Designing a Web Service with REST-API by using Python and Flask

REST (Representational State Transfer) has developed as the standard architecture design for web services and web APIs.

REST:

The characteristics of a REST system are defined by six design rules:

**Client-Server:** There should be a separation between the server that offers a service, and the client that consumes it.

**Stateless:** Each request from a client must contain all the information required by the server to carry out the request. In other words, the server cannot store information provided by the client in one request and it in another request.

**Cacheable:** The serve must indicate to the client if requests can be cached or not.

**Layered system:** Communication between a client and server should be standardized is such a way that allows intermediaries to respond to requests instead of the end server, without the client having to do anything different.

**Uniform Interface:** The method of communication between a client and a server must be uniform.

**Code on demand:** Servers can provide executable code or scripts for clients to execute in their context. This constraint is theonly one that is optional.

REST web service:

The REST architecture was originally designed to fit the HTTP Protocol that the world wide web uses.

Central to the concept of RESTful web services is the notion of resources. Resources are represented by URIs. The clients send requests to these URIs using the methods defined by the HTTP protocol, and possibly as a result of that the state of the affected resource changes.

The HTTP request methods are typically designed to affect a given resource in standard ways.

**Method:**

GET: Obtain information about a resource

POST: Create a new resource

PUT: Update a resource

DELETE: Delete a resource

The REST design does not require a specific format for the data provided with the requests. In general data is provided in the request body as a Json blob, or sometimes as arguments in the query string portion of the URL.

## Designing a simple web service

The task of designing a web service or API that adheres to the REST guidelines then becomes an exercise in identifying the resources that will be exposed and how they will be affected by the different request methods.

Let's say we want to write a To Do List application and we want to design a web service for it.

Here I have decided to include the name of the application and the version of the API in the URL. Including the application name in the URL is useful to provide a namespace that separates this service from others that can be running on the same system. Including the version in the URL can help with making updates in the future, since new and potentially incompatible functions can be added under a new version, without affecting applications that rely on the older functions.

The next step is to select the resources that will be exposed by this service. This is an extremely simple application, we only have tasks, so our only resource will be the tasks in our to-do list.

We can define a task as having the following fields:

* **id**: unique identifier for tasks. Numeric type.
* **title**: short task description. String type.
* **description**: long task description. Text type.
* **done**: task completion state. Boolean type.

And with this we are basically done with the design part of our web service. All that is left is to implement it!