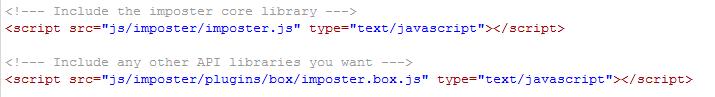
Imposter – JSON API Testing Library  
By: Kenji776 / Dan Llewellyn

**Description:** Imposter is a simple library that supports plugins for testing javascript applications tha rely on APIs that return JSON that cannot be reached normally due to browser security settings. It works by intercepting http requests from the browser, checking to see if they match the signatures of any of the defined plugins, reading JSON template from a file, performing desired translations and returning it in the response text of the http request. It allows for seamless testing of any JSON API without having to worry about proxies and such. With its template engine and unit testing methods you can easily create JSON that matches what the real endpoint would return and compare the generated results against expectations to find discrepancies.

**\*PLEASE NOTE\* Imposter will not run on a local machine due to the way the JSON template files must be loaded. It must be run on a web server. For this reason the lightweight mongoose webserver is included with this distribution. Please run mongoose with the sample file in the same directory and access the test file via** [**http://localhost:8080/**](http://localhost:8080/)

**Installation:** Imposter is composed of a core file that contains the interception, translation, unit testing and other basic features. Support for API’s is added via plugins, so at minimum to use imposter you will include the core library, and one plugin that represents the API you wish to test against. Download the files and include them in your project folder. Then include the scripts in the page you wish to test.  
  
For example to include imposter and the box plugin you would include them like this.



**Configuration:**

Once the scripts are included, you will need to initialize imposter. This is done by calling the createImposter function in the imposter namespace. Like so



From then on, any endpoint registered via one of the imposter plugins will be intercepted and handled by the plugin returning the template JSON as specified. There is no need to modify your HTTP requests and it should be compatible with any framework that issues requests using the XMLHttpRequest Javascript object. Once imposter has been loaded you may enable or disable it setting the imposter.active flag to true or false.

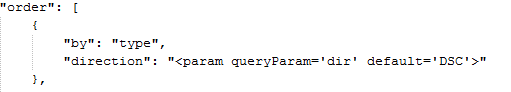
Imposter only has a few public methods which are outlined below.

|  |  |  |
| --- | --- | --- |
| **I want to** | **Use** | **Description** |
| See all the endpoints registered in imposter | imposter.endpoints(‘get’); | Returns an object with every registered namespace as a key/property. All the data for that plugin is held within the key. |
| Remove an imposter endpoint | imposter.endpoints(‘remove’,endpoint); | Removes an endpoint from imposter. The endpoint object passed in only needs to contain the namespace to which the endpoint belongs. |
| See if imposter contains an endpoint for a url | imposter.endpoints(‘contains’,endpoint); | Checks to see if any of the registered endpoints contains the url inside the given endpoint. The only only needs to contain the url to look for. |
| Add an endpoint to imposter | imposter.endpoints(‘add’,endpoint); | Given a valid imposter object |
| Stop all imposter interception | Imposter.active = false; | Stops any interception of HTTP traffic. |

**The JSON Templates:**

Any request that is handled by the plugin will fetch the indicated JSON template file and then parse out the template and replace them with information included in the request. Replaceable params are indicated in the JSON template files using a simple HTML like syntax.

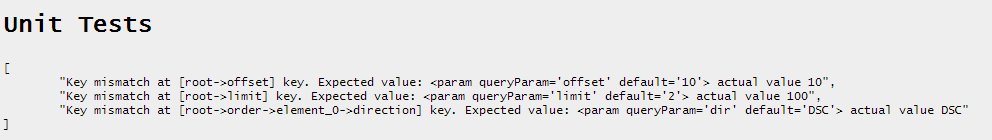
For example



Would indicate that the direction value in the JSON file should be replaced by whatever value is passed in the dir query parameter of the originating request. This allows you to modify the JSON template on the fly for each issued request further enhancing the robustness of the plugin. The template params take only two arguments, a queryParam to replace the placeholder with, and a default value to use if that param is not included in the request.

**Unit Testing:**

Imposter includes a very basic unit test mechanism that allows you to compare the actual values and structure of a JSON response to the one you expected to receive. The unit test method provides an array of messages that indicate differences on the key/value level between the expected and actual JSON structure. When printed they look like this



Here you can see that the actual JSON differed from the template in the ways we would expect. The parameter tags have been replaced with their actual values. We are told which key/subkey was different, the original/expected value and the actual. As you can see differences reported by the unit test should not be considered errors as this case shows. They should instead be considered informational and help you identify replacements performed and assist in verifying proper template translation.

The unit test method takes 2 parameters, the source JSON file to load, prefaced by the namespace separated with a period, and the JSON object to compare the contents of the file to. For example



Shows making an HTTP request, getting the results, inspecting the request to get the namespace and the loaded JSON template (both automatically set by imposter and the box plugin) and then passing the result as well as the source template into the runUnitTest function to get a unit test result.

**Plugin Authoring:**

As we now know Imposter operates using the idea of plugins. Each plugin is expected to represent the functionality of one real API. Every plugin is given its own namespace within imposter where its methods and properties are held and the http requests are passed off to. To create a plugin create a new javascript file and pass an endpoint object into the imposter.endpoints method with ‘add’ as the first argument. Included with imposter is a basic plugin for the box. This plugin shows the basics of adding an endpoint to imposter, and implementing the various HTTP verb handlers. In general every plugin must pass in a url for which it will be intercepting requests, a namespace to store it’s data in, a templatePath which tells imposter (relative to the imposter.js core library) where the template files are stored, a checkAuthorization function which accepts a request object and returns an ‘imposter.requestResult’ object and at least one HTTP verb handler as outlined in the Endpoint Object Description.

**Endpoint Object Description:**

Imposter uses a custom class to represent remote API’s. These objects that contain the various methods and properties used by imposter to operate are called endpoints. An endpoint is defined as follows.

|  |  |  |  |
| --- | --- | --- | --- |
| Property Name | Type | Required | Description |
| url | String | true | The base url of the API for which all request should be intercepted and handled. |
| active | Boolean | false | Is this endpoint actively intercepting requests. |
| namespace | String | true | A namespace to contain all this plugins methods. Should be the same as the name of the folder containing this endpoint file. |
| templatePath | String | true | The folder path relative to the core imposter file where templates for this plugin are located. |
| getHandler |  |  | Function implemented to handle GET HTTP requests |
| patchHandler | function | false | Function implemented to handle PATCH HTTP requests |
| postHandler | function | false | Function implemented to handle POST HTTP requests |
| deleteHandler | function | false | Function implemented to handle DELETE HTTP requests |
| traceHandler | function | false | Function implemented to handle TRACE HTTP requests |
| connectHandler | function | false | Function implemented to handle CONNECT HTTP requests |
| headHandler | function | false | Function implemented to handle HEAD HTTP requests |
| copyHandler | function | false | Function implemented to handle COPY HTTP requests |
| moveHandler | function | false | Function implemented to handle MOVE HTTP requests |
| lockHandler | function | false | Function implemented to handle LOCK HTTP requests |
| unlockHandler | function | false | Function implemented to handle UNLOCK HTTP requests |
| mkcolHandler | function | false | Function implemented to handle MKCOL HTTP requests |
| propfindHandler | function | false | Function implemented to handle PROPFIND HTTP requests |
| proppatchHandler | function | false | Function implemented to handle PROPPATCH HTTP requests |
| checkAuthorization | function | false | Special function that is called before any of the other handlers. This function must an object with a property of ‘success’ set to true to continue processing. Otherwise the content in the ‘data’ property will be returned. It is recommended to instantiate a imposter.requestResult object, modify its properties during processing and return that. This function is intended to simulate an authorization check of any normal API. You may include any logic you like here to simulate authorization checking. |