Explain REST using Trello

The goal to analyze a real world RESTful application and explain how it adheres to the 5 REST restraints, using Trello's web-services APIs. Bellow there are the 5 main principles.

REST Restraints

Client-Server Architecture

A REST architecture should always separate Server code from Client code. In case of Trello web application we can assume that is being done by:

Component	Role
Web Browser	Client
API Client	Client
Web Server	Server
Application Server	Server

Client and Server do not know this internal details, only the interface and message format / protocol they should use in order to communicate each other.

Layered System

I could not find prove that Trello's has a layered architecture, but as most of modern and hight demand applications it probably contains many applications and web servers that are probably scaled and called depending on user request.

Stateless

The server does not save client state during REST API interaction. It is the opposite of **Statefull** applications that generally persist in internal storage client's session data, for example.
With Trello's api the client needs to provide always its oAuth token and API key, so the server can do the authorization. The browser caches the user credentials, so server does not need to be aware of user state.
URI example:

```
POST
https://api.trello.com/1/boards/560bf4298b3dda300c18d09c?fields=name,url&key={YOUR-API-
KEY}&token={AN-OAUTH-TOKEN}
```

Clients can cache responses:

Web browser caches the user credentials as cookies, so user can stay logged in even without server knowing client's state. The same can be done with API client using cache headers.

Uniform Communication Interface

The server must provide a uniform interface for communication with client.
API must use common HTTP methods: Lets take the *board* resource as example:
POST: Create a new *board*

```
POST /1/boards/?name=name+here
```

PUT: Update an existing *board*

```
PUT /1/boards/123456/name?value=new+name+here
```

GET: Get *board* by identifier:

```
 GET /1/boards/123456
```

DELETE: Remove a *board*

```
DELETE /1/boards/123456
```

The resource must be identifiable in the URI. Example: Get *board* by identifier:

```
   GET /1/boards/123456
```

Where *boards* is the resource and **123456** is the resource identifier.
Uses *nouns* to build resource URIs: As you can see in the API documentation, boards, cards, labels, members, etc are all resources and *nouns*.
Unified response format: Trello uses JSON messages format as response. Example:
Request example:

```
  curl --request POST \
  --url
'https://api.trello.com/1/boards/?name=name&defaultLabels=true&defaultLists=true&keepFr
omSource=none&prefs_permissionLevel=private&prefs_voting=disabled&prefs_comments=member
s&prefs_invitations=members&prefs_selfJoin=true&prefs_cardCovers=true&prefs_background=
blue&prefs_cardAging=regular&key=SECRET&token=SECRET'
```

Response example:

```
{
    "id": "123456",
    "name": "name",
    "desc": "",
    "descData": null,
    "closed": false,
    "idOrganization": null,
    "idEnterprise": null,
    "pinned": false,
    "url": "https://trello.com/b/RWoxB8KB/name",
    "shortUrl": "https://trello.com/b/RWoxB8KB",
    "prefs": {
        "permissionLevel": "private",
        "hideVotes": false,
        "voting": "disabled",
        "comments": "members",
        "invitations": "members",
        "selfJoin": true,
        "cardCovers": true,
        "isTemplate": false,
        "cardAging": "regular",
        "calendarFeedEnabled": false,
        "background": "blue",
        "backgroundImage": null,
        "backgroundImageScaled": null,
        "backgroundTile": false,
        "backgroundBrightness": "dark",
        "backgroundColor": "#0079BF",
        "backgroundBottomColor": "#0079BF",
        "backgroundTopColor": "#0079BF",
        "canBePublic": true,
        "canBeEnterprise": true,
        "canBeOrg": true,
        "canBePrivate": true,
        "canInvite": true
    },
    "labelNames": {
        "green": "",
        "yellow": "",
        "orange": "",
        "red": "",
        "purple": "",
        "blue": "",
        "sky": "",
        "lime": "",
        "pink": "",
        "black": ""
    }
}
```

REST Design Principles

Use only nouns for a URI

URI should not use verbs, only *nouns*, otherwise they will not be **resource based**. Let the HTTP methods be the *verbs* when define your REST API.

Analyzing Trello's API documentation, there are only *nouns* used to identify the resources. Examples:

```
GET /boards/{id}
DELETE /customFields/{id}
```

Although there are some URIs not using sub resources in plural. Examples: checklist, checkItem, avatars.

```
POST /members/{id}/avatar
PUT /cards/{idCard}/checklist/{idChecklist}/checkItem/{idCheckItem}
```

GET methods should not alter the state of resource

The HTTP GET must be only a **Query** method, which means, it can be used to search and retrieve a single or collection of resources, but never change resource's state.

The only HTTP methods allowed to do that are **PUT**, **POST** or **DELETE**.

Use plural nouns for a URI

As mentioned before, it is a convention to use always plural for resources and sub-resources on REST apis, but some exceptions were found in Trello's API, such as:

```
POST /members/{id}/avatar
PUT /cards/{idCard}/checklist/{idChecklist}/checkItem/{idCheckItem}
```

Use sub-resources for relationships between resources

Always when we need to get a resource related to another. It is preferable to use sub-resources rather than main resource as query string. It is more declarative.

Examples found in Trello's api:

```
GET /cards/{id}/stickers/{idSticker}
GET /members/{id}/boards
```

Use HTTP headers to specify input/output format

In REST apis we can specify specific headers to identify input/output format. These would be:

Content-Type: The request input format send to the server in the request body.

Accept: The output format accepted in the response by the client.

In case of Trello's API, there is no need to provide neither of the headers above, cause Trello supports only JSON format and will always return objects with this format.

But for the input, Trello's API only works with **query string** parameters even for write HTTP methods like POST and DELETE. Example: POST /boards/.

```
curl --request POST \
  --url
'https://api.trello.com/1/boards/?name=name&defaultLabels=true&defaultLists=true&keepFr
omSource=none&prefs_permissionLevel=private&prefs_voting=disabled&prefs_comments=member
s&prefs_invitations=members&prefs_selfJoin=true&prefs_cardCovers=true&prefs_background=
blue&prefs_cardAging=regular&key=SECRET&token=SECRET'
```

It is inconsistent in my opinion.

Provide users with filtering and paging for collections

To avoid overload client and server with high volume and unnecessary resources, the API server should provide some sort of pagination. It is generally done by usage of query strings like *limit* and *offset* or *page*.

There are some cases that there is no pagination available, only limit of resources returned. Example:

GET /boards/{id}/labels?fields=all&limit=100

So the API could be optimized for those cases.

Version the API

According to the first segment of URI, Trello's API is versioned and its current version is **1**.

```
https://api.trello.com/1/boards
```

Provide proper HTTP status codes

A RESTful API must return proper status according to action executed and the generated result. Example:

HTTP Method	**action**	**response**	
GET	/resources/{id}	404	Resource not found
GET	/resources/{id}	200	Resource found
POST	/resources	201	Resource created
POST	/resources	400	Invalid request
PUT	/resources/{id}	200	Resource updated
DELETE	/resources/{id}	204	Resource removed

The Trello's API as the Web application are providing the prover status code in the response, but I noticed some inconsistencies. For example, when creating a resource, the API returns status code *200* instead of *201*.