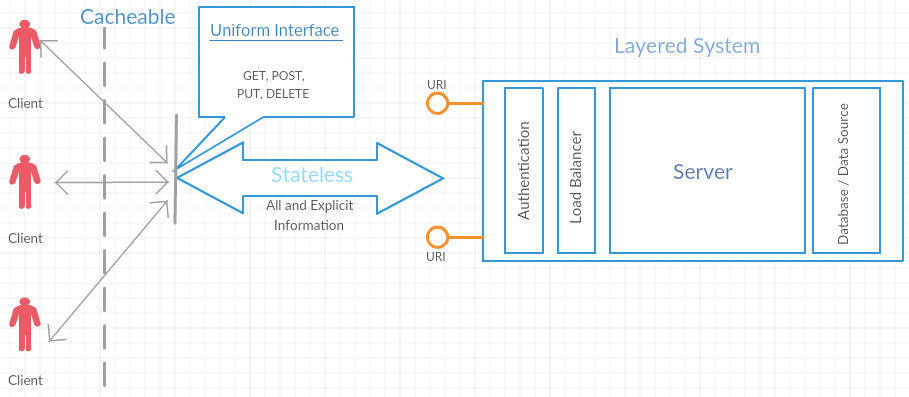
**A Look at REST API Design Patterns**

In this article, we talk a bit about useful and intuitive design patterns in RestFul Webservice API architecture. In general, design patterns are formalized best practices that a programmer can use to solve common problems when designing an application or system. Below are different elements of design patterns for a REST architecture.

**REST architecture style constraints:**There are design rules that are applied to establish the different characteristics of the REST architectural style, which are referred to as REST constraints:



1. *Client-server*
2. *Statelessness*
3. *Cacheable*
4. *Uniform interface*
5. *Layered systems*

**Goals of RESTful API design:**Restful APIs should be straightforward, unambiguous, easy to consume, well-structured, and most importantly, accessible with well-known and standardized HTTP methods.

API design goals:

**1. Affordance**

Affordance is the possibility of an action on an object or environment. An API with clear perceived affordance allows the developer to understand its purpose and to use it seamlessly inside the Cybernetic Environment it was designed for.

**2. Loosely Coupled**

In Restful APIs, multiple clients are connected to the same backend server. So when the internal representation of a server is changed, it should not affect API consumption at the client-side. In a loosely coupled design, APIs are independent, and modifications in one won't impact the operation of consumers. Within an API, the components get added, modified, or replaced. However, the loose coupling approach offers clients better flexibility and reusability of APIs while its elements are added, replaced, or changed. Well-designed APIs exhibit loose coupling and well-composed functionalities across service boundaries to maximize scalability factors.

**3. Leverage Existing Web Architecture**

RESTful APIs should use HTTP as a transport layer since the infrastructure, server, and client libraries for HTTP are widely available already. RESTful APIs should take advantage of HTTP methods, or verbs, such as GET, PUT, and POST.

**RESTful API Design Patterns:**API design patterns provide a description or templates to solve specific, recurring API design problems that any software architects and API designers would like to adopt in their API designs.

**1. Statelessness**

Communication between client and server should be stateless, which means that every client request contains all the information necessary for the server to process the request. So there is no global state thereby reducing the complexity of the server.

The good news is that some Restful web frameworks provide an out-of-the-box implementation for Statelessness. For example **Spring Boot's REST API framework.**

**2. Content Negotiation**

Content-negotiation is a mechanism or process that services and clients can select as their resource representation format for their communication and handshakes during their usual course of communication.

HTTP specification comes up with a set of standard headers, through which the client can get information about a requested resource and carry the messages that indicate its representations.

So, for content negotiation, REST services need to use HTTP headers; that is, when the client makes requests, it includes the accepts header, the list of file types that the client and server can handle with no additional steps to the client requests, the server processes, and replies.

**E.g. In Java we can use produces property in @GetMapping annotation**

**@GetMapping(path="/investors/{investorId}/stocks/{symbol}", produces={MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})**

**3. URI Templates**

It is always the case that the client may need to include some additional information in their request, and how the server lets the client include that information about resources in the URIs. Server-side developers require the ability to describe the layout of the URIs that their services will respond to. So we can use the URI template. URI templates provide a way to describe a set of resources as variables.

**E.g. If there are multiple resources,**

**People:**[**https://swapi.co/api/people/**](https://swapi.co/api/people/)

**Planets:**[**https://swapi.co/api/planets/**](https://swapi.co/api/planets/)

**Films:**[**https://swapi.co/api/films/**](https://swapi.co/api/films/)

**Species:**[**https://swapi.co/api/species/**](https://swapi.co/api/species/)

**We can use simply**[**https://swapi.co/api/{resource\_id}/**](https://swapi.co/api/%7bresource_id%7d/)**as a URI template**

**The @PathVariable annotation provided by Spring Boot helps us implement the URI template pattern in our code seamlessly.**

**4. Design for Intent**

Design for intent is a method that expresses the different relationships between objects so that changes to one object automatically propagates changes to others.

In a RESTful API world, the API should be developed to ensure they meet the requirements of the desired use cases provided and faced by users, but without exposing the internal business objects. Design for intent is a strategic design pattern that's intended to influence or result in specific and additional user behaviors.

**5. Pagination**

When there are multiple rows of data available then APIs should give the requested data in batch-wise (Pagination).

Pagination is a concept that helps in serving only part of the data as a response, however, with information about how to access all the data from the server, page by page, without much load and high computation for the server to serve the whole data.

**E.g. xxx.api.com/students?page=2.**

**There are three variants of resource representation ways of pagination:**

1. **Offset-based**
2. **Time-based**
3. **Cursor-based**

**5. Discoverability**

Discoverability is a very important factor in API designing, helping developers figure out programmatically whether the site that's being accessed has an API enabled or not will be the most critical responsibility of the API.

Using the following two ways we can ensure discoverability of API for developers,

**1. By Valid HTTP Methods**

When clients call REST services with invalid HTTP methods, the response of that request should end up in the 405 HTTP error code; that is, 405 Method Not Allowed. In addition to the error code, the response header should provide flexibility to the client to find the supported methods that allow headers in its response.

**2. Providing the URI of the Newly Created Resource**

Including the URI as part of the location header as the response to the newly created resource is another method of discoverability. The returned URI as a location will be available through GET.

In Sprint boot it gives out-of-the-box solutions for discoverability E.g. Using @GetMapping annotation.

**6. Unicode**

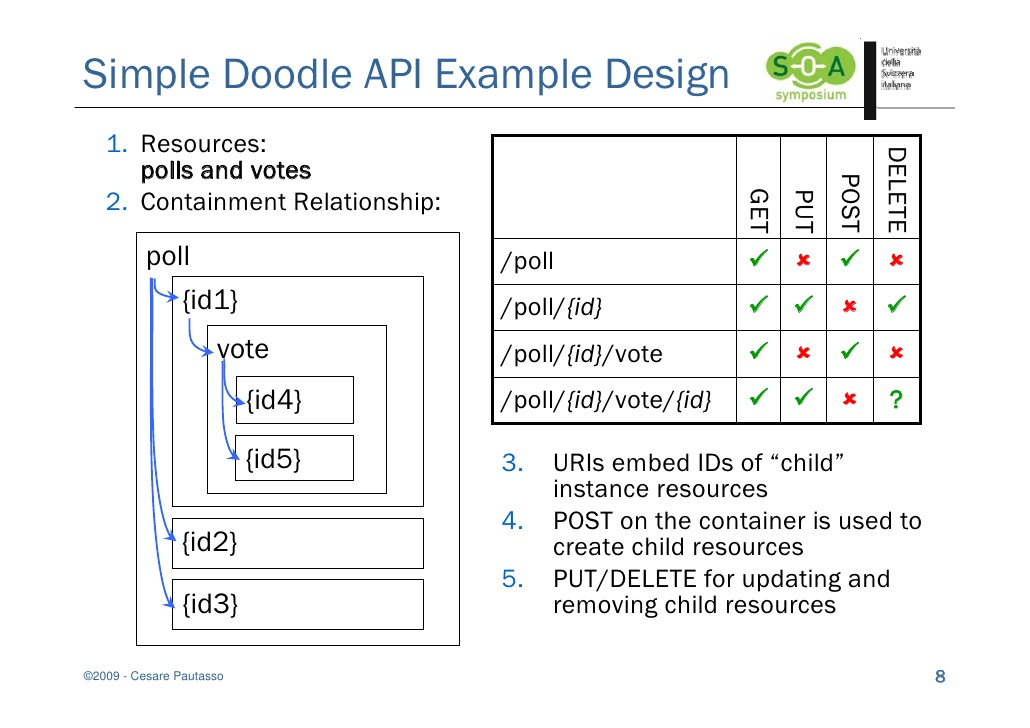
A simple yet powerful way to make API support multiple languages is to enable the API to support Unicode.

Unicode is an encoding standard that supports an international character set. It has a unique number for every character across multiple languages including Chinese, Korean, and Arabic, and their scripts. The unique number makes almost all characters identifiable and accessible across platforms, programs, and devices.

**API Design Principles**

In general, the following standard guidelines should be followed while designing high-quality Restful APIs.

1. Ubiquitous web standards
2. Flexibility
3. Standardization
4. Optimization
5. Granularity
6. Sandbox or playground



This should be an excellent starting point for anyone who wants to get their hands into RESTful services, with not just the basics, but essential patterns as well. The enemy of design patterns are anti-patterns, which seem sounds but are counter-productive when executed. Unfortunately, anti-patterns are hard to detect. [Read more](https://docs.embold.io/anti-patterns/#what-are-anti-patterns) about how a free static analyzer like [Embold](https://embold.io/" \t "_blank) detects up to 30 structural design issues in Java programming. You can check out my other article on anti-patterns as well.