به نام خدا

پژوهشکده فناوری اطلاعات

گروه توسعه نرم افزار

ملاحظات مطرح در توسعه API

پروژه: مدیریت طرح جویشگر

کد پروژه: ۹۰۱۹۳۰۱۰۰

تهیه‌کنندگان: سیدرضا متقی، بامداد وفایی، زهرا گل‌میرزایی، احسان کشتکاری

تاریخ ارائه: ۹۵/۳/۱

نسخه/وضعیت: ۲,۰

خواننده گرامی، در راستای تحقق ماموریت پژوهشگاه ارتباطات و فناوری در فراهم سازی سکویی برای ارتقاء دانش، انتقال فناوری و بومی‌سازی محصولات و خدمات حوزه فاوا و با هدف جلب مشارکت علاقه‌مندان در توسعه و بهره مندی از دستاوردهای پژوهشگاه ارتباطات و فناوری اطلاعات، آزاد رسانی این دستاوردها در زمره برنامه های اولویت دار پژوهشگاه به شمار می‌آید. به همین منظور مستند حاضر تحت مجوز بین المللی **CC-BY-SA-NC** نسخه ۴، در دسترس عموم قرار گرفته است.

شایان ذکراست تحت این مجوز، ضمن حفظ کلیه حقوق مالکیت فکری این مستند برای پژوهشگاه ارتباطات و فناوری اطلاعات، بازانتشار و بکارگیری آن صرفاً برای موارد تحقیقاتی و با ذکر نام **پژوهشگاه ارتباطات و فناوری اطلاعات (مرکز تحقیقات مخابرات ایران)** بلامانع است.

شناسنامه گزارش

عنوان: ملاحظات مطرح در توسعه API		**شماره نسخه: ۲,۰**
کد: P-PD-ALL-SBM-S-019.3.00	**نوع گزارش:** فنی	**تاریخ ارائه گزارش:** ۹۵/۴/۲۷
نام پروژه: مدیریت طرح جویشگر		**نوع پروژه: پژوهشی - کاربردی**
تاریخ شروع: ۹۳/۱۱/۲۰		**تاریخ پایان: ۹۶/۵/۲۰**
نام گروه: توسعه نرم‌افزار		
کد پروژه: ۹۰۱۹۳۰۱۰۰		**شماره و تاریخ قرارداد: ۹۳/۱۱/۲۰**
مجری: علیرضا یاری		**ناظر / ناظرین:** -
تهیه کننده/ تهیه کنندگان: بامداد وفایی، سیدرضا متقی، زهرا گل‌میرزایی، احسان کشتکاری		
نام و نشانی مجری: تهران، انتهای خیابان کارگر شمالی، پژوهشگاه ارتباطات و فناوری اطلاعات (مرکز تحقیقات مخابرات ایران)_ کد پستی: ۱۴۳۹۹۵۵۴۷۱ _ تلفن: ۱۰-۸۰۰۵۵۰۸		
نام و نشانی حمایت کننده: تهران، خیابان شریعتی، وزارت ارتباطات و فناوری اطلاعات		
ملاحظات:		
چکیده: واسط برنامه کاربردی یا API، امکان به کارگیری داده‌ها یا کارکردهای یک سیستم نرم‌افزاری را فراهم می‌نماید. API روشی پایدار، سازمان‌یافته و استاندارد برای دسترسی به منابع نرم‌افزاری، اتصال سیستم‌های مختلف نرم‌افزاری به یکدیگر، یکپارچه سازی و توسعه سیستم‌های نرم‌افزاری پیچیده و بزرگ مقیاس می‌باشد. این مستند با هدف استاندارد سازی APIهای ارائه شده در طرح جویشگر و بررسی ملاحظات مطرح در طراحی APIها تهیه و تدوین شده است. با توجه به گستردگی، کارایی و سادگی استفاده از REST APIها، تمرکز این مستند صرفاً بر روی این دسته از APIها قرار دارد. در این مستند نحوه طراحی فرمت درخواست، ملاحظات مطرح در طراحی فرمت پاسخ نظیر مدیریت کدهای خطا، فرمت داده، فرمت پاسخ و ملاحظات غیرکارکردی نظیر امنیت و کارایی APIها بررسی شده است.		
کلمات کلیدی: API، REST، JSON، HTTP ، XML، فرمت درخواست، فرمت پاسخ،		
وضعیت گزارش: نهایی	**زبان گزارش: فارسی**	
وضعیت دسترسی: عادی	**تعداد صفحات: ۲۲**	

چکیده

واسط برنامه کاربردی یا API، امکان به کارگیری داده‌ها یا کارکردهای یک سیستم نرم‌افزاری را فراهم می‌نماید. API روشی پایدار، سازمان‌یافته و استاندارد برای دسترسی به منابع نرم‌افزاری، اتصال سیستم‌های مختلف نرم‌افزاری به یکدیگر، یکپارچه‌سازی و توسعه سیستم‌های نرم‌افزاری پیچیده و بزرگ مقیاس می‌باشد. همچنین وجود API به منظور طراحی سیستم‌های با کیفیت بالا و امکان به اشتراک گذاری داده و کارکردها ضروری است.

یکی از محل‌های پرکاربرد APIها، در موتورهای جستجو است. در حال حاضر اکثر موتورهای جستجو نظیر Yandex, Baidu, Google, Bing, Yahoo, Naver دارای APIهایی برای استفاده کاربران، شرکا و داخل سازمانی می‌باشند. از APIهای موجود می‌توان APIهای ماشین ترجمه(به زبان‌های مختلف)، ارائه هم معنی، نقشه، جستجو (صوت، متن، تصویر) و مرورگر را نام برد. این APIها علاوه بر اینکه کاربرد درون سازمانی داشته و سبب تهسیل توسعه و گسترش برنامه‌های کاربردی مختلف می‌گردد، زمان رسیدن به بازار را نیز به میزان قابل توجهی کاهش می‌دهد. علاوه براین API می‌تواند به عنوان یک کانال توزیع جدید در نظر گرفته شود که سرویس یا داده سازمان را به شکلی استاندارد و قابل کنترل و مدیریت در اختیار مشتریان آن قرار می‌دهد.

این مستند با هدف استانداردسازی APIهای ارائه شده در طرح جویشگر و بررسی ملاحظات مطرح در طراحی APIها تهیه و تدوین شده است. با توجه به گستردگی، کارایی و سادگی استفاده از REST APIها، تمرکز این مستند صرفاً بر روی این دسته از APIها قرار دارد. در این مستند نحوه طراحی فرمت درخواست، ملاحظات مطرح در طراحی فرمت پاسخ نظیر مدیریت کدهای خطا، فرمت داده، فرمت پاسخ و ملاحظات غیرکارکردی نظیر امنیت و کارایی APIها بررسی شده است.

اطلاعات مرتبط

مستندات مرتبط

شماره مستند	نوع مستند	نام مستند
	فنی	مروری بر مفاهیم پایه در توسعه API

تغییرات اعمال‌شده در نسخه‌های پیشین

شماره نسخه	تاریخ	تغییرات اعمال شده
۱,۰	۹۵/۳/۱	
۲,۰	۹۵/۴/۲۷	اصلاح فرمت، مراجع و شناسنامه گزارش

تأییدکنندگان

	نام و نام خانوادگی	تاریخ	امضاء	ملاحظات
مجری پروژه	علیرضا یاری			
تهیه کننده/ تهیه کنندگان	سیدرضامتقی، بامداد وفایی، زهرا گل‌میرزایی، احسان کشتکاری			
ناظر پروژه	-			
مدیر گروه	احسان کشتکاری			
مسئول مستندات پژوهشکده	مانا روزی‌طلب			
رئیس پژوهشکده/ معاون پژوهشی	علیرضا یاری			

فهرست

۱. مقدمه

هدف این مستند بررسی و شرح نکات کلیدی در طراحی یک REST API می‌باشد. از آنجایی که میزان استفاده و اقبال عمومی از REST APIها در چند سال اخیر به میزان قابل توجهی افزایش یافته و کارایی و سادگی بیشتری نسبت به سایر انواع APIی وب دارند تمرکز اصلی این مستند بر روی این دسته از APIها می‌باشد. به منظور مطالعه بیشتر در رابطه با تفاوت‌ها و شباهت‌های مابین انواع APIهای وب در پیوست این مستند مقایسه اجمالی ما بین REST, SOAP و RPC صورت گرفته است و توضیحات کامل‌تری در مستند «مروری بر مفاهیم پایه در توسعه API » ارائه شده است.

از آنجایی که REST یک نوع معماری و ساختار طراحی API می‌باشد و هنوز به طور دقیق استاندارد نشده است، انعطاف پذیری زیادی در پیاده‌سازی آن وجود دارد. و به دلیل آزادی و انعطاف پذیری در پیاده‌سازی ساختار، نیاز به وجود چهارچوب‌ها و نکات کلیدی در طراحی APIها وجود خواهد داشت. در تعاریف هر نوع سرویس وب که پنج ویژگی واسط یکسان[1]، Stateless، قابلیت کش نمودن[2]، معماری کلاینت-سرور و معماری لایه‌ای[3] را داشته باشد، دارای معماری REST است. در متون عموماً به سرویسی که تمامی این ویژگی‌ها را به طور کامل پیاده کرده باشد Pure REST می‌گویند. این در حالی است که اکثر وب سرویس‌ها یک یا چند ویژگی را ندارند و صرفاً تعداد معدودی از آنها در این دسته می‌گنجد. بنابراین در مقابل این عبارت، عبارت Pragmatic REST مطرح گردید که چهارچوب کلی قوانین معماری REST را در برمی گیرد اما لزومی به رعایت تمامی قواعد وجود نخواهد داشت. مهمترین ویژگی که این دسته از APIها ندارند ویژگی HATEOAS (زیرمجموعه‌ای از واسط یکسان) مبنی بر عدم hard code نمودن API است (آدرس تمامی APIها باید از طریق تعامل به دست آید). در این حالت بکارگیری مجموعه‌ای از قوانین به عنوان نکات کلیدی ضروری است تا یکپارچه‌سازی و ارتباط سیستم‌ها و سامانه‌های نرم‌افزاری به شکل صحیحی صورت پذیرد. در این مستند به بررسی اجمالی این نکات خواهیم پرداخت.

۲. فرمت درخواست

در صورتی که الگوی درخواست به خوبی طراحی شده باشد، استفاده، توسعه و به‌روز رسانی API بسیار ساده خواهد شد. دو نکته اساسی که در طراحی درخواست باید به آن توجه شود عبارتند از:

- استفاده از نام به جای فعل
- استفاده صحیح از متدهای HTTP برای REST API

[1] Uniform Interface
[2] Cacheable
[3] Layered System

۲-۱. استفاده صحیح از متدهای HTTP

در حالت کلی سبک معماری REST به صورتی است که تمام اعمال آن تحت چهار عمل ایجاد (Create)، خواندن(Read)، بروزرسانی(Update) وحذف (Delete) انجام گرفته و تحت آنها خلاصه می‌شود. به چهار عمل اصلی اشاره شده به اختصار CRUD گویند. پیاده سازی REST به این صورت است که هر یک از اعمال آن به یک یا دو متد HTTP نگاشت می شود. برای درک بهتر به جدول زیر نگاه کنید:

جدول ۱ شرحی از متدهای HTTP

عمل درخواستی	نوع درخواست	حالت کلی	پاسخ مناسب
POST	ایجاد	ایجاد یک منبع جدید	201
GET	خواندن	خواندن یا مرتب کردن منابع	200
PUT	بروزرسانی/جایگزینی	به روز رسانی منابع یا عمل حذف یک منبع و ایجاد یک منبع جدید	200
PATCH	بروزرسانی/اصلاح	به روز رسانی و اصلاح(به جز فیلد اصلی)	200
DELETE	حذف	حذف یک منبع	200

همانطور که در بالا هم گفته شد REST یک سبک معماری است. اکثر APIهای وب آدرس URI خود را مبتنی بر متد طراحی می‌کنند و این متدها در ابتدا یا انتهای نام خود حاوی یک فعل هستند. برای مثال، در جدول زیر نمونه‌ای از APIهای یک سبد خرید لیست شده است. در این فراخوانی‌ها سعی شده اعمال مرتبط با خرید یک آیتم از فروشگاه بازنمایی شود. در این مثال اعمالی مانند لیست کردن یک یا چند آیتم، اضافه نمودن، حذف آن از سبد خرید و حذف سبد خرید آورده شده است.

جدول ۲ نمونه‌ای از بکارگیری متدهای HTTP به شکل نادرست

فعالیت	متد[1]	URI
اضافه نمودن یک آیتم جدید به سبد خرید	POST	http://api.shopping.com/InsertNewItem
حذف یک آیتم از سبد خرید	POST	http://api.shopping.com/DeleteItem
لیست نمودن کل موجودی سبد خرید	GET	http://api.shopping.com/ListCart?cartid=X
نمایش یک آیتم از سبد خرید	GET	http://api.shopping.com/ShowItem??cartid=X&itemid=Y
حذف سبد خرید	POST	http://api.shopping.com/DeleteCart

شاید در نگاه اول به نظر برسد استفاده از API فوق چندان سخت نیست اما نیاز است که تک تک متدها و عملیات‌ها را به خاطر سپرد و در صورتی که تعداد متدهای تعریف شده برای API زیاد باشد این امر دشوار خواهد شد. برای مثال تصور کنید به جز متدهای

[1] Operation

مرتبط با سبد خرید، ۵۰ شئ دیگر نیز موجود بودند که هر یک متدها و توابع خود را داشتند در این صورت نیاز بود برای هر فراخوانی API به مستند API رجوع کرده و مرتب این مستند را جستجو نمایید تا بتوانید متد مورد نیاز خود را بیابید.

در معماری REST پشت هر URI یک منبع یا شیء قرار دارد و نه یک عملیات. بنابراین به جای تعریف یک متد در آدرس اصلی وب سرویس بهتر است با بکارگیری صحیح متدهای HTTP برای تعریف عملیات‌ها و بکارگیری نام متناظر با منبع در URI آدرس وب سرویس را پیاده‌سازی نمود.

جدول ۳ نمونه‌ای از بکارگیری متدهای HTTP به شکل درست

فعالیت	متد	URI
اضافه نمودن یک آیتم جدید به سبد خرید	POST	http://api.shopping.com/Cart/cartName
حذف یک آیتم از سبد خرید	DELETE	http://api.shopping.com/Cart/cartName/item/itemName
لیست نمودن هر چه در سبد خرید موجود است	GET	http://api.shopping.com/Cart/cartName
نمایش یک آیتم از سبد خرید	GET	http://api.shopping.com/Cart/cartName/item/itemName
جایگزینی یک آیتم در سبد خرید	PUT	http://api.shopping.com/Cart/cartName/item/itemName
حذف سبد خرید	DELETE	http://api.shopping.com/Cart/cartName

در جدول فوق فراخوانی‌ها بسیار ساده‌تر هستند و به خاطر سپردن این مجموعه از URIها و الگوی بکارگرفته شده بسیار ساده‌تر بوده و توسعه محصول و بکارگیری API را نیز سهل‌تر می نماید. برای مثال برای لیست نمودن تمامی سبدهای خرید در سیستم، کافی است دستور زیر به سیستم اضافه شود.

```
HTTP GET to http://api.shopping.com/carts
```

در حقیقت به جای به خاطر سپردن انبوهی از متدها و آدرس‌ها نام منبع و الگوی ساخت آدرس به خاطر سپرده خواهد شد. در این الگو، پارمترهای درخواست[1] همچنان جایگاه ویژه‌ای دارند. در حقیقت پارمترها به منظور تعیین فاکتورهای اضافی کاربرد دارند. برای مثال یک سبد خرید را در نظر بگیرید که آیتم‌های زیادی را شامل می‌شود و به علت تعدد بالا، نیاز است زمانی که می‌خواهید لیست این آیتم‌ها را مشاهده کنید آن را صفحه بندی نمایید. مثلا برای مشاهده آیتم‌های ۲۰ تا ۲۹ باید URI ذیل را فراخوانی نمود:

```
http://api.shopping.com/cart/cartName?start=20&count=10
```

یا فرض کنید در این فروشگاه علاوه بر سبد خرید اطلاعات مشتریان نیز ذخیره می‌شود، در این صورت برای حذف، اضافه کردن و یا لیست مشتریان باید از توابع متناظر HTTP (به ترتیب DELETE، POST و GET) استفاده شود و در صورتی که علاوه براین متدها نیاز باشد که مثلاً پیش از اضافه نمودن یک مشتری، آن را تایید نمود باید به این صورت عمل کرد:

[1] Query parameters

```
…/customers/123/operation/approve
```

و یا

```
…/customers/123?operation=approve
```

۲-۲. اسم مفرد یا جمع

سوال مطرح در این بخش این است که در مثال مشتریان فروشگاه بکارگیری کدام یک از دو مورد زیر صحیح‌تر است؟

```
…/customer
…/customers
```

هر یک از این دو گزینه موافقین و مخالفانی در میان توسعه دهندگان دارد. اما عموماً آنچه در بین توسعه دهندگان رایج است استفاده از نام جمع است زیرا مفهوم‌تر می‌باشد. برای مثال customers در حقیقت به مجموعه ای از مشتریان اشاره می‌کند و /customers/123… به مشتری ۱۲۳ .

نکته حائز اهمیت در این بخش یکدست بودن و پیروی از یک قاعده یکسان است. بدین معنی که برای نام‌گذاری تمامی منابع موجود یا از نام جمع استفاده شود و یا از اسم مفرد. بکارگیری هر دو سیاست به صورت همزمان سبب سردرگمی استفاده کنندگان API خواهد شد.

پس به طور خلاصه پیشنهاد می‌شود برای طراحی آدرس API به نکات زیر توجه نمایید:

- تا حد امکان URI پایه خود را ساده نگاه دارید
- از اسم به جای فعل استفاده کنید.
- اسامی را به صورت جمع بکار برید.
- متدهای پروتکل HTTP را در جای صحیح خود بکار برید

۳. نسخه بندی

نسخه بندی[1] یکی دیگر از نکات مهم در طراحی API می‌باشد. یکی از تصمیمات کلیدی در طراحی API وجود یا عدم وجود نسخه‌بندی است. این انتخاب تاثیر مستقیمی بر روی برنامه‌های کاربردی استفاده کننده از API خواهد داشت و به همین دلیل اهمیت بسزایی دارد. در حقیقت API یک نوع قرارداد مابین ارائه دهنده API و استفاده کننده از آن می‌باشد. بنابراین ضروری است که بین نیاز به ارتقا قابلیت‌ها و اضافه کردن ویژگی‌های جدید به API از یک سو و پشتیبانی از نسخ قدیمی‌تر به طور مستمر و پایداری API از سوی دیگر توازنی ایجاد نمود. به بیان دیگر از یک سو باید تلاش فراوانی صورت گیرد که API به عنوان یک قرارداد

[1] Versioning

بدون تغییر باقی بماند(ممکن است تغییراتی در API رخ دهد و تیم توسعه دهنده بهبودهایی را بر روی آن انجام دهند اما این تغییرات سبب مختل شدن کد برنامه‌هایی که از API استفاده می‌کنند نخواهد شد.) و از سوی دیگر در طول عمر یک API مطمئناً زمان‌هایی وجود دارد که نیاز است API به طور کامل تغییر داده شود و نسخه بعدی آن شروع به کار نماید. برای حل این چالش هر یک از شرکت‌ها و ارائه دهندگان API یکی از دو راهکار ارائه نسخه یا عدم ارائه آن را انتخاب کرده اند البته آنچه بیشتر رایج است ارائه API به همراه نسخه بندی است. برای برقراری توازن بین این دو نیاز و بررسی تجربیات ارائه دهندگان API پیشنهاد می‌شود:

- API خود را به همراه نسخه و ترجیحاً یک نسخه یک رقمی ارائه دهید (روش رایج استفاده از نسخه در URI است که در ادامه شرح داده خواهد شد).
- در هر زمان که نیاز به تغییر جدیدی در آدرس API احساس شد، تا حد امکان از اعمال این تغییر جلوگیری کنید و یا بررسی کنید که آیا راه‌کار دیگری به جز تغییر آدرس وجود دارد یا خیر. در صورت نیاز نسخه API خود را یک واحد افزایش دهید.
- تا حد امکان از نسخ قبلی API خود پشتیبانی کنید.

چالش بعدی درنسخه بندی API روش نسخه بندی است. برای مثال دو نمونه از روش‌های مدیریت نسخه که توسط دو شرکت Twilio، Salesforce ارائه شده است در جدول زیر نمایش داده شده است.

Twilio	*/2010-04-01/Accounts/*
Salesforce	*/services/data/v20.0/objects/Account*

در مثال اول از یک زمان که نشان دهنده زمان کامپایل شدن کد یا زمان ارائه API است استفاده شده است. این کد در تمام درخواستهای HTTP رد و بدل می‌شود. در این حالت، زمانی که یک درخواست به سرویس وب می‌رسد یک جستجو در سمت سرور انجام خواهد گرفت. از طریق زمان مشخص شده در آدرس می‌توان تشخیص داد که این نسخه از API مربوط به زمانی است که کد کامپایل شده (یا API ارائه شده است) و مسیر یابی با توجه به آن انجام خواهد گرفت.

در مثال دوم از شماره برای نسخه‌بندی API استفاده شده است که این روش، عموماً رایج‌ترین روش ارائه نسخه API است. در این حالت بیشتر توسعه دهندگان و ارائه دهندگان API پیشنهاد می‌کنند که ترجیحاً از یک عدد طبیعی استفاده شود و از ممیز (بعنوان مثال 2.1) که نشان دهنده زیر نسخه است استفاده نشود.

مورد دیگری که اینجا نیاز به اشاره دارد آن است که آیا نسخه API باید در آدرس قرار گیرد یا در سربار (Header)؟ این مساله نیز موافقان و مخالفان خود را دارد و شاید هنوز در این رابطه یک استاندارد و روش واحد وجود ندارد اما بررسی‌ها نشان می‌دهد که استفاده از آدرس در بین شرکت‌های بزرگ نظیر Google search, twitter, Atlassian, Azure و... و همین طور در میان توسعه دهندگان API رایج‌تر می‌باشد. در این روش نکته‌ای که باید در نظر گرفته شود این است که در حالتی که استفاده کننده به هر دلیلی فراموش کرد نسخه را در آدرس قرار دهد، بهتر است به طور پیش فرض آخرین نسخه فراخوانی شود. یعنی در دو حالت زیر پاسخ یکی باشد و پیغام خطایی رخ ندهد.

```
http://api.shopping.com/customers/123
http://api.shopping.com/v2/customers/123
```

۴. کدهای خطا

در طراحی API ارائه کدهای خطای مناسب یکی دیگر از اجزای مهم محسوب می‌شوند. از دیدگاه استفاده کننده از API، هر چیزی که در سمت API است بصورت جعبه سیاه است در نتیجه با ارائه کدهای خطای مناسب به کاربر می‌توان بخوبی چگونگی عملکرد API را نمایش داد. از طرفی دیگر، در زمانی که توسعه دهندگان در حال رفع مشکل برنامه خود هستند، ارائه کدهای خطای مناسب توسط API مورد استفاده از برنامه می‌تواند به عیب یابی و رفع سریعتر مشکل کمک کند. روشهای مختلفی برای نمایش کدهای خطا وجود دارد که بهترین روش برای ارائه آن مانند مثال زیر است:

```
HTTP Status Code: 401
{"status":"401", "message":"Authenticate","code": 20003, "more
info": "http://www.twilio.com/docs/errors/20003"}
```

در این مثال که مربوط به API شرکت Twilio می‌باشد، می‌توان مشاهده کرد که کد حالت ۴۰۱ بازگردانده شده و در ادامه و در بدنه پیام توضیحات بیشتر و همچنین مستندی که می‌تواند به کاربر در رفع مشکل کمک آورده شده است.

۴-۱. استفاده از کدهای حالت HTTP

کدهای حالت HTTP بخشی از استاندارد RFC 7231 هستند که در مجموع شامل ۷۰ کد می‌باشند. در طراحی کدهای خطای وب سرویس بهتر است از این کدها به شکل مناسب و متناسب با خطای رخ داده شده استفاده شود. البته نیازی به به استفاده از همه کدها نمی‌باشد. برای مثال در API ارائه شده توسط Google Gdata بطور کلی ۱۰ کد، در NetFlix در مجموع ۹ و Digg ۸ کد را پوشش می‌دهد. از میان این کدها سه کد زیر بیشترین استفاده را دارند:

جدول ۴ رایج‌ترین کدهای حالت HTTP

کد حالت	عملیات	توضیح
200	OK	موفق بودن درخواست- همه چیز به خوبی کار می‌کند
400	Bad Request	مشکل در کلاینت- خطایی در سمت برنامه رخ داده است
500	Internal Server Error	بروز خطای داخلی در سرور

در صورتی که نیاز به ارائه کدهای خطا با جزئیات بیشتری باشد می‌توان از ۵ کد زیر بهره برد:

جدول ۵ کدهای حالت HTTP

کد حالت	عملیات	توضیح
201	Created	درخواست انجام شده و مقدار مورد نظر ساخته شد
304	Not Modified	درخواست انجام نشده و تغییر ایجاد نشد
404	Not Found	منبع مورد نظر یافت نشد
401	Unauthorized	اجازه دسترسی به منبع مورد نظر (بدلیل عدم شناسایی کاربر) وجود ندارد
403	Forbidden	اجازه دسترسی به منبع مورد نظر وجود ندارد و سرور به آن پاسخ نخواهد داد

توضیحات جامعی در رابطه با کدهای حالت در سایت ویکی پدیا و restapitutorial.com وجود دارد.

در زمان نمایش خطا در قسمت Payload بهتر است تا آنجا که می‌توان دلایل بروز خطا را به کاربر شرح داد تا کاربر بخوبی بتواند در جهت رفع خطا اقدام نماید. همچنین پیشنهاد می‌شود که لینکی به منظور ارائه توضیحات بیشتر نیز به کاربر ارائه شود.

۵. فرمت پاسخ

در زمان پاسخ دادن به درخواستهای ارسال شده به REST باید در مورد چگونگی ارسال پاسخ به کاربر به مواردی مانند نوع و فرمت پاسخ‌دهی، صفحه بندی و چگونگی ارسال سربارها دقت داشت. در ادامه به مواردی که در این زمینه نیاز به توجه بیشتری دارند، اشاره خواهد شد.

۵-۱. فرمتهای مختلف نمایش داده

روشهای ارائه داده بصورتی هستند که بتوانند شمای کلی موجودیت‌ها را نمایش دهند. این روش‌ها می‌توانند در بدنه پیامهای HTTP قرار داده شوند و داده‌ها را بصورت مرتب شده ارائه دهند. مدلهای زیر مدلهای معمول ارائه داده هستند. در جدول همچنین می‌توان نوع محتویات (Content-Type) مورد استفاده در هرکدام از روش‌ها را مشاهده کرد:

جدول ۶ فرمت‌های مختلف نمایش داده

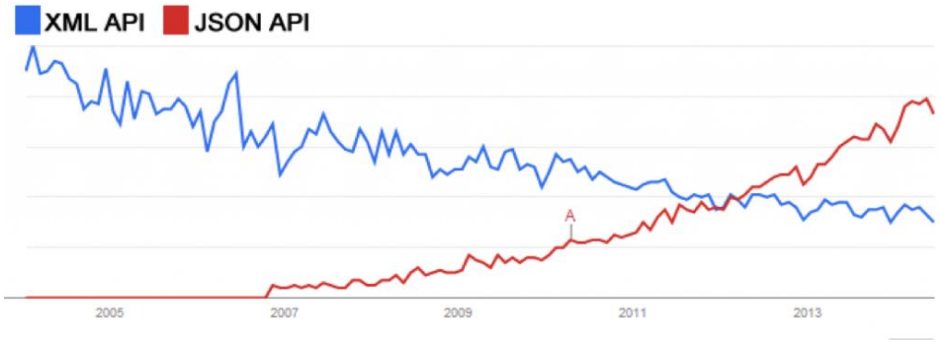
نوع	نوع محتویات
JSON	application/x-resource+json application/x-collection+json
YAML	application/x-resource+yaml application/x-collection+yaml
XML	application/x-resource+xml application/x-collection+xml

HTML	text/html

ارائه داده به فرمت JSON بسیار ساده است و بطور پیش‌فرض داده‌ها بصورت JSON ارائه می‌شوند. فرمت YAML نیز بسیار مشابه JSON بوده و تفاوت بسیار کمی با آن دارد. در مقابل فرمت XML پیچیده‌ترین فرمت ارائه داده هم از نظر پیچیدگی و هم از نظر محدودیت‌های آن می‌باشد. از میان فرمتها مختلف داده در سرویس‌های وب فرمت XML از ابتدای ایجاد پر استفاده ترین فرمت بوده است. برخی از دلیل پراستفاده بودن این فرمت عبارتند از:

- فرمتی ساده، باز، خود توصیف کننده و برپایه متن
- قابل‌خواندن و نوشتن توسط انسان و ماشین
- قابل شناسایی
- دارای مجوعه های بزرگی از انواع داده

XML مشکلات خاص خود را نیز دارد و مهم‌ترین آن کند بودن و هدر رفتن چرخه‌های پردازشگر در زمان تحلیل این فرمت، می‌باشد. علاوه براین فرمت XML به میزان قابل توجهی نسبت به JSON حجیم‌تر می‌باشد. از طرفی به دلیل آن که REST APIها کاربرد بیشتری پیدا کرده و نیاز به فرمت داده‌ای دارند که با حجم کم، سربار پایین، عملکرد بالا و امکان پردازش سریعتر این فرمت، توسط مفسرها کار کنند، این فرمت داده محبوبیت بیشتری نسبت به XML بدست آورد.



شکل ۱میزان محبوبیت فرمت داده APIها از سال ۲۰۰۵ تا سال ۲۰۱۳

در استفاده از فرمت داده‌های، می‌توان به این صورت عمل کرد که فقط یکی از فرمتهای XML، JSON و یا YAML استفاده شوند اما پیشنهاد می‌شود که بیش از یک فرمت داده در API پشتیبانی شود و فرمت پیش‌فرض JSON قرار داده شود.

همچنین در نام‌گذاری صفات و فیلدهای پاسخ تا حد امکان از قوانین نام‌گذاری درست (انتخاب نام‌هایی تا حد امکان مرتبط و قابل فهم و استفاده از روش‌های نام‌گذاری نظیر CamelCase) استفاده نمایید.

۵-۲. صفحه بندی و پاسخی دهی جزئی

در زمان ارسال پاسخ باید دقت داشت که پاسخ‌ها بصورت مختصر و مفید ارسال شوند. این بدان منظور است که بتوان از پهنای باند موجود بطور بهینه استفاده کرد همچنین از منظر نمایش در واسط کاربری برنامه‌ها باید بگونه‌ای باشد که نمایش داده‌ها فضای زیادی را اشغال نکند. در نتیجه باید امکانی به کاربر داد که بتواند نتایج دریافتی را محدود کند در عین حالی که بتواند از میزان کل داده‌ها نیز مطلع شود. به عنوان مثال فیس‌بوک از روش زیر استفاده می‌کند:

```
/joe.smith/friends?fields=id,name,picture
```

که در آن می‌توان دوستان کاربر را با توجه به مشخصات مورد نظر کاربر استخراج کرد. با این عمل تنها اطلاعاتی که کاربر نیاز دارد برایش ارسال می‌شود که این کار از نظر مصرف پهنای باند مخصوصاً در برنامه‌های موبایل بهینه‌تر است. همچنین می‌توان از پارامترهای مانند Offset و Limit برای مشخص کردن شماره رکورد آغازی و تعداد رکوردهای درخواستی بعد، page و RPP[1] برای صفحه بندی رکوردها استفاده کرد.

۶. امنیت

در بخش امنیت API دو موضوع احرازهویت[2] و اجازه دسترسی[3] از موضوعات اصلی می‌باشند.

احرازهویت به معنی شناسایی درخواست دهنده منابع و اجازه دسترسی به معنی تعیین میزان دسترسی‌های درخواست دهنده به منابع می‌باشد. عموماً مراحل اجرای درخواست‌ها بصورت زیر است:

۱. کلاینت درخواستی را ارسال می‌کند که توکن احراز هویت یا بصورت سربار X-Authorization یا به صورت پارامتر درخواست در آدرس درخواست شده (Query Parameter) قرار داده می‌شوود.
۲. سرویس، توکن احراز هویت را بررسی می‌کند و در صورتی که مقدار آن منقضی نشده باشد، آن را تأیید کرده و مقدار آن را ذخیره می‌کند.
۳. سرویس یک درخواست به همراه مقدار دریافتی به سرویس احرازهویت ارسال کرده و داده، مقادیر کاربر، منابع درخواست شده و دسترسی‌های مجاز را دریافت می‌کند.
۴. در صورتی که اجازه دسترسی وجود داشته باشد، سرویس به روند پاسخ‌دهی خود ادامه می‌دهد.

[1] Request Per Page
[2] Authentication
[3] Authorization

۶-۱. احراز هویت

این قسمت را می‌توان به چهار دسته تقسیم کرد:

- پروتکل HTTP روش Basic Authentication
- کوکی و مدیریت نشست
- قرار دادن مقدار توکن در سربار HTTP
- قرار دادن مقدار توکن در آدرس درخواستی

پروتکل HTTP روش Basic Authentication

در اولین مورد از پروتکل HTTP برای ارسال مقدار کلید استفاده می‌شود، این روش ساده‌ترین روش و روش پیش‌فرض برای احراز هویت می‌باشد.

```
GET /spec.html HTTP/1.1
Host: www.example.org
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

این روش در صورتی که از پروتکل HTTPS استفاده نشود بسیارآسیب پذیر خواهد بود و درصورت استفاده از پروتکل HTTPS نیز در برابر حملاتی مانند MIM و Replay Attack آسیب‌پذیر است. در این روش همچنین امکان تعیین نشست برای ورود و خروج کاربر وجود ندارد.

کوکی و مدیریت نشست

در دومین روش همه داده‌ها در کوکی مشخص می‌شوند. کوکی ها توسط سرور مدیریت می‌شوند و در سمت کلاینت فقط کوکی‌ها در هر بار درخواست سرویس ارسال می‌شوند و باز نمی‌شوند.

```
GET /spec.html HTTP/1.1
Host: www.example.org
Cookie: theme=light; sessionToken=abc123
```

روش استفاده از کوکی نیز از پروتکل HTTP بهره می‌برد در نتیجه مانند مورد قبل پروتکل آسیب‌پذیری محسوب می‌شود.

قرار دادن مقدار توکن در سربار HTTP

در سومین روش از یک توکن در سربار استفاده می‌شود . در این حالت از پروتکل OAuth می‌توان استفاده کرد

```
GET /resource/1 HTTP/1.1
 Host: example.com
 Authorization: Bearer mF_9.B5f-4.1JqM
```

این روش بسیار شبیه روش ارسال کوکی است و مانند آن روش دچار آسیب پذیریهایی مانند MIM و Replay Attack است. ولی با استفاده از HTTPS می‌توان تا حدی جلوی حملات را گرفت.

قرار دادن مقدار توکن در آدرس درخواستی

آخرین روش استفاده از پارامتر احراز هویت در قسمت آدرس درخواستی است. این روش بهترین روش از نظر همخوانی با استانداردهای پروتکل REST است و می‌توان با یک مدیریت نشست حداقلی آن را پیاده‌سازی کرد.

```
GET /object?apiKey=Qwerty2010
```

در حال حاضر روش‌های basic Auth و OAuth در صورت استفاده تحت گواهی SSL روشهای قابل قبولی هستند که پیشنهاد می‌شود از آن‌ها در طراحی API استفاده شود. از نظر میزان امنیت روش HTTP Digest Access Authentication دارای بالاترین ضریب امنیتی میباشد که می‌توان برای اطلاعات بیشتر به لینک ۳ مراجعه کرد.

۶-۲. اجازه دسترسی

ارائه سرویس اجازه دسترسی در API با اجازه دسترسی در برنامه‌ها تفاوتی ندارد. در این حالت می‌توان از یک لیست کنترل دسترسی (ACL) برای هر متد استفاده کرد. موارد امنیت دیگری که در مورد سرویسهای برپایه Rest حاکم است:

- مقادیری که وارد می‌شوند بعنوان پارمتر ورودی و ورودی های JSON و XML قبل پردازش اعتبارسنجی شوند
- در برابر حملات SQL Injection محافظت ایجاد شود
- داده‌های خروجی توسط کتابخانه‌های anti-xss رمزگذاری شوند.
- اندازه پیام به مقداری معین محدود شود.
- فعالیت‌های مشکوک لاگ شوند

بوسیله استفاده از یک سرویس مدیریت API و API Gateway می‌توان براحتی موارد زیر را انجام داد:

- کنترل میزان استفاده از APIها به منظور آنکه بتوان فعالیتهای عادی و غیر عادی را شناسایی کرد
- محدود کردن استفاده از API در زمان درخواست بیش از حد به منظور جلوگیری از حملات DOS
- انجام احراز هویت توسط سرویس مدیریت API در نتیجه ذخیره کلیدها در سمت Gateway که به امنیت بیشتر کلیدها کمک می‌کند.

در بخش بعد قابلیت‌های زیرساخت مدیریت API تا حدودی شرح داده خواهد شد.

۷. طراحی زیرساخت‌های API

از دیگر ملاحظات طراحی API، ملاحظات مرتبط با زیرساخت نظیر caching ، کنترل نرخ و سطح دسترسی، امنیت و توزیع بار است.

۷-۱. سیاست‌های Caching

از دید سیستم و کاربر استفاده کننده از API هر چه API سریعتر پاسخ دهد بهتر است. در صورتی که API کند عمل کند به همین میزان برنامه کاربردی که از API استفاده می‌کند نیز سرعت کمی خواهد داشت و متعاقباً کاربر نیز ناراضی خواهد بود و ممکن است از برنامه دیگری که سریع‌تر عمل می‌کند، جهت رفع نیاز خود استفاده کند. مکانیزم Caching با کاهش زمان پاسخ می‌تواند یکی از راه‌کارهای مناسب جهت حل این مساله باشد. البته ذکر این نکته ضروری است که امروزه زیرساخت‌های API لایه‌های مختلفی را تعریف کرده اند که هر لایه Cache خود را دارد. مسلماً بسته به نوع محتوا پیچیدگی فرایند caching نیز متفاوت خواهد بود. برای مثال اطلاعات مربوط به پیش‌بینی آب و هوا عموماً برای یک روز معتبر هستند و cache نمودن آن بسیار آسان است اما این مکانیزم در رابطه با Twitter دشوار است زیرا هر کاربر اطلاعات مربوط به خود را دارد به همین دلیل توییتر لایه‌های مختلفی را برای caching در نظر گرفته است. نکات قابل توجه در طراحی مکانیزم‌های Caching عبارتند از:

- بهتر است از میان APIها مواردی را که بیشترین فراخوانی را دارند لیست کرده و از آن میان APIهایی را که کندتر هستند را به عنوان اولین گزینه‌ها برای استفاده از مکانیزم Caching در نظر بگیرید. به خصوص آن دسته از APIهایی که مرتب فراخوانی می‌شوند و پاسخ آنها به ندرت یا در بازه‌های زمانی طولانی تغییر می‌کند.
- سخت‌ترین بخش Caching، انتخاب گزینه‌ای که Cache شود نیست بلکه تعیین مدت زمان اعتبار داده Cache شده می‌باشد.
- انتخاب داده‌ای که نرخ hit آن پایین است عملاً باعث کاهش کارایی خواهد شد. بنابراین در طراحی موضوع و زمان دقت کنید که Cache طراحی شده دارای نرخ Hit نسبتاً بالایی باشد.

۷-۲. کنترل ترافیک API

کنترل نرخ دسترسی یکی دیگر از نیازهای ضروری مدیریت API می‌باشد. به منظور تعریف سطوح مختلف سطح خدمات، کنترل میزان دسترسی برای کاربران مختلف و همچنین جلوگیری از اعمال بار زیاد بر روی API این ویژگی مفید می باشد. وجود این ویژگی کمک خواهد کرد که بتوان کاربران API را به دسته‌های مختلف تقسیم نمود. برای مثال می‌توان برای توسعه دهندگان و کاربران دسترسی محدودی بر روی API ایجاد نمود تا بتوانند API را تست نموده و برای تعداد درخواست محدودی فراخوانی و استفاده کنند. اما برای بکارگیری جدی و با حجم فراخوانی بالا باید برای مثال ثبت نام کنند و یا حتی پرداختی برای آن داشته باشد (این سیاست‌ها بر اساس استراتژی‌های کسب و کار شرکت تولید کننده API مشخص می‌شود.) البته در طراحی سطوح دسترسی و Quota ها به ازای هر بخش از کاربران باید دقت فراوانی به خرج داد تا علاوه بر کنترل ترافیک API سبب جذب مشتریان شود و نه اینکه با اعمال محدودیت‌های بیش از حد باعث از دست دادن کاربران و مشتریان API گردد.

۷-۳. زیرساخت مدیریت API

در بخش قبل به طور مختصر بخش امنیت شرح داده شد. دو مورد Caching و نرخ دسترسی نیز به عنوان دو نمونه از ملاحظات ضروری زیرساختی در طراحی API به طور اجمالی بررسی گردید. علاوه بر موارد فوق چالش‌های بسیار دیگری نظیر مدیریت چرخه زندگی API، مکانیزم‌های پرداخت، رصد میزان استفاده و مانیتور نمودن API، بررسی و تحلیل اطلاعات API و... نیز وجود دارند. برای حل این دست چالش‌ها نیاز به وجود پلتفرمی یکپارچه برای مدیریت API ها است تا بتواند کل چرخه زندگی API را مدیریت نمایند. این پلتفرم‌ها را زیرساخت مدیریت API می‌نامند. به طور کل کارکردهای مورد انتظار از این پلتفرم عبارتند از:

- قابلیت انتشار و عرضه انواع APIها (و استانداردسازی عرضه APIها و فراخوانی آنها تا حد امکان)
- مدیریت سطح دسترسی: احراز هویت و مجوز دسترسی سیستم‌ها به یکدیگر و کاربران به سرویس‌ها
- مدیریت نرخ دسترسی و اعمال سیاست‌ها و قوانین به کارگیری
- مدیریت سرویس‌ها شامل: پایش و نظارت (monitoring)، ثبت حداکثری وقایع (Logging)، نسخه‌بندی و مدیریت ویرایش، مدیریت پشتیبانی، مدیریت کاربران (Accounting)
- مدیریت و توزیع بار در جهت حفظ حداکثر گذردهی
- اندازه‌گیری و تحلیل میزان مصرف هر کدام از ذی‌نفعان (شرکاء/ توسعه‌دهندگان/ کاربر نهایی) و تولید قبض
- گردآوری و تحلیل داده‌های آماری کلان سیستم

ابزارهای مدیریت API به‌منظور ارائه سرویس‌هایی قابل‌اطمینان، امن و مستند شده به کاربران و توسعه‌دهندگان بکار برده می‌شوند. از دیگر ویژگی‌های این زیرساخت‌ها افزایش دسترس‌پذیری[1]، امکان جستجو سرویس‌ها[2]، افزایش امنیت، مانیتور کردن، هشدار دادن، حسابرسی[3] و مدیریت SLA به‌طور یکپارچه می‌باشد.

[1] accessibility
[2] discovery
[3] auditing

مراجع

1. “API Architecture: The Big Picture for Building APIs”, Biehl M., 2015, API-University Series Book2, England: S.N.
2. "APIs A Strategy Guide“ , Jacobson D. , Brail G., Woods D., 2012, O’REILLY , Inc.
3. "REST API Design Rulebook“, Masse M., 2011, O’REILLY, Inc.
4. “Web API Design - Crafting Interface that Developers Love ”, B. Mulloy, 2012, apigee
5. “Building Successful APIs”, akana, 2015
6. “RESTful Service Best Practices- Recommendations for Creating Web Services”,T. Fredrich, P. eCollege, 2012, RestApiTutorial.com
7. "API Evangelist: API Industry Guide, ALI Definition“, Lane K., 2015,
8. “Building APIs You Won’t Hate”, P. Sturgeon, 2013, Leanpub
9. “Thoughts on Restful API design, Release 1.0”, G.Jansen, 2015
10. http://programmableweb.com
11. https://github.com/ServiceStackV3/mythz_blog/blob/master/pages/154.md
12. http://json.org

پیوست

یک سرویس وب قسمتی از نرم‌افزار یا یک سیستم است که دسترسی به سرویسش از طریق یک آدرس بر روی شبکه جهانی وب (اینترنت) امکان‌پذیر است. این آدرس به‌عنوان URI[1] یا URL[2] شناخته می‌شود. نکته مهم در اینجا آن است که سرویس‌های وب اطلاعاتشان را به شکلی ارائه دهند که برنامه‌ها بتوانند آن را متوجه شده و تجزیه و تحلیل کنند. انواع روزمره‌ای از APIها توسط سرویس‌های وب استفاده می‌شوند شامل مواردی نظیر SOAP، RPC و REST می‌باشد.

پیام‌های RPC معمولا در سیستم‌های توزیع شده کاربرد دارند. این گونه پیام‌ها بگونه‌ای عمل می‌کنند که یک پروسه یا روتین را در یک فضای آدرس دیگر (ماشین و یا شبکه دیگر) اجرا می‌کنند. معماری SOAP از استایل و چگونگی نمایش داده‌ها در RPC استفاده می‌کند. این معماری به‌وسیله W3C استانداردسازی شده و به‌طور گسترده‌ای برای سرویس‌های وب استفاده می‌شود. SOAP بیشتر برای یکپارچگی ابزارها درون یک سازمان استفاده می‌شود. SOAP پروتکلی است که متد ارتباطات، ساختار پیام‌رسانی را تعریف می‌کند. این در حالی است که REST یک معماری سرویس بر روی پروتکل HTTP است و از سادگی و کاربری خوبی برخوردار است.

از منظر کارایی، معماری REST از دو پروتکل RPC و SOAP کارایی بالاتری دارد زیرا در این معماری، نوع فرایند از طریق header مشخص می‌شود و در اصل در این معماری، چارچوب طوری تعیین شده است که فرایند های پیچیده به فرایندهای کوچک تر شکسته شوند و در نتیجه کارایی افزایش می‌یابد. از منظر انعطاف‌پذیری RPC وSOAP از REST بهتر می‌باشد زیرا هر نوع فرایندی را بدون محدودیت می‌توان پیاده نمود. در حقیقت موازنه‌ای مابین کارایی و انعطاف‌پذیری وجود دارد با افزایش هر چه بیشتر انعطاف پذیری و شخصی سازی متدها و پیاده‌سازی بر حسب تمایل کارایی کاهش خواهد یافت. بنابراین نیاز است تا ما بین کارایی و انعطاف پذیری موازنه‌ای برقرار گردد تا نتیجه مناسب به‌دست آید. از منظر ماجولاریتی SOAP از دو پروتکل دیگر بهتر است زیرا قسمت های مختلف یک پیغام را از هم جدا و تفکیک کرده است.

[1] Uniform Resource Identifier
[2] Uniform Resource Locator

شکل ۲میزان محبوبیت API های برپایه سرویس وب

همان طور که در شکل فوق مشاهده می‌شود دو معماری SOAP و REST استفاده و کاربرد بیشتری نسبت به RPC دارند. و امروز معماری REST رشد چشمگیری داشته و کاربران بیشتری از آن استفاده می‌کنند. این دو معماری نسب به هم مزایا و معایب خود را دارند که در ادامه به شرح این موارد می‌پردازیم. مزایای پروتکل REST عبارتند از:

- استفاده از استانداردهای HTTP
- سادگی و راحتی کاربرد
- مستندات ساده و قابل‌فهم
- برقراری ارتباطات به‌صورت Stateless (یعنی تغییر حالت سرور تاثیری در پاسخگویی به کلاینت نخواهد داشت.)
- امکان ذخیره سازی[1] پاسخ متدها: در صورتی که داده‌ها در سمت سرویس وب تغییر داده نشوند و پویا نباشند، این ویژگی می‌تواند بر روی کارایی و سرعت پاسخ‌دهی تاثیر بسزایی داشته باشد.
- سازگاری با وب‌سایتها و برنامه های کاربری مختلف: برای استفاده از سرویس‌های REST نیازی به کد نویسی از ابتدای برنامه نیست و فقط توابع مورد نظر اضافه خواهند شد.

در مقابل این مزایا، معایب REST عبارتند از:

- این پروتکل و متدهای مرتبط با آن ممکن است برای انسان معمولی و قابل استنتاج باشد ولی برای ماشین براحتی قابل پردازش نیست و نیاز به تعریف متدهایی برای شناسایی لیستها، تواناییها و الگوهای هرسرویس وب به‌طور جداگانه است. که برای حل این مشکل تا حدودی از زبان‌های توصیف استفاده می‌شود.
- نیاز به تعریف نقاط دسترسی و آدرسهای مختلف برای انجام متدهای مختلف

در مقابل پروتکل SOAP قرار دارد که نسبت به پروتکل REST با هدف ارائه منطق برنامه و نه داده برنامه به‌عنوان سرویس ارائه شده است. SOAP بر روی دسترسی به نام عملگرها از طریق واسطهای مختلف متمرکز شده است. به عنوان مثال متد

```
getUser(User);
```

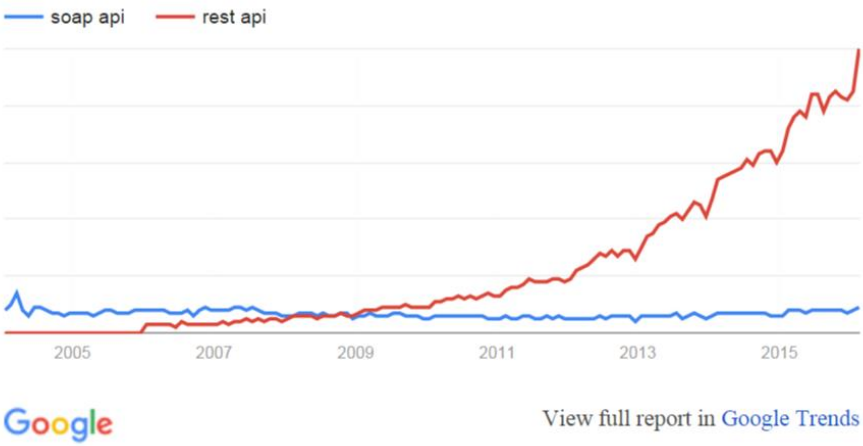
یک عملگر REST است که از طریق آن منابع (داده) در دسترس خواهند بود و متد

```
switchCategory(User, OldCategory, NewCategory);
```

یک عملگر SOAP است که در آن عملیاتی (در اینجا تغییر نام دسته) صورت گرفته است.

در SOAP می‌توان به‌طور SateFull با سرور در ارتباط بود در صورتی که REST به‌صورت Stateless عمل می‌کند. SOAP بگونه‌ای طراحی شده است که مدیریت اتصالات بین سرور و کلاینت را خود انجام می‌دهد.

[1] Caching



شکل ۳میزان محبوبیت دو پروتکل REST و SOAP