# Testing Restful Webservices or API Testing

I have been working on API testing for sometime now. To be specific, I have been testing RESTful web services. I like the idea of REST and from what I have seen, many projects will move in this direction. In few projects I have been working on, RESTful web services are becoming backbone for mobile apps and web clients. If RESTful web services is a new concept for you, you can either dive deep down in the original thesis by Roy Fielding or get a quick overview here.

I like testing applications with RESTful interfaces as their backbone. Often these applications are more testable than tightly integrated applications because interfaces (web services or APIs) used by clients (Web, mobile apps, support or public) are available to testers as well.

In this post, I will explain the model / checklist I follow to test RESTful Web Services.

I usually test APIs for three broad categories

Functionality

Consumability

Maintainability

In my opinion, for any APIs to be successful, it has to be consumable and maintainable along with functional. Let me explain what I mean by these three things in a bit more detail.

Is API functional? Testing functionality of APIs

Functional is a very broad term. For me it covers everything from security to behaving in an expected manner. There are many things I normally check under this category. Instead of explaining, I will list kind of questions I usually ask (to API, developers, PO - whoever can answer) to explore relevant things for every category.

Paging Is this API returning list? What’s the size of list? Does it support paging? Is there default size for paging? Can I control how many results I get in every page? Could it be different for every user? How does API get this information for user? Are there any constraint for different consumers such as web, mobile or tablet?

Authentication Who can access this API? How they are authenticated? How authentication is maintained for subsequent calls? Does API support multiple authenticated sessions for the same users? What about multiple authentications from the same machine? What should happen if user is logged out from one device or tab or browser? How many times a user can fail authentication? For how long authentication would be valid?

Parameters / query strings How are we sending data - Payload or parameters in the query string? What is mandatory? What is optional? What is validated? Is it validated for types, null, or correct values? What defines correct values? What if those values change? Do I need to escape certain characters? Can I use unicode characters?

Authorisation Who can access this API? How access is limited? How can I get the privileged access? What’s the risk of someone getting unauthorised access? What is API protecting by authorisation? Is there hierarchy of access or different access for different roles? How many levels of authorizations are there? How authorisation is enforced at the system level?

Security Are there any private APIs? Are there any undocumented accessible APIs? Is there any encryption? Is API key or any other sensitive information present in the URL? Is this API accessible over SSL? Is there any other form of encryption implemented at transport layer? What HTTP methods are allowed for a given end-point? How is this API protected from XSS - Cross Site Scripting? How it is protected from automated / unauthorised usage? How XML or JSON for the response is constructed? Can this API be used to attack the system? Can this API be used by competitors? Can this API be used to get information about internal systems, architectures or data?

Behave Is API behaving as expected? Is it functioning as designed? Is it meeting the expectation of users? Is it behaving under expected load? Is it performing satisfactory at expected load levels? What content type does it support? Is it tested for XML and JSON both? Is the content type header present?

Error Handling Is API giving right error codes? Is it giving right HTTP codes as well? Is input validated? Is input validated for correctness? Does it handle missing parameters? Does it handle wrong input? Does it issue appropriate error if wrong content type is requested? Does it block users after issuing same kind of errors? Are there any asynchronous calls - if yes what if there is any error in that? Are these errors logged? What if whole system or some part of the system is unavailable, how would it affect user? What if system crash during transaction, how would it recover? What error would it give?

State How state is maintained in the API? For how long a given state is maintained for the user? Can I use state blob for other user and pretend? How tokens are managed? How tokens and API keys are related?

Filter What different filters are supported for the given resource / result set? How will user know about supported filters? Are these filters case sensitive? Are we sanitising the filters? Is API removing leading / trailing white spaces, encoding or using as it is? How do we handle error conditions? How many records can be filtered? Is caching involved in any way?

Order Is it possible to get an ordered list? What parameters are supported for ordering? Can API limit number of results? What sort of validation is present around ordering?

This checklist often serves as a nice guideline for me to ensure that I do not miss obvious things while testing APIs. Not all of them are relevant for every API and there may be many more relevant things to test for other APIs - but it’s a good start nevertheless. I created PAPAS BE @ SFO mnemonics to remember this checklist.

This checklist covers functional aspects - let’s explore few things which will make APIs more consumable.

Is API Consumable? Testing APIs for consumability

How easy it is to consume APIs? Unlike, web, desktop or mobile applications, APIs does not have a graphical user interface. Consumer of the APIs are often developers and so the API has to be designed in such a way that it becomes easier for developers to consume it. Let’s explore some of the factors which will make API testing easy. I will follow the same approach and instead of explaining what I mean by every category, I will list questions I tend to ask under that category.

Domain Specific Names What’s the purpose of this endpoint? What operations (GET/POST etc.) does it support? Is this name in line with the name used by business? Is this name in line with what existing dev community is using? Does this name (along with operation) meaningfully convey what is achieved by this API?

Examples Are there any example for developers to look? Are there any case studies? Are these examples available in the preferred language used by developers? Are these examples working? Is there anything in the examples that will not work for developers? Are example available for every version?

Easy to learn Is it easy for developers to use what they already know about this API? Does this API re-learning? Does this API changes the way dev community used to think about an existing concept?

Documentation Does documentation easily accessible to developers? Is it available for every version? Does documentation reflect functionality? Is it possible to link documentation with the examples?

Hard to misuse Is it possible to use API in the wrong way? Does API guides or restrict developers in a meaningful way? Are there multiple ways to achieve same thing? If yes, why? Are there any chained calls? Are there any private APIs which can be called by some public APIs? Is there any undocumented side effect of calling any API?

Ease to use How easy it is for developers to use? How response is constructed / structured? Can developers request right content type? Is information / structure / naming consistent across every response? Does API require same kind of processing for many calls or such things are handled at the API level? Does API guide users if there is any problem? Is API is consistent with the documentation and examples?

Lead to readable code Does API consumer code looks simple? Is complexity hidden behind the interface or consumers has to write complex code to consume API? Does API names give enough information to users to sense what might be achieved by the API?

Principle of Least Astonishment / Surprise Are there any side effects of any API calls? Are there any constraints developers should be aware of? Are there any magic sequences, magic numbers or hidden implementations in the APIs? Does it behave as documented? Does API usage is in-line with the existing understanding or knowledge of developer? Is there anything developers need to re-learn as part of using this API? Does API follow existing conventions?

Guessability Is it possible for us to guess endpoints based on our existing knowledge and pattern? Is it possible to guess parameters and query strings? Is it possible to guess what would be valid or invalid value for a given parameter?

Consistency Is this API consistent with the older version or existing knowledge of developers? Does it use same kind of validations, give similar error codes and messages for all APIs or is it different? Wherever it is different, is it because of a legitimate reason?

Like functional, this checklist helps me ensure that APIs are consumable. If APIs are not consumable and developers avoid it - it’s a failure. I use DEED HELP GC to ensure that APIs are consumable.

Is API Maintainable? API Testing for maintainability

In my opinion there are two aspects for maintainability in API testing world - How easy it is to maintain APIs as a developer and how easy it is for consumers to maintain their applications built on this API.

Let’s explore some of the factors I find interesting for testing maintainability of APIs for developers and consumers.

Diagnostic How can developers identify if there is any problem in the API? What sort of guidance is available to them? Is there any health check or status board for developers? Is this covered in the documentation? How easy it is for developers to debug? How easy it is for API developers to replicate issues?

Versioning Is it possible for developers to use specific version of API? How developers will be affected by future updates? Would developers be forced to update their applications? Would it be possible to support multiple concurrent versions? How developers will know if they must update to a given version? What happens if you choose not to support a given version - would that be deployed but not supported or would it be taken down?

Logging What would be logged? Who will be able to access it? How will they access it? What is the purpose of logging? For how long logs would be retained? Does it contain any sensitive information?

Accessibility What aspects of APIs are visible to developers? Is accessibility relaxed or strict? Can devs access classes or methods they are not supposed to use? Can dev access parts of the system directly by avoiding APIs?

Purpose What is the purpose of this API? Does it serve any meaningful functionality? Is this API needs to be called in sequence? Who will use this API and in what circumstances? What would they achieve by calling this service? Are there other API calls which can serve the same purpose? Is there other ways to achieve same purpose?

Consumer Who will use this API? Do we need to know if it is called from mobile, web or tablet? Could it be another machine or an automated program? Do we expect consumers to follow specific protocol? Should they give data in JSON / XML? Does it matter? How often they can call? How frequent they can call? Would they wait for our response? If yes, for how long?

This checklist helps me ensure that APIs are maintainable. Mnemonic I use for maintainability is - DVLA PC

As mentioned earlier, this list is in no way complete, but it's a good starting point for API Testing. It's easy for me to check APIs against

Functional - PAPAS BE @ SFO

Consumable - DEED HELP GC

Maintainable - DVLA PC

Did you find it useful? What else do you test for APIs? Can we make these mnemonics better? Let me know in the comments. Please share information about this article in your network. Thanks.