

REST API

OVERVIEW

Design and of Web APIs using the REST paradigm.



POLITECNICO
DI TORINO



Goal

- How to use REST architectures to integrate (call and/or offer) remote services
- How to design a consistent set of REST APIs
- How to implement REST APIs in python/Flask

Summary

- REST (Representational State Transfer)
- Rest API Design Guidelines
- Implementing REST APIs in Python

{ REST }

REpresentational State Tranfer

REST

REST

- **Representational State Transfer**

- A style of software architecture for distributed systems
- Platform-independent
 - you don't care if the server is Unix, the client is a Mac, or anything else
- Language-independent
 - C# can talk to Java, etc.
- Standards-based
 - runs on top of HTTP
- Can easily be used in the presence of firewalls



Roy T. Fielding

Senior Principal Scientist, [Adobe](#)

Co-founder, [Apache HTTP Server Project](#)

Director, [The Apache Software Foundation](#)

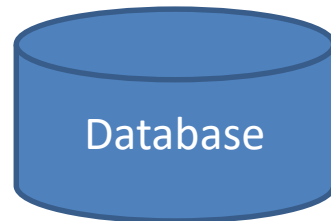
Ph.D., [Information and Computer Science, UC Irvine](#)

- [@fielding](#); Blog: [Untangled](#)
- Email: fielding at (choose **one** of) gbiv.com, adobe.com, apache.org

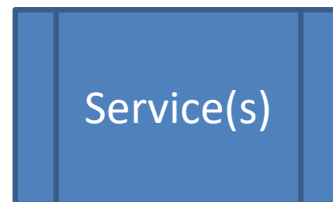
REST Architecture

Application

- Web backend
- Web frontend
- IoT device
- Mobile app



Database



Service(s)

What is a Resource?

- A resource can be anything that has identity
 - a document or image
 - a service, e.g., "today's weather in New York"
 - a collection of other resources
 - non-networked objects (e.g., people)
- The resource is the **conceptual mapping** to an entity or set of entities, not necessarily the entity that corresponds to that mapping at any particular point in time!

REST Architecture

Application

- Web backend
- Web frontend
- IoT device
- Mobile app

Resource(s)

Database

Service(s)

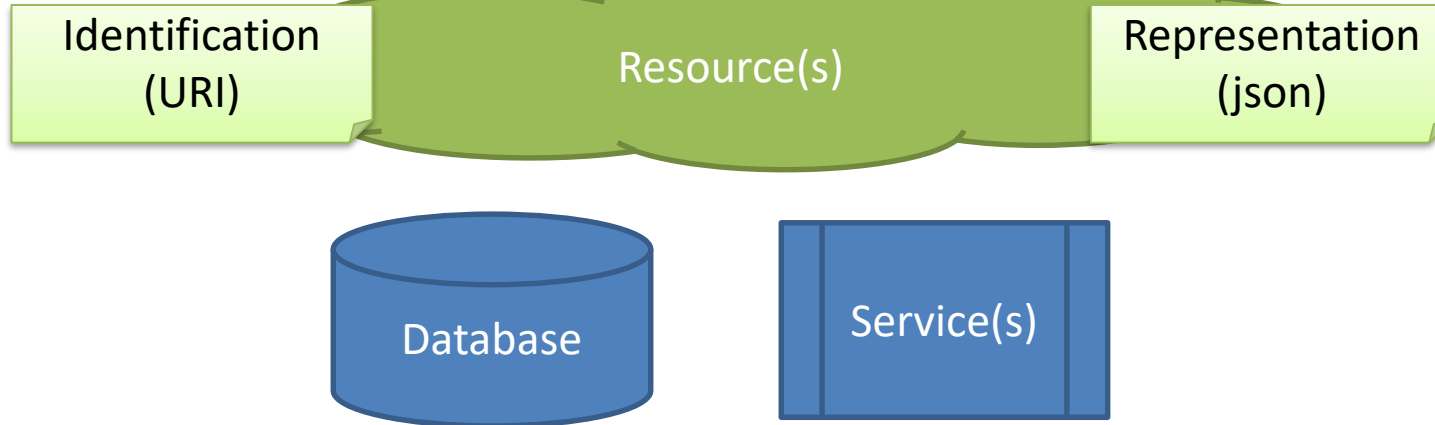
Main Principles

- Resource: source of specific information
- Mapping: Resources \Leftrightarrow URIs
- Client and server exchange *representations* of the resource
 - the same resource *may* have different representations
 - e.g., XML, JSON, HTML, RDF, ...

REST Architecture

Application

- Web backend
- Web frontend
- IoT device
- Mobile app



Main Types of Resources

- **Collection** resource

- Represents a set (or list) of resources of the same type
- Format: /resource
 - `http://api.polito.it/students`
 - `http://api.polito.it/courses`



- **Element** (Item, Simple) resource

- Represents a single item, and its properties
- Has some state and zero or more sub-resources
 - Sub-resources can be simple resources or collection resources
- Format: /resource/identifier
 - `http://api.polito.it/students/s123456`
 - `http://api.polito.it/courses/01zqp`



Best Practice

- Nouns (not verbs)
- Plural nouns
- Concrete names (not abstract)
 - /courses, not /items

Main Principles

- Resources support Operations (Actions)
 - Add
 - Delete
 - Update
 - Find
 - Search
 - ...

REST Architecture

Application

- Web backend
- Web frontend
- IoT device
- Mobile app

Operations



Identification
(URI)

Resource(s)

Representation
(json)

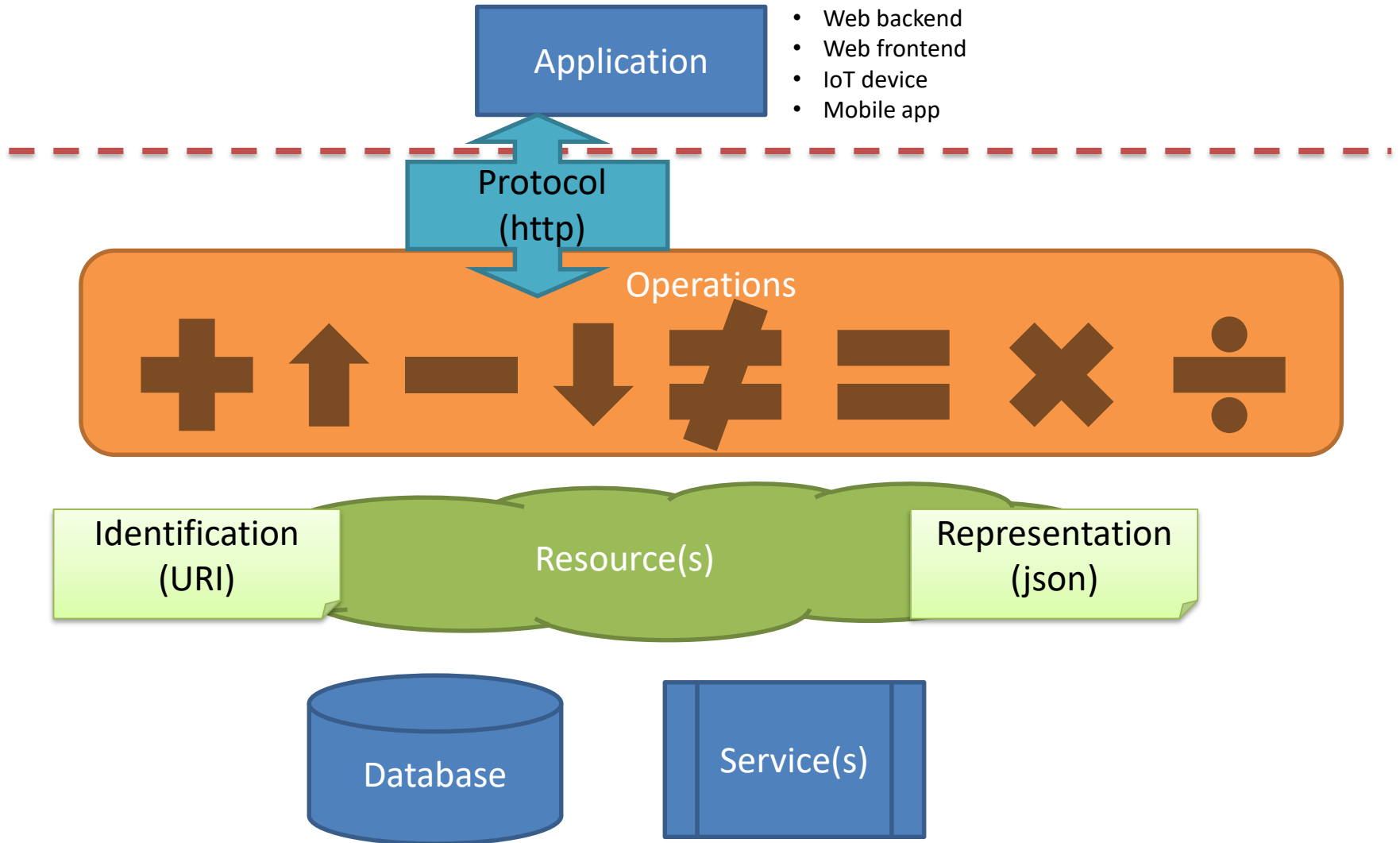
Database

Service(s)

Actions use HTTP Methods

- GET
 - Retrieve the representation of the resource (in the HTTP response body)
 - Collection: the list of items
 - Element: the properties of the element
- POST
 - Create a new resource (data in the HTTP request body)
 - Use a URI for a Collection
- PUT
 - Update an existing element (data in the HTTP request body)
 - Mainly for elements' properties
- DELETE

REST Architecture



- Web backend
- Web frontend
- IoT device
- Mobile app

Actions on Resources: Example

Resource	GET	POST	PUT	DELETE
/dogs	List dogs	Create a new dog	Bulk update dogs (<u>avoid</u>)	Delete all dogs (<u>avoid</u>)
/dogs/1234	Show info about the dog with id 1234	ERROR	If exists, update the info about dog #1234	Delete the dog #1234

Relationships

- A given Element may have a (1:1 or 1:N) relationship with other Element(s)
- Represent with: `/resource/identifier/resource`
- `http://api.polito.it/students/s123456/courses`
(list of courses followed by student s123456)
- `http://api.polito.it/courses/01qzp/students`
(list of students enrolled in course 01qzp)

Representations

- Returned in GET, sent in PUT/POST
- Different formats are possible
- Mainly: XML, JSON
 - But also: SVG, JPEG, TXT, ...
 - In POST: URL-encoding
- Format may be specified in
 - Request headers
 - **Accept: application/json**
 - URI extension
 - `http://api.polito.it/students/s123456.json`
 - Request parameter
 - `http://api.polito.it/students/s123456?format=json`

Real Life: GitHub API

GitHub Developer

Docs ▾ Blog Forum Versions ▾

🔍 Search...

REST API v3

Reference Guides Libraries

Overview

This describes the resources that make up the official GitHub REST API v3. If you have any problems or requests, please contact [GitHub Support](#).

- i. [Current version](#)
- ii. [Schema](#)
- iii. [Authentication](#)
- iv. [Parameters](#)
- v. [Root endpoint](#)
- vi. [GraphQL global node IDs](#)
- vii. [Client errors](#)
- viii. [HTTP redirects](#)
- ix. [HTTP verbs](#)
- x. [Hypermedia](#)
- xi. [Pagination](#)
- xii. [Rate limiting](#)
- xiii. [User agent required](#)

▼ Overview

[Media Types](#)

[OAuth Authorizations API](#)

[Other Authentication Methods](#)

[Troubleshooting](#)

[API Previews](#)

[Versions](#)

▶ [Activity](#)

▶ [Checks](#)

▶ [Gists](#)

▶ [Git Data](#)

▶ [GitHub Apps](#)

[GitHub Marketplace](#)

<https://developer.github.com/v3/>

Real Life: Twitter API



Search all documentation...

Docs

Basics

Accounts and users

Tweets

Direct Messages

Media

Trends

Geo

Ads

Metrics

Publisher tools & SDKs

Stay Informed

Staying informed about changes to our APIs is important for those developing on the platform and can be critical to maintaining your applications. We have a number of channels to help you stay in-the-loop.

[Learn how >](#)

Search Tweets

Use the Search API to find historical Tweets. Free to enterprise versions available.

[Learn more](#)

Filter realtime Tweets

Get only the Tweets you need by using advanced filtering tools with the realtime streaming API.

[Learn more](#)

<https://developer.twitter.com/en/docs>

Real Life: Google Calendar API

Resource Summary

- › Acl
- › CalendarList
- › Calendars
- › Channels
- › Colors
- › Events
- › Freebusy
- › Settings

API Reference



This API reference is organized by resource type. Each resource type has one or more data representations and one or more methods.

Resource types

Acl

For Acl Resource details, see the [resource representation](#) page.

Method	HTTP request	Description
URIs relative to https://www.googleapis.com/calendar/v3 , unless otherwise noted		
delete	DELETE <code>/calendars/<i>calendarId</i>/acl/<i>ruleId</i></code>	Deletes an access control rule.
get	GET <code>/calendars/<i>calendarId</i>/acl/<i>ruleId</i></code>	Returns an access control rule.
insert	POST <code>/calendars/<i>calendarId</i>/acl</code>	Creates an access control rule.

Contents

- Resource types
- Acl
- CalendarList
- Calendars
- Channels
- Colors
- Events
- Freebusy
- Settings

<https://developers.google.com/calendar/v3/reference/>

Real life: Facebook Graph API

facebook for developers

Documenti

Strumenti

Assistenza

Cerca su developers.facebook.com

Inizia

API Graph

Panoramica

Uso dell'API Graph

FAQ

Reference

Webhook

Suggerimenti avanzati

Registro modifiche

Server-Sent Events

API Graph

Versione più recente: v3.2

L'API Graph rappresenta il metodo principale tramite cui le app possono leggere e scrivere nel social graph di Facebook. Tutti i nostri SDK e i nostri prodotti interagiscono in qualche modo con l'API Graph, mentre le altre API sono sue estensioni, pertanto è fondamentale capire come funziona l'API Graph stessa.

Se non hai familiarità con l'API Graph, ti consigliamo di iniziare consultando questa documentazione:

Panoramica

Scopri come è strutturata l'API Graph, cosa sono i token d'accesso e come funzionano le versioni.

Uso dell'API Graph

Scopri come eseguire le operazioni più comuni.

Tool di esplorazione per la API Graph

Scopri come elaborare query e ricevere risposte dall'API Graph tramite la nostra app denominata Tool di esplorazione per la API Graph.

Riferimento

Scopri come consultare la documentazione di riferimento, in modo da trovare facilmente ciò che stai cercando.

Dopo aver acquisito familiarità con questi concetti di base, puoi passare ad argomenti più avanzati come quelli indicati qui sotto:

- Consulta la documentazione sui nostri SDK per iOS, Android, JavaScript, PHP o di terzi per scoprire di più su

Condividi cosa pensi

<https://developers.facebook.com/docs/graph-api>

Complex resource search

- Use `?parameter=value` for more advanced resource filtering (or search)
 - E.g.,
`https://api.twitter.com/1.1/statuses/user_timeline.json?screen_name=twitterapi&count=2`

Errors

- When errors or exceptions are encountered, use meaningful HTTP Status Codes
 - The Response Body may contain additional information (e.g., informational error messages)

```
{  
  "developerMessage" : "Verbose, plain language description of  
the problem for the app developer with hints about how to fix  
it.",  
  "userMessage": "Pass this message on to the app user if  
needed.",  
  "errorCode" : 12345,  
  "more info": "http://dev.teachdogrest.com/errors/12345"  
}
```

Authentication

Twitter Streaming API

```
Authorization: OAuth  
oauth_consumer_key="xvz1evFS4wEEPTGEFPHBog", ...
```

Amazon Web Services API

```
Authorization: AWS  
AKIAIOSFODNN7EXAMPLE:frJIUNo//y11qDzg=
```

Google API

```
Authorization: Bearer 1/fFBGRNJru1FQd44AzqT3Zg
```





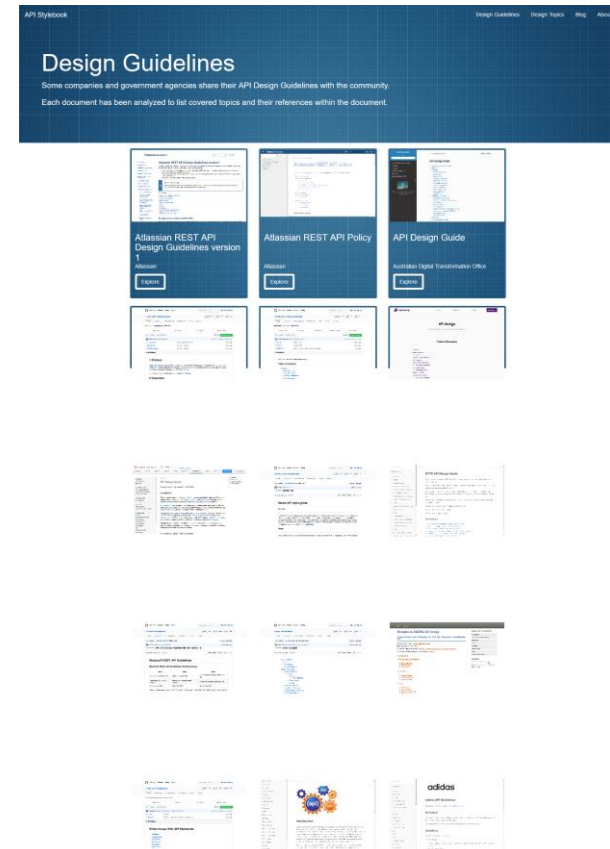
ADI Design Guidelines

API DESIGN GUIDE

<https://cloud.google.com/apis/design/>

API Design

- How to design a set of APIs for your application?
- Practical guidelines, with applied standard practices
- We will mainly follow the Google API Design Guide
 - <https://cloud.google.com/apis/design/>



<http://apistylebook.com/design/guidelines/>

API Design Flow

- Determine what types of **resources** an API provides.
- Determine the **relationships** between resources.
- Decide the resource **name schemes** based on types and relationships.
- Decide the **resource schemas**.
- Attach minimum set of **methods** to resources.

General approach

- A resource-oriented API emphasizes resources (data model) over the methods performed on the resources (functionality)
- A typical REST API exposes a large number of resources with a small number of methods
- Methods can be **standard** methods or **custom** methods.
- Standard methods are: List, Get, Create, Update, and Delete

Example (Gmail API)

- API service: `gmail.googleapis.com`
- A collection of users: `users/*`. Each user has the following resources.
 - A collection of messages: `users/*/messages/*`.
 - A collection of threads: `users/*/threads/*`.
 - A collection of labels: `users/*/labels/*`.
 - A collection of change history: `users/*/history/*`.
 - A resource representing the user profile: `users/*/profile`.
 - A resource representing user settings: `users/*/settings`.

Resource Names

- *resources* are named entities
- *resource names* are their identifiers
- Each resource **must** have its own unique resource name
- The resource name is made up of the ID of the resource itself, the IDs of any parent resources, and its API service name, separated by ‘/’

API Service Name	Collection ID	Resource ID	Resource ID	Resource ID
//mail.googleapis.com	/users	/name@example.com	/settings	/customFrom

Resource names

- Names used in APIs should be in correct American English.
- Commonly accepted short forms or abbreviations of long words may be used for brevity. E.g., API.
- Use intuitive, familiar terminology where possible. E.g., delete is preferred over erase.
- Use the same name or term for the same concept, including for concepts shared across APIs.
- Avoid name overloading. Use different names for different concepts.
- Avoid overly general names that are ambiguous within the context of the API.
- Carefully consider use of names that may conflict with keywords in common programming languages.

Collection names

- Must be **valid** C/C++ identifiers.
- Must be in **plural** form with lowerCamel case. If the term doesn't have suitable plural form, such as "evidence" and "weather", the singular form should be used.
- Must use **clear** and **concise** English terms.
- Overly **general** terms should be **avoided** or qualified. For example, rowValues is preferred to values.
- The following terms should be **avoided** without qualification: elements, entries, instances, items, objects, resources, types, values

Standard Methods

Standard Method	HTTP Mapping	HTTP Request Body	HTTP Response Body
List	GET <collection URL>	N/A	Resource* list
Get	GET <resource URL>	N/A	Resource*
Create	POST <collection URL>	Resource	Resource*
Update	PUT or PATCH <resource URL>	Resource	Resource*
Delete	DELETE <resource URL>	N/A	google.protobuf.Empty**

Let's read:

https://cloud.google.com/apis/design/standard_methods

Guidelines

- Design with standards in mind – for example RSS & ATOM
- *Create* should return URIs not resources
- Use the right HTTP methods for the right actions
- You are on HTTP – use the infrastructure
 - Proxy, Caching, Etag, Expires

URL Design

Guidelines (1/2)

Plural nouns for collections	/dogs
ID for entity	/dogs/1234
Associations	/owners/5678/dogs
HTTP Methods	POST GET PUT DELETE
Bias toward concrete names	/dogs (not animals)
Multiple formats in URL	/dogs.json /dogs.xml
Paginate with limit and offset	?limit=10&offset=0
Query params	?color=red&state=running
Partial selection	?fields=name,state
Use medial capitalization	"createdAt": 1320296464 myObject.createdAt;
Use verbs for non-resource requests	/convert?from=EUR&to=CNY&amount=100
Search	/search?q=happy%2Blabrador
DNS	api.foo.com developers.foo.com

Versioning

Include version in URL

/v1/dogs

Keep one previous version long enough for developers to migrate

/v1/dogs
/v2/dogs

Guidelines (2/2)

Errors

Status Codes

200 201 304 400 401 403 404 500

Verbose messages

{"msg": "verbose, plain language hints"}

Client Considerations

Client does not support HTTP status codes

?suppress_response_codes=true

Client does not support HTTP methods

GET /dogs?method=post
GET /dogs
GET /dogs?method=put
GET /dogs?method=delete

Complement API with SDK and code libraries

1. JavaScript
2. ...
3. ...



Using Python and Flask for implementing REST APIs

REST API IMPLEMENTATION

General rules

- Use `@app.route` to match the REST method
 - Specify methods: GET or POST
 - Use parametric routes for resource IDs
- Use `jsonify` to create a JSON response (with correct Content-type) from a Python object
- In case of POST, `request.json` parses a JSON-encoded request body

List method

```
@app.route('/users')  
def api_users():  
    return jsonify(users)
```

Get method

```
@app.route('/users/<name>')
def api_user(name):
    user = [u for u in users
            if u['name'] == name]
    if len(user) == 1:
        return jsonify(user)
    else:
        response = jsonify(
            { 'message': "No user "+name })
        response.status_code = 404
        return response
```

Create method

```
@app.route('/users', methods=['POST'])
def api_create_user():
    if request.headers['Content-Type']
        == 'application/json':
        new_user = request.json
        users.append(new_user)
    else:
        response = jsonify(
            { 'message': "Invalid Request" })
        response.status_code = 404
    return response
```

Calling / testing REST APIs

- Use the wonderful 'requests' package
 - `import requests`
 - <http://docs.python-requests.org/en/master/>
 - See: <http://docs.python-requests.org/en/master/user/quickstart/>
- Cheatsheet
 - `r = requests.get(url)`
 - `r = requests.post(url, json=data)`
 - `r.json()`



Calling List method

```
def list_users():  
  
    url = base_url+'/users'  
  
    r = requests.get(url)  
  
return r.json()
```

Calling Get method

```
def one_user(name):  
    url = base_url + '/users/' + name  
  
    r = requests.get(url)  
  
    if r.status_code == 200:  
        return r.json()  
    else:  
        return None
```




Calling Create method

```
def add_user(name, firstname, lastname):  
  
    user = { 'name': name,  
            'firstname': firstname,  
            'lastname': lastname }  
  
    url = base_url + '/users'  
  
    r = requests.post(url, json=user)
```

Resources

- http://en.wikipedia.org/wiki/Representational_state_transfer
- R. Fielding, Architectural Styles and the Design of Network-based Software Architectures, <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- Learn REST: A Tutorial: <http://rest.elkstein.org/>
- <https://pages.apigee.com/ebook-web-api-design-registration.html>
- <http://www.slideshare.net/apigee/api-design-3rd-edition>
- Google API Design Guide: <https://cloud.google.com/apis/design/>
- Discussion forum: groups.google.com/group/api-craft

License

- This work is licensed under the Creative Commons “Attribution-NonCommercial-ShareAlike Unported (CC BY-NC-SA 4.0)” License.
- You are free:
 - to **Share** - to copy, distribute and transmit the work
 - to **Remix** - to adapt the work
- Under the following conditions:
 -  – **Attribution** - You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
 -  – **Noncommercial** - You may not use this work for commercial purposes.
 -  – **Share Alike** - If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.
- To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-sa/4.0/>