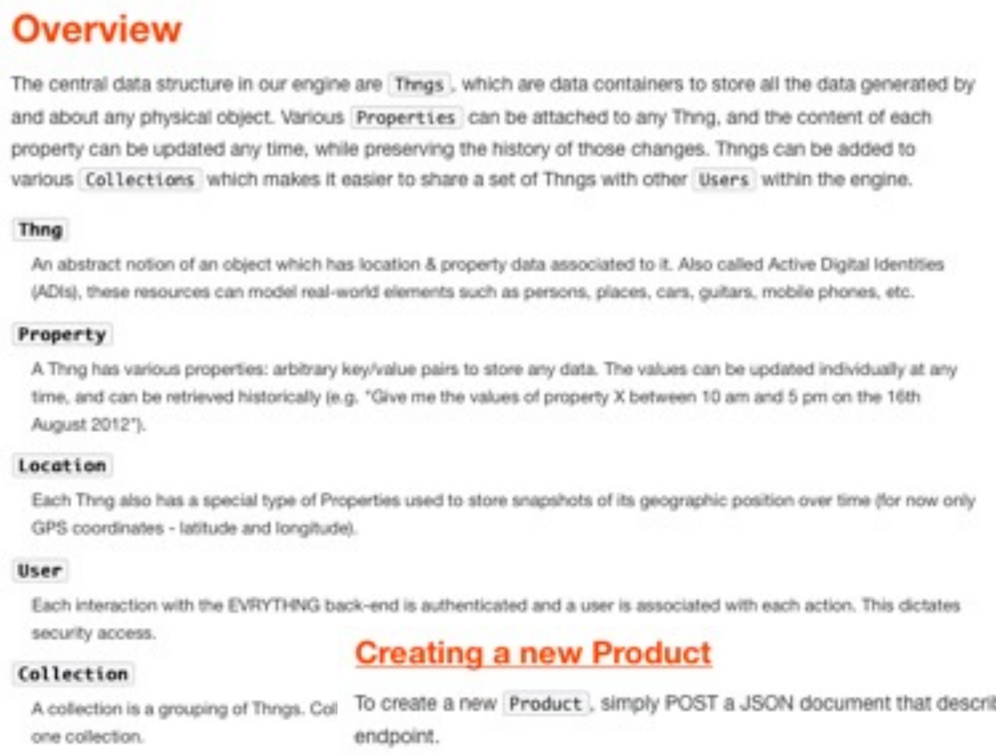


Response Body

```
{
  "type": "gauge",
  "display_name": "cpu_temp",
  "resolution": 60,

                 : {
                 e.com": [

                   1,
                 _time": 1234567890,
                   84.5


                   1,
                 _time": 1234567950,
                   86.7


                   1,
                 _time": 1234568010,
                   84.6


                   1,
                 _time": 1234568070,
                   89.7



                 {
                 ": 0,
                 nsform": null,
                 ts_short": "&#176;F",
                 ua": "librato-metrics/0.7.4 (ruby; 1.9.3p194; x86_64-linux) direct-faraday/0.8.4
                 ": null,
                 ts_long": "Fahrenheit",
                 cked": true

                   "Current CPU temperature in Fahrenheit",
                 emp"
```

Short description of the whole domain model

More details about the Product resource (domain model) & payload structure



Overview

The central data structure in our engine are `Thngs`, which are data containers to store all the data generated by and about any physical object. Various `Properties` can be attached to any Thng, and the content of each property can be updated any time, while preserving the history of those changes. Thngs can be added to various `Collections` which makes it easier to share a set of Thngs with other `Users` within the engine.

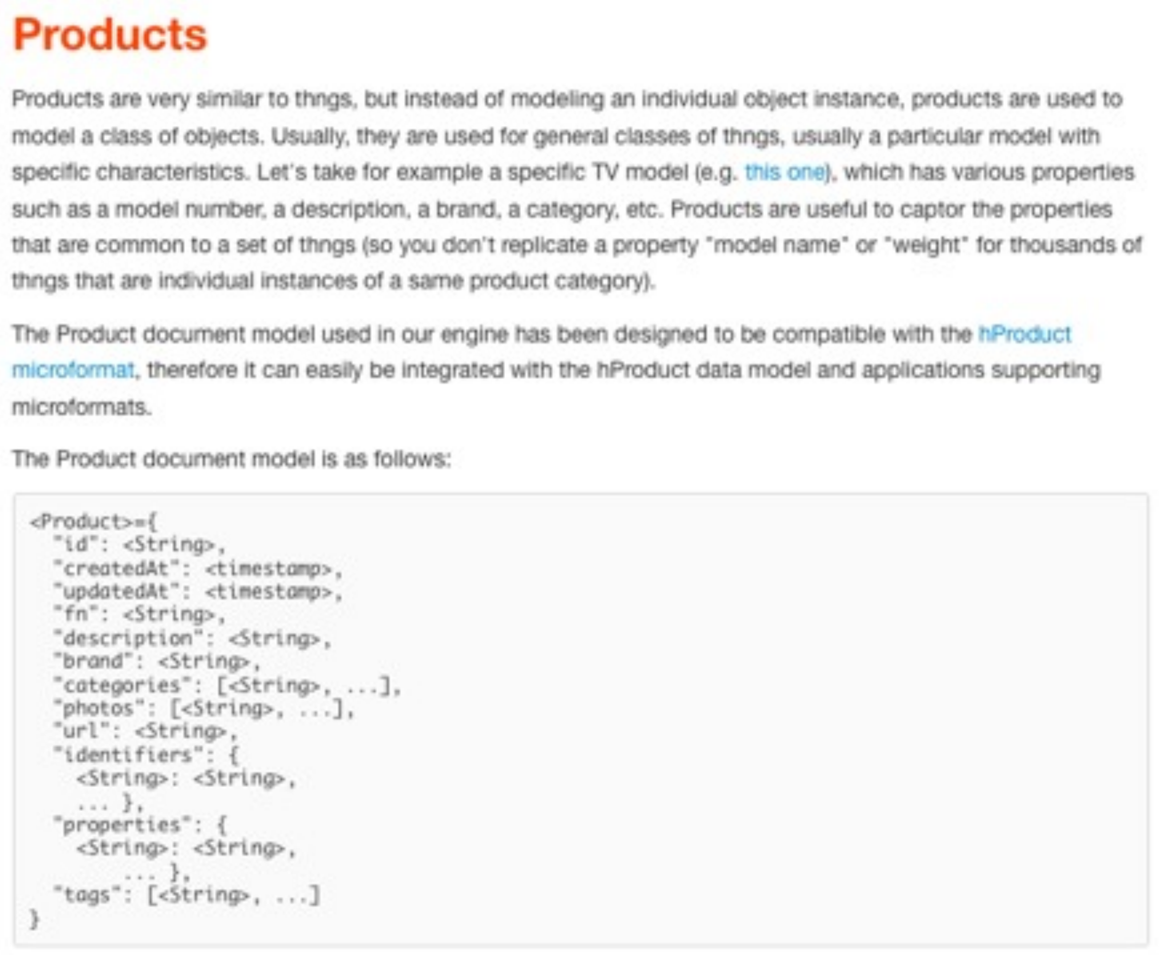
`Thng`
An abstract notion of an object which has location & property data associated to it. Also called Active Digital Identities (ADIs), these resources can model real-world elements such as persons, places, cars, guitars, mobile phones, etc.

`Property`
A Thng has various properties: arbitrary key/value pairs to store any data. The values can be updated individually at any time, and can be retrieved historically (e.g. "Give me the values of property X between 10 am and 5 pm on the 16th August 2012").

`Location`
Each Thng also has a special type of Properties used to store snapshots of its geographic position over time (for now only GPS coordinates - latitude and longitude).

`User`
Each interaction with the EVRYTHNG back-end is authenticated and a user is associated with each action. This dictates security access.

`Collection`
A collection is a grouping of Thngs. Col
one collection.



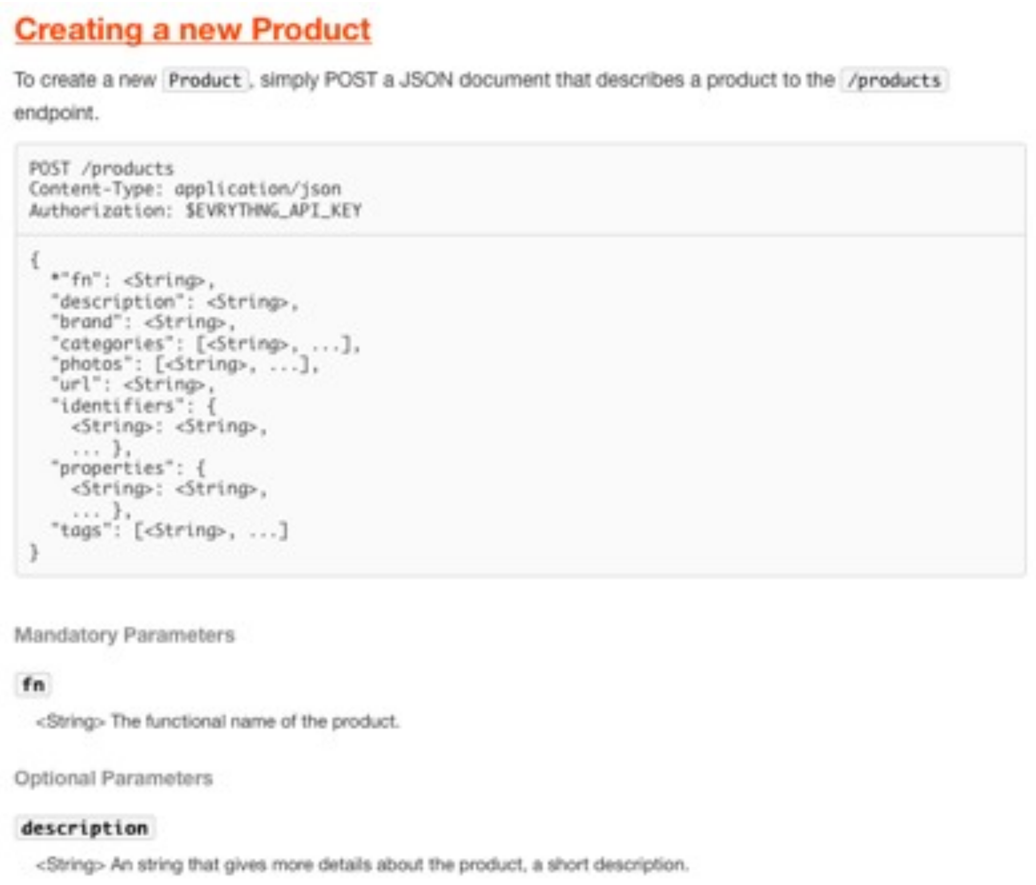
Products

Products are very similar to thngs, but instead of modeling an individual object instance, products are used to model a class of objects. Usually, they are used for general classes of thngs, usually a particular model with specific characteristics. Let's take for example a specific TV model (e.g. this one), which has various properties such as a model number, a description, a brand, a category, etc. Products are useful to captor the properties that are common to a set of thngs (so you don't replicate a property "model name" or "weight" for thousands of thngs that are individual instances of a same product category).

The Product document model used in our engine has been designed to be compatible with the hProduct microformat, therefore it can easily be integrated with the hProduct data model and applications supporting microformats.

The Product document model is as follows:

```
<Product>={
  "id": <String>,
  "createdAt": <timestamp>,
  "updatedAt": <timestamp>,
  "fn": <String>,
  "description": <String>,
  "brand": <String>,
  "categories": [<String>, ...],
  "photos": [<String>, ...],
  "url": <String>,
  "identifiers": {
    <String>: <String>,
    ... },
  "properties": {
    <String>: <String>,
      ... },
  "tags": [<String>, ...]
}
```



Creating a new Product

To create a new `Product`, simply POST a JSON document that describes a product to the `/products` endpoint.

```
POST /products
Content-Type: application/json
Authorization: $EVRYTHNG_API_KEY
```

```
{
  *"fn": <String>,
  "description": <String>,
  "brand": <String>,
  "categories": [<String>, ...],
  "photos": [<String>, ...],
  "url": <String>,
  "identifiers": {
    <String>: <String>,
    ... },
  "properties": {
    <String>: <String>,
    ... },
  "tags": [<String>, ...]
}
```

Mandatory Parameters

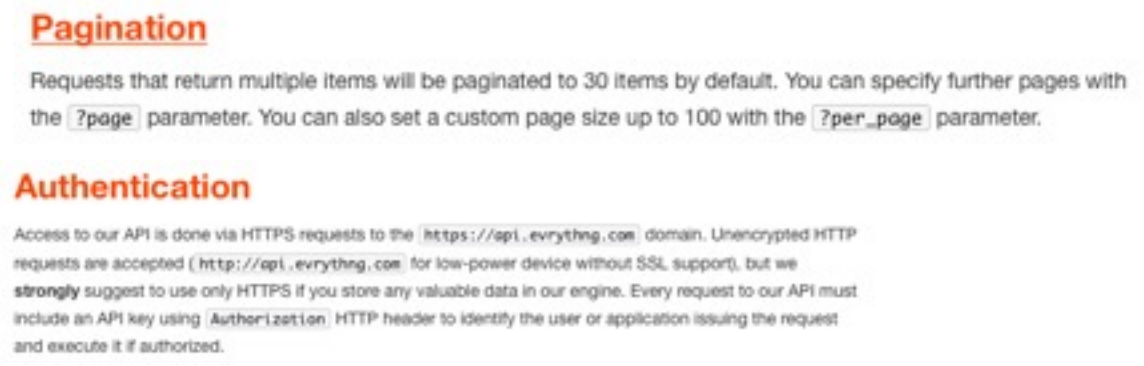
`fn`
<String> The functional name of the product.

Optional Parameters

`description`
<String> An string that gives more details about the product, a short description.

CRUD method description

Cross-cutting concerns



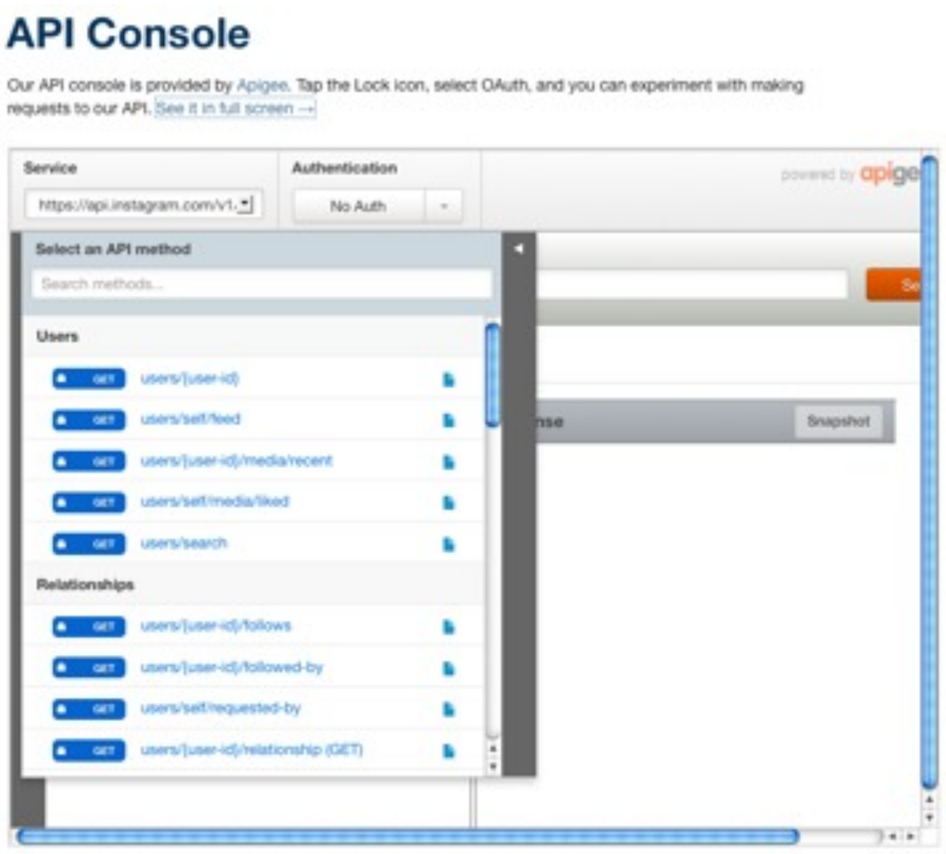
Pagination

Requests that return multiple items will be paginated to 30 items by default. You can specify further pages with the `?page` parameter. You can also set a custom page size up to 100 with the `?per_page` parameter.

Authentication

Access to our API is done via HTTPS requests to the `https://api.evrythng.com` domain. Unencrypted HTTP requests are accepted (`http://api.evrythng.com` for low-power device without SSL support), but we **strongly** suggest to use only HTTPS if you store any valuable data in our engine. Every request to our API must include an API key using `Authorization` HTTP header to identify the user or application issuing the request and execute it if authorized.

Interactive test console



API Console

Our API console is provided by Apigee. Tap the Lock icon, select OAuth, and you can experiment with making requests to our API. See it in full screen →

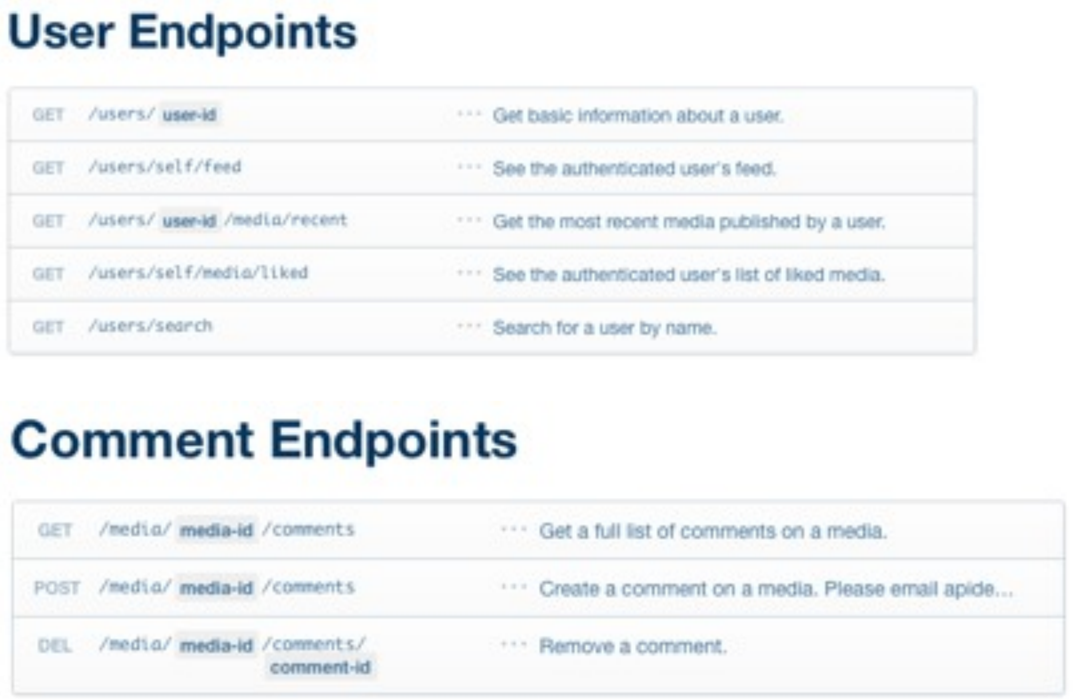
Service: https://api.instagram.com/v1.
Authentication: No Auth

Select an API method

Users
- GET users/[user-id]
- GET users/self/feed
- GET users/[user-id]/media/recent
- GET users/self/media/liked
- GET users/search

Relationships
- GET users/[user-id]/follows
- GET users/[user-id]/followed-by
- GET users/self/requested-by
- GET users/[user-id]/relationship (GET)

List of supported CRUD methods for each resource (R, R/W)

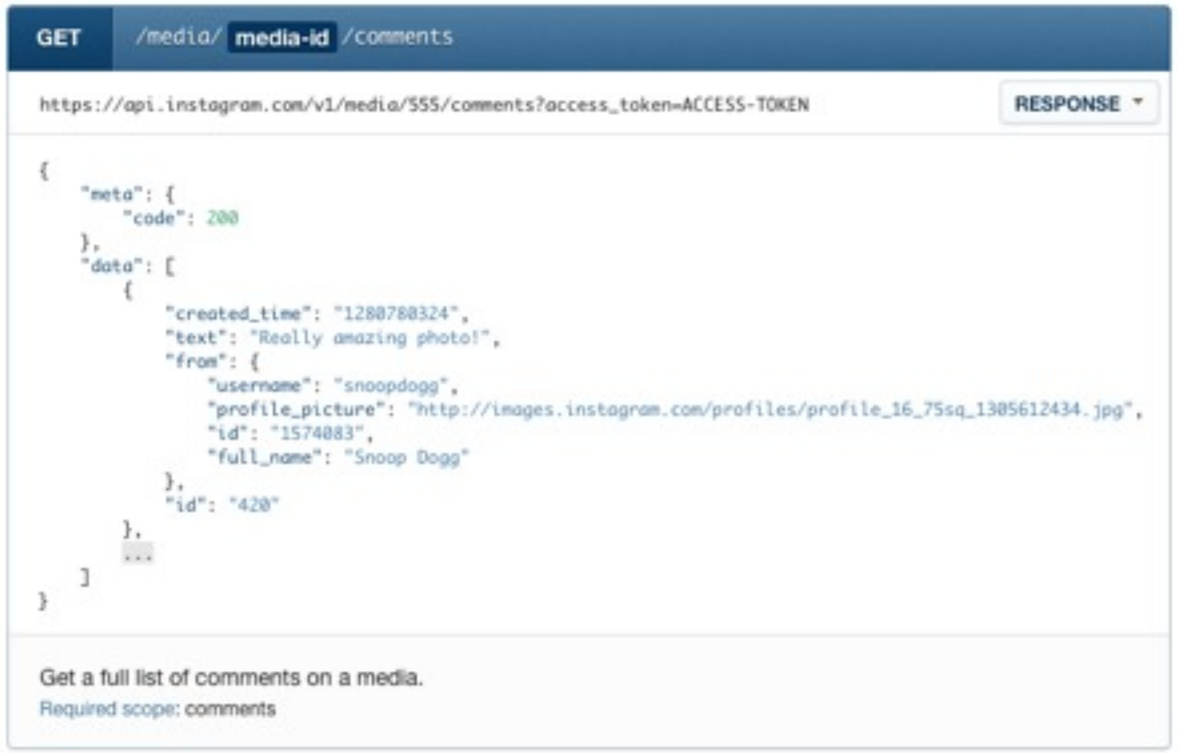


User Endpoints

GET	/users/ user-id	Get basic information about a user.
GET	/users/self/feed	See the authenticated user's feed.
GET	/users/ user-id /media/recent	Get the most recent media published by a user.
GET	/users/self/media/liked	See the authenticated user's list of liked media.
GET	/users/search	Search for a user by name.

Comment Endpoints

GET	/media/ media-id /comments	Get a full list of comments on a media.
POST	/media/ media-id /comments	Create a comment on a media. Please email apide...
DEL	/media/ media-id /comments/ comment-id	Remove a comment.



GET /media/ media-id /comments

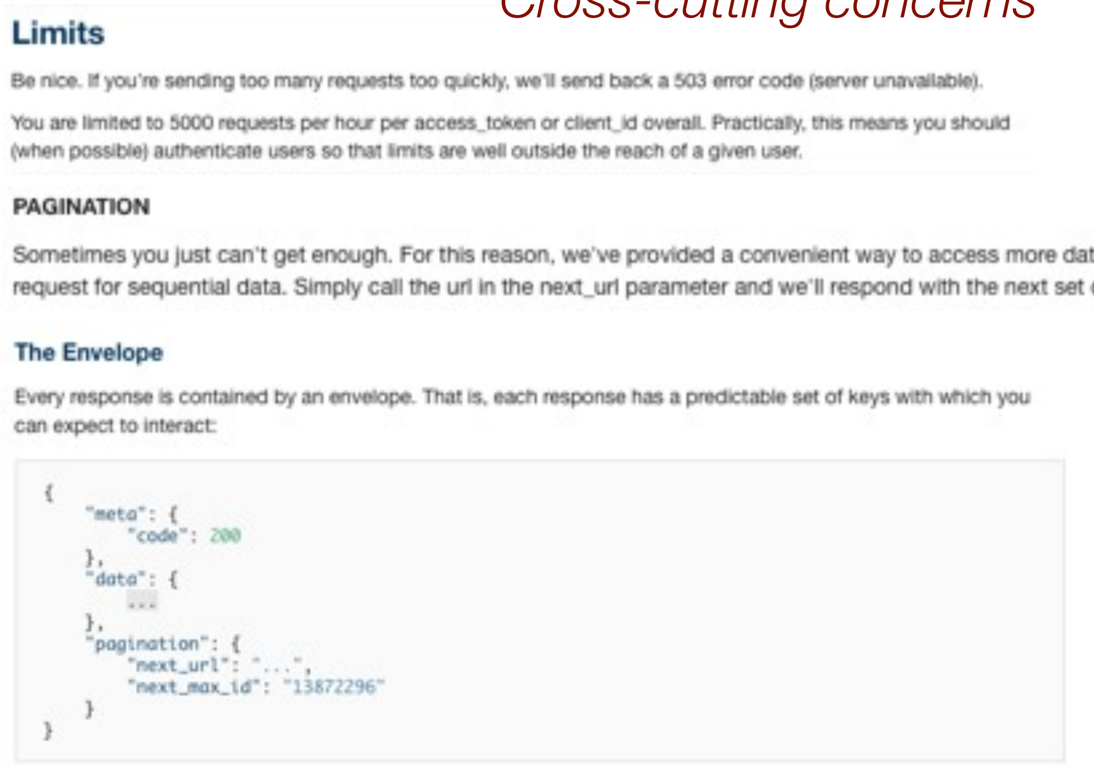
https://api.instagram.com/v1/media/555/comments?access_token=ACCESS-TOKEN RESPONSE

```
{
    "meta": {
        "code": 200
    },
    "data": [
        {
            "created_time": "1280780324",
            "text": "Really amazing photo!",
            "from": {
                "username": "snoopdogg",
                "profile_picture": "http://images.instagram.com/profiles/profile_16_75sq_1305612434.jpg",
                "id": "1574083",
                "full_name": "Snoop Dogg"
            },
            "id": "420"
        },
        ...
    ]
}
```

Get a full list of comments on a media.
Required scope: comments

CRUD method description

Cross-cutting concerns



Limits

Be nice. If you're sending too many requests too quickly, we'll send back a 503 error code (server unavailable).

You are limited to 5000 requests per hour per access_token or client_id overall. Practically, this means you should (when possible) authenticate users so that limits are well outside the reach of a given user.

PAGINATION

Sometimes you just can't get enough. For this reason, we've provided a convenient way to access more data in any request for sequential data. Simply call the url in the next_url parameter and we'll respond with the next set of data.

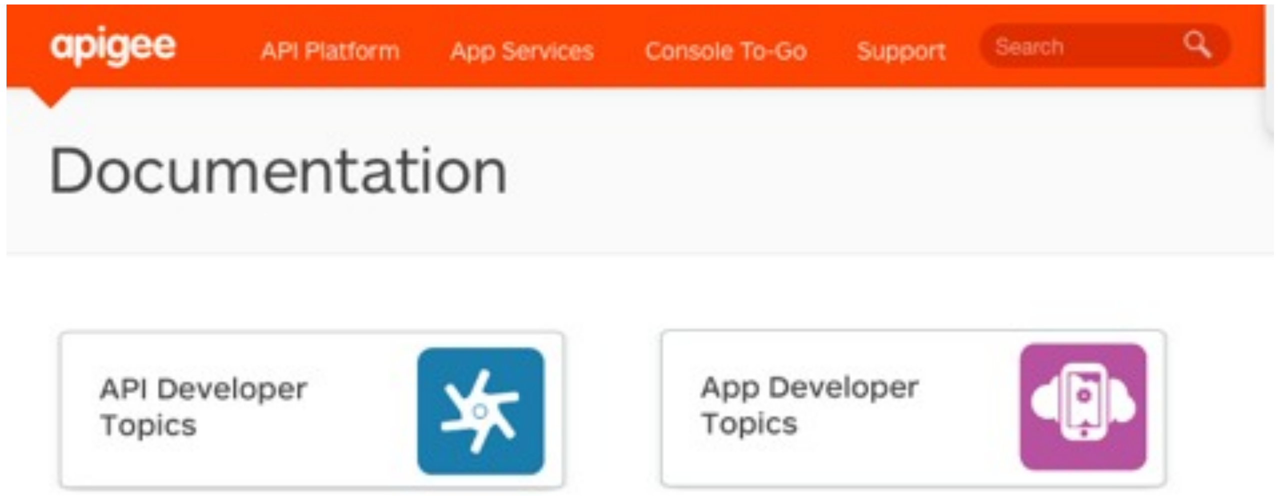
The Envelope

Every response is contained by an envelope. That is, each response has a predictable set of keys with which you can expect to interact:

```
{
    "meta": {
        "code": 200
    },
    "data": {
        ...
    },
    "pagination": {
        "next_url": "...",
        "next_max_id": "13872296"
    }
}
```

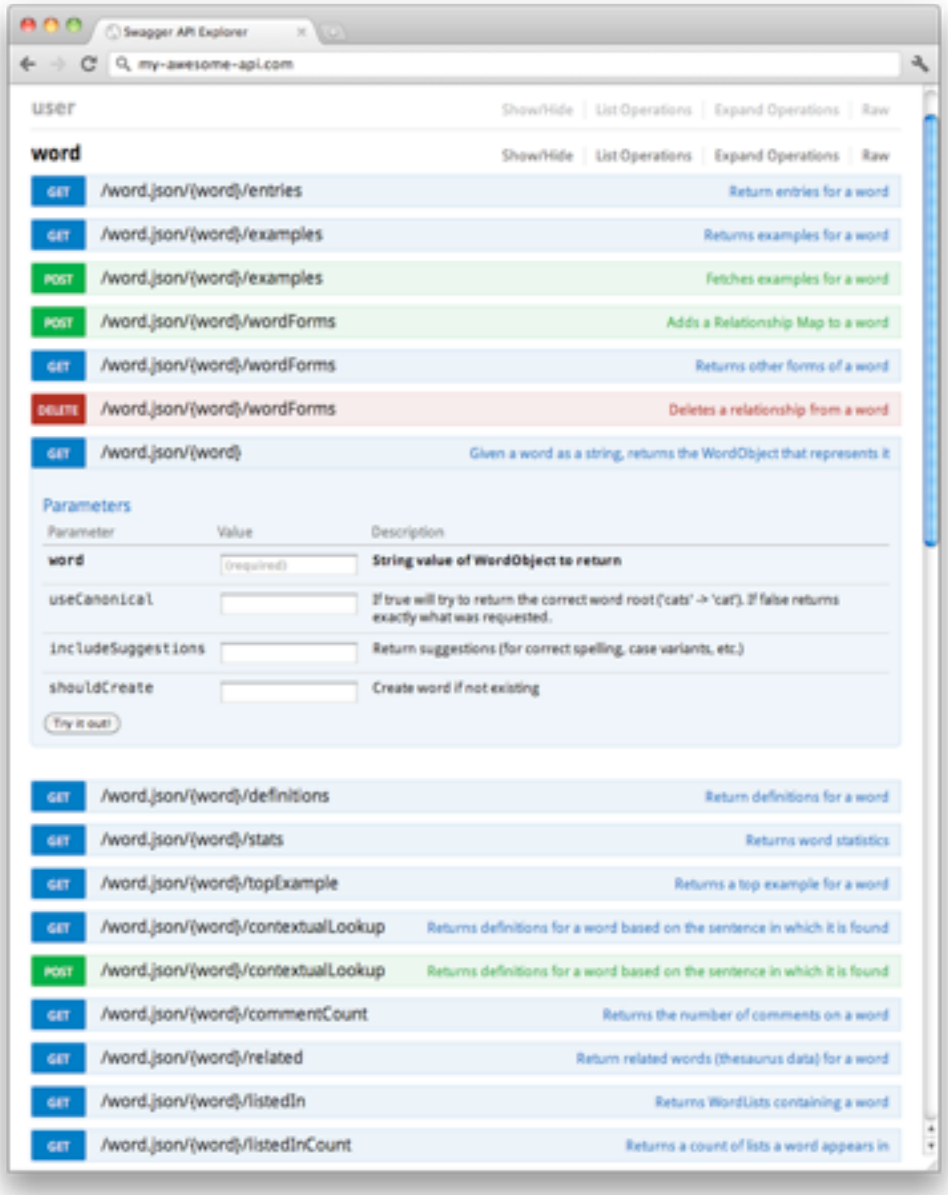
MSE MASTER OF SCIENCE IN ENGINEERING

Some Tools that Might Help/Inspire You



http://apigee.com/docs/

FOR DEVELOPERS
Swagger


Swagger API Explorer
my-awesome-api.com
user
Show/Hide | List Operations | Expand Operations | Raw
word
Show/Hide | List Operations | Expand Operations | Raw
Try it out!

GET	/word.json/{word}/entries	Return entries for a word
GET	/word.json/{word}/examples	Returns examples for a word
POST	/word.json/{word}/examples	Fetches examples for a word
POST	/word.json/{word}/wordForms	Adds a Relationship Map to a word
GET	/word.json/{word}/wordForms	Returns other forms of a word
DELETE	/word.json/{word}/wordForms	Deletes a relationship from a word
GET	/word.json/{word}	Given a word as a string, returns the WordObject that represents it

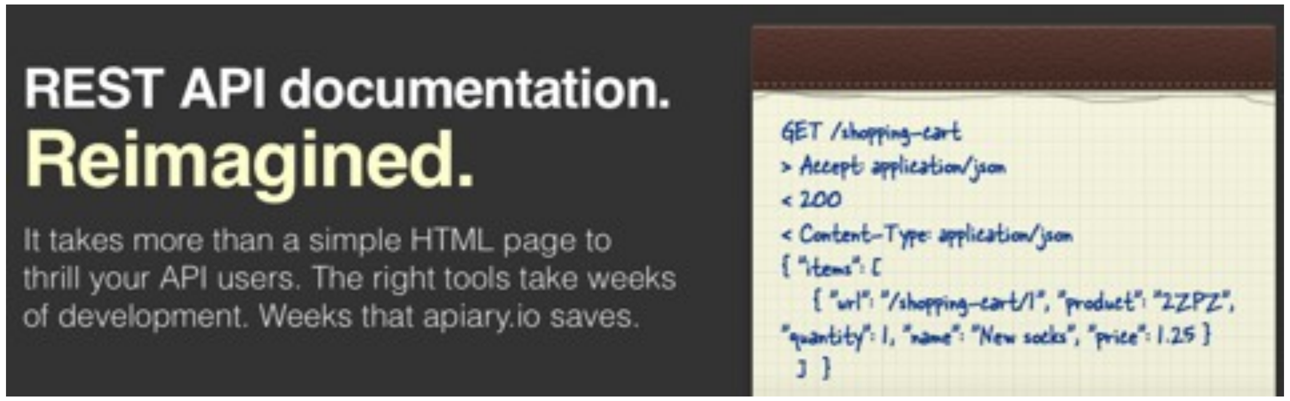
Parameters

Parameter	Value	Description
word	(required)	**String value of WordObject to return**
useCanonical		If true will try to return the correct word root ('cats' -> 'cat'). If false returns exactly what was requested.
includeSuggestions		Return suggestions (for correct spelling, case variants, etc.)
shouldCreate		Create word if not existing

GET	/word.json/{word}/definitions	Return definitions for a word
GET	/word.json/{word}/stats	Returns word statistics
GET	/word.json/{word}/topExample	Returns a top example for a word
GET	/word.json/{word}/contextualLookup	Returns definitions for a word based on the sentence in which it is found
POST	/word.json/{word}/contextualLookup	Returns definitions for a word based on the sentence in which it is found
GET	/word.json/{word}/commentCount	Returns the number of comments on a word
GET	/word.json/{word}/related	Return related words (thesaurus data) for a word
GET	/word.json/{word}/listedIn	Returns WordLists containing a word
GET	/word.json/{word}/listedInCount	Returns a count of lists a word appears in

https://developers.helloreverb.com/swagger/

apiary.io beta



REST API documentation.
Reimagined.

It takes more than a simple HTML page to thrill your API users. The right tools take weeks of development. Weeks that apiary.io saves.

```
GET /shopping-cart
> Accept: application/json
< 200
< Content-Type: application/json
{ "items": [
   { "url": "/shopping-cart/1", "product": "2ZPZ",
"quantity": 1, "name": "New socks", "price": 1.25 }
  ] }
```

http://apiary.io/