SpyREST: Automated Documentation for RESTful Web APIs

S M Sohan, Craig Anslow, and Frank Maurer
Department of Computer Science
University of Calgary
Calgary, Alberta T2N 1N4
Email: {smsohan, craig.anslow, frank.maurer}@ucalgary.ca

***Abstract*—Documentation of RESTful APIs are expensive to produce and maintain. It is expensive because there is a lack of reusable tools and automated solutions for RESTful API documentation. Most RESTful APIs are documented manually and the API developers are responsible for keeping an updated documentation as the API evolves making the process both costly and error-prone. In this paper we introduce SpyREST, a reusable tool that can automatically generate RESTful API documentation. SpyREST uses a proxy to intercept example API calls and intelligently produces API documentation for RESTful Web APIs by processing the request and response data. Using SpyREST, RESTful API developers can significantly reduce the cost of producing and maintaining API documentation by replacing a large part of the manual process of documentation with an automated one.**

Keywords*—*RESTful API, Web API, Documentation, Automation, Example based documentation

I. Introduction

RESTful APIs, introduced by Fielding, are used as a primary interconnection mechanism among modern day Web based systems [1]. For example, the website of a restaurant can use the RESTful API from Twitter to show the latest tweets mentioning the restaurant so that prospective customers can read the experience shared by others. To allow others to use their APIs, Twitter and other RESTful API developers publish documentation describing different features of their RESTful API. The documentation of such RESTful APIs are commonly produced and maintained using a manual process.

API documentation for library APIs, such as Java Standard Edition, commonly leverage reusable tools such as JavaDoc. The documentation produced by such tools include description of objects and methods, with custom texts primarily sourced via comments. On the other hand, RESTful APIs documentation includes additional information such as HTTP headers, request parameters, request and response data in serialized formats such as JSON, XML that cannot be easily derived from looking at the objects and methods in the source code. Using comments for these additional information also requires significant manual effort because there is a lack of reusable tools to automate the documentation process. This makes the task of RESTful API documentation a costly and error-prone one. In addition to producing the API documentation, API developers also need to publish and often maintain the documentation for multiple versions as the RESTful API evolves. This requires further manual effort because it becomes a continuous process.

To produce RESTful API documentation with information about HTTP headers, URL parameters, request and response data, the manual process is driven by example API calls. Example calls are made and recorded against an API endpoint to gather information for documentation. This process can be described as a six-step process as follows: 1) craft an example call to an API endpoint with required headers, URL parameters and request body, 2) make the call, 3) capture the response headers and data, 4) strip any unwanted data from the captured information, 5) add custom descriptions to the captured data and 6) publish the API documentation. This six-step process is essentially repeated for all API endpoints that are documented.

With SpyREST, we have implemented an innovative solution to largely automate the aforementioned manual process of RESTful API documentation so that all but steps 1 and 5 are automated. Steps 1 and 5 are left to a manual process to allow for a pragmatic solution that leverages computers to automate the repeated part of the process while leaving the rest to humans. Our solution relies on a lightweight HTTP proxy server to intercept example API calls so that all the information about HTTP headers, URL parameters, request and response data can be automatically gathered. The collected data is then processed to present the documentation for RESTful resources under a hierarchy defined by API versions, resources and actions. Sensitive data such as authentication tokens and passwords are automatically filtered and are not captured. Because SpyREST uses a HTTP proxy server, it can generate documentation for all RESTful APIs irrespective of the technology used to implement the API. The resultant is an automated yet customizable, version-aware, collaboration enabled and reusable API documentation software as a service platform that can be used to generate and maintain documentation for any RESTful API.

The implication of SpyREST is two-fold. It helps lower the cost of producing and maintaining RESTful API documentation through automation and provides a shared platform for publishing the documentation of different RESTful APIs.

The remainder of this paper is organized as follows: in the next section we discuss the related work on API documentation. The implementation details of SpyREST is provided next. Then, we present a case study to demonstrate SpyREST in action. Then, we discuss the implications and limitations of our work in the discussion section. We present our core contributions and future work in the conclusion.

II. Related Work

Several papers have been published on areas related to RESTful API documentation that are discussed under the following two categories: general API documentation and RESTful API documentation.

A. General API Documentation

Several papers have studied the documentation of APIs to understand and recommend best practices that are also applicable to RESTful API documentation. Robillard et al. discussed the obstacles that make APIs hard to learn by surveying API developers and users [2] [3]. They found that developers faced severe obstacles learning new APIs due to inappropriate documentation and other learning resources. Robillard's recommendations for good API documentations include the following: include good examples, be complete, support many complex usage scenarios, be conveniently organized, and include relevant design elements. Kuhn et al. discussed the importance of good examples in API documentation as a key recommendation based on a survey of software developers using APIs [4]. In addition to examples, they identified trustworthiness, confidentiality, and limiting information overload as other key recommendations for API documentation. When API documentation is published for external use, Kuhn identified the need for the documentation tools to be able to protect proprietary and confidential information. Hoffman et al. recommended making the API example scenarios to be executable test cases so that a user can execute an API and understand the related business rules [5]. We recognize all these recommendations to be equally important for RESTful APIs. SpyREST automatically generates RESTful API documents from example API calls, with information about the publisher and allows the users to execute the examples following the aforementioned recommendations.

Nasehi et al. performed a case study based on StackOverflow discussions to find out what makes good code examples [6]. They recommended API developers to include a comprehensive set of code examples in the API documentation and the use of wiki-like collaborative tools with online API documentation so that users can ask questions and get answers on officially published API documentation. Parnin et al. found that social media holds a key place in software documentation as it provides additional knowledge about APIs and gives readers a chance to engage with authors of the APIs [7]. They found active collaboration about API related questions resulting into 81% of posts receiving at least one comment with a median of 8 comments per question. Chen et al. recommended integrating crowdsourced frequently asked questions (FAQs) into API documents so that users can easily find relevant discussions when questions arise as they are browsing API documentation [8]. They presented a tool that can embed FAQs into API documents based on a user's browsing behavior. Subramanian et al. presented an automated approach to link API documentation of different Java and JavaScript libraries with code examples that are shared on StackOverflow by the API users so that the collaborative crowd documentation of APIs can be linked with official documentations [9]. To link API documentation with valuable crowdsourced content, the collaborative features of SpyREST allows users to discuss API related questions and answers on the same web pages where the auto generated RESTful API documentations are shown.

Stepalina discussed the advantages of SaaS based solutions for API documentation systems where a web platform can be used to publish many different API documentations [10]. They identified several benefits of such a reusable platform, such as, cost effective yet powerful, platform agnostic and high accessibility, improved document quality, content reuse, automated tools and organization of robust and scalable documentation process. SpyREST is a SaaS based tool that can be used to leverage these benefits as it allows the generation and publishing of RESTful API documentations for many different APIs under a single platform. They also identified security and reliability of such a shared platform as potentially open issues. To overcome these issues, SpyREST allows a self hosted alternative to SaaS model where API developers can get full isolation for their RESTful API documentation.

Several tools exist that help automatic generation of API documentation for local APIs such as JavaDoc [1], RDoc [2] etc. that convert formatted comments from source code into corresponding HTML documentation for the classes and methods. Jadeite is a tool that adds placeholder API objects for library APIs to improve the search and discovery of desired API objects [11]. While these tools have been proven to work for local library APIs, they have a limited applicability for documenting RESTful APIs because HTTP specific information such as request and response headers, url, request and response payloads are not natively supported by these tools. Hence, there exists a gap for tool support to automatically generate RESTful API documentation. SpyREST fills this gap as it automatically generates RESTful API documentation by recording and synthesizing example API calls.

B. RESTful API Documentation

To address the need for a standard format to describe RESTful APIs several related works proposed candidate specifications. Maleshkova et al. analyzed multiple Web APIs to identify the different approaches related to Web API descriptions, data formats, protocols, reusability, granularity, and authentication [12]. They found that a lack of a standard format to document Web APIs and manual documentation led to API under-specification causing confusions about how to use the APIs for different use cases. They identified a need for automated approaches. Espinha et al. observed that most RESTful APIs are documented manually by API developers making the documentation a less reliable one [13]. To standardize the terminology for RESTful API documentation, Danielsen et al. presented a vocabulary for documenting RESTful Web APIs called Web Interface Language (WIfL) [14]. The vocabulary comprises of these following objects: Resource, Request, Response, Representation and Parameters that can describe the structure and example usage of RESTful Web APIs. Generation of WIfL specific objects for RESTful APIs requires manual effort or bespoke implementation since a reusable automated approach is not presented. With SpyREST, we focus on automation so that reusable tools can be used to produce RESTful API documentation.

[1] http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html

[2] http://rdoc.sourceforge.net/

Verborgh et at. presented RESTdesc, a Resource oriented and Hyper-link based specification for describing RESTful APIs [15]. RESTdesc relies on pre and postconditions to describe the outcome of an API call so that a general purpose API client can parse RESTdesc formatted documentation to invoke API calls to specific RESTful APIs. Mangler et al. presented RDDL, a XML based specification for describing RESTful APIs [16]. RDDL descriptions are composed of resources, operations, parameters and headers. Similar to WIfl, manual effort is needed to generate RESTdesc and RDDL formatted documentations.

Kopecky et al. presented hRESTS, a machine readable micro-format to describe RESTful APIs that uses an alternate representation compared to WIfL [17]. hRESTS specification comprises of Service, Operation, Address, Method, Input, Output and Label objects to describe RESTful APIs. Automated XML transformation is used to convert a formatted HTML documentation of a RESTful API into hRESTS so that the machine readable format can be used by RESTful API clients for automated invocation. SpyREST aims to minimize the cost of a manual or custom process to generate and maintain formatted HTML documentation by leveraging an automated approach.

Ning et al. presented OmniVoke, a RESTful API based invocation engine that provides an abstraction layer for RESTful API calls to multiple APIs that follow different conventions [18]. A general purpose RESTful API client can be used against OmniVoke since the different conventions are wrapped under a uniform RESTful API. Manual configuration is required for each existing API to wrap under OmniVoke.

Myers et al. performed a user study based on the documentation of Web APIs used in large scale enterprises to understand the desirable contents for Web API documentation [19]. They recommended providing a consistent look-and-feel with explanation for the starting points and an overall map comprising of both text and diagrams, providing a browsing experience with breadcrumb trail following a hierarchy, an effective search interface, providing example code and a way to exercise the examples online without writing code. SpyREST aims to achieve requirements for RESTful API documentation using an automated yet customizable approach that can be reused across different RESTful APIs.

In addition to the research community, there are several RESTful API specification formats that are observed in the industry. Swagger is a JSON based specification that allows users to describe RESTful APIs in terms of paths, parameters, request and response headers and bodies [3]. RAML is a RESTful API documentation format that uses YAML files to describe RESTful API documentation using a similar hierarchy as Swagger [4]. Blueprint is another RESTful API specification format that uses Markdown files with additional tags to describe RESTful API objects [5]. In addition to producing RESTful API documentation, there are tools and software as a service (SaaS) providers that can be used to publish the documentation and auto generate API client code for RESTful APIs that are described using one of these formats. To use these tools to generate HTML based RESTful API documentation, API developers need to manually construct the intermediate documentation format that is required by these platforms since there is no automated tool to produce this. In contrast, SpyREST generates RESTful API documentation by synthesizing data collected from example API calls without relying on any intermediate representation.

[3]https://github.com/swagger-api/swagger-spec/blob/master/versions/2.0.md

[4]http://raml.org/spec.html

[5]https://github.com/apiaryio/api-blueprint

III. SPYREST

A. SpyREST Requirements

The following list of requirements for SpyREST is derived from analyzing the aforementioned related work and current API documentation practices as observed in the industry:

Automated RESTful API documentation: RESTful APIs are either documented manually or using custom tools to partially automate the process. Both processes can be expensive to maintain since API documentation is a continuous process to support the evolution of APIs. The primary requirement for SpyREST is to find a pragmatic approach to automatically generate RESTful API documentation.

Example based: As discussed in the related work section, several authors have emphasized on including example scenarios with API documentation to help users understand how to use an API [2] [4] [5] [6]. SpyREST generated API documentation needs to include example scenarios.

Executable documentation: In addition to including examples, SpyREST also needs to allow users to execute the example scenarios so that they can try the API features without having to write code as recommended in related work [5] [19].

Version awareness: SpyREST needs to allow API developers to publish documentation for multiple versions of a Web API as it evolves. The following comment from a API user of Stripe shows the importance of a version-aware documentation tool [6] :

> "Does the full API documentation only reflect the current version of the API? Is there a way to access the API docs for outdated versions? ...That would be very helpful. When you are trying to upgrade from one version to another it's impossible to know the implementation differences. We are currently about 4 API versions behind and are stuck behind a version that causes a significant amount of work on our end to support. I'd like to be able to upgrade incrementally through each version."

Customizable: Robillard identified the importance of customized overview information about how to use an API to reduce API learning obstacles [2]. Custom content is also required to explain business rules and overview information about APIs that may not be automatically derivable. SpyREST needs to allow users to add customized content to auto generated RESTful API documentation to enrich the quality of the documentation.

Reusable: SpyREST needs to work as a reusable platform so that multiple REST API documentations can be generated and published on a single platform to get the benefits as outlined by Stepalina [10].

[6]https://groups.google.com/a/lists.stripe.com/forum/#!searchin/api-discuss/version/api-discuss/li4PyVcweiw/NT9SFTtF-vQJ

Collaborative: SpyREST needs to allow people to collaborate on the API documentation so that questions and answers about APIs can coexist with the API documentation to overcome the obstacles with fragmented knowledge sources as identified by Chen et al. [8].

B. SpyREST Components

SpyREST is composed of three main components as shown in Fig. 1.

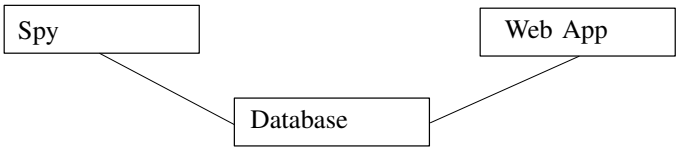


Fig. 1: SpyREST Component Diagram

The Spy: SpyREST is driven by example calls to the RESTful APIs to generate the documentation for the APIs. The Spy component, a HTTP proxy server, in SpyREST is named so because it records the HTTP traffic for example API calls that are made using it as a proxy server. For example, when using Spy as a HTTP proxy to make the following HTTP request:

```
Method  GET
URL     https://api.github.com/repositories?since=100
Headers accept: application/vnd.github.v3+json
```

that produces the following response:

```
Headers status: 200 OK
        content-type: application/json; charset=utf-8
        ...

Body
        [
          {
            "id": 1,
            "name": "grit",
            "full_name": "mojombo/grit",
            ...
          }
        ]
```

Spy automatically saves the raw request and response data with HTTP headers. Additionally, it synthesizes the data and saves the following meta data about this example API call:

version	v3
resource	repositories
action	GET /repositories
query	since: 100
strippedResponseBody	a subset of the response body
description	blank
digest	base64 hash value of the version, resource, url and description
requiresAuth	false
apiToken	blank, used when a SpyREST API token is provided
userId	blank, used when a user is found for given apiToken

As shown in this example, the version "v3" is automatically detected by parsing the accept request header. The automatic version detection algorithm can extract version information from either the request URL or accept header for most commonly used formats. Next, the resource field is also auto detected as "repositories" by parsing the request URL. The action "GET /repositories" is automatically detected by combining the request HTTP method with request path. The strippedResponseBody field automatically saves a shorter version of the actual response where large arrays in the response body are recursively truncated to smaller arrays with two sample items to reduce noise from the generated documentation. These meta fields allow SpyREST to structure the example API calls for a given API host in a hierarchical model as follows: an API has many versions, each version has many resources, each resource has many actions, and each action has many examples. Fig. 2 shows a simplified UML diagram of SpyREST's data model.

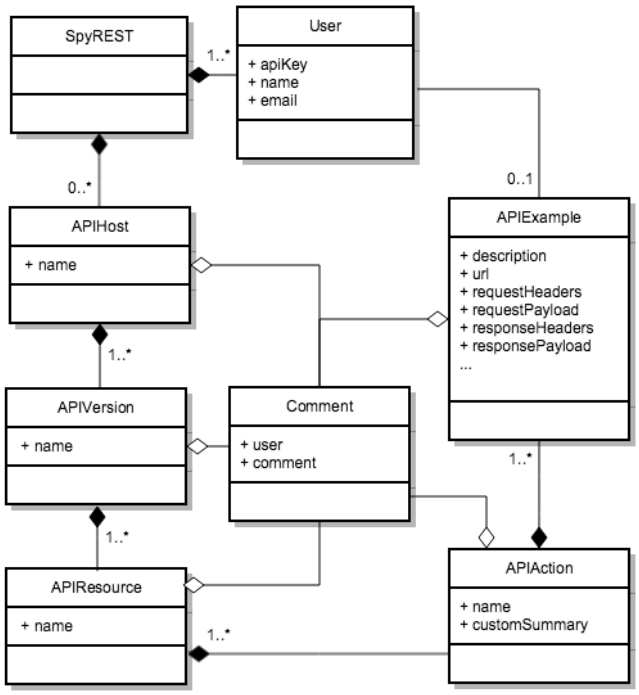
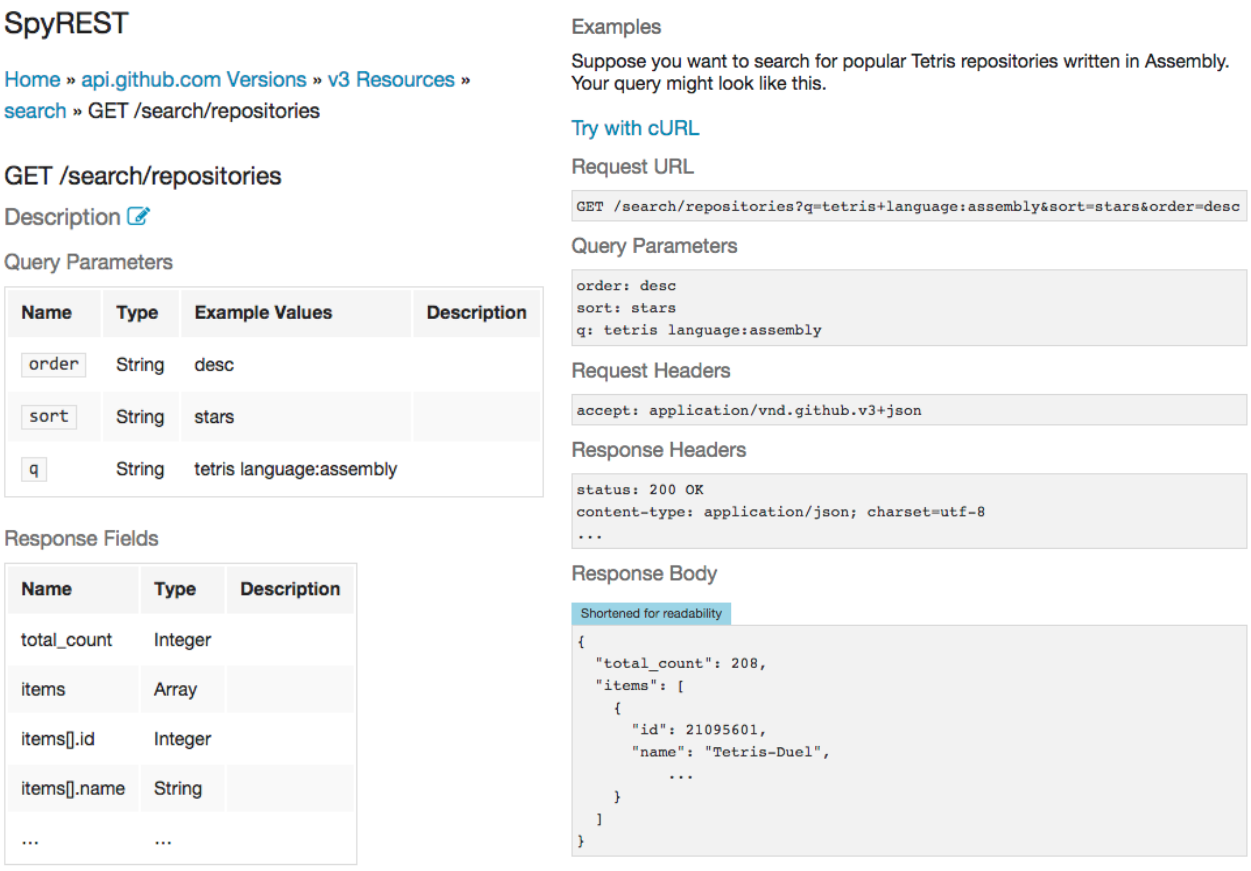


Fig. 2: SpyREST data model

Auto detection algorithms for version, resource and action fields may be overridden by providing SpyREST with additional headers when example API calls are executed. For example, a user can use the custom header "x-spy-rest-resource: repos" when making example API calls to override the automatic detection of resource from "repositories" to "repos". Using a similar approach, users can override the version, action, and description fields using the "x-spy-rest-version", "x-spy-rest-action" and "x-spy-rest-description" headers respectively.

The digest is automatically computed based on the version, resource, url and description fields. Using this digest, Spy allows the users to replay the the same example and update the recorded information without creating duplicate entries for each run of the same example API.

When saving request information, Spy automatically strips off values used in "authorization" request headers to avoid leaking secrets in the API documentation. If an authorization



(a) API documentation summary section

(b) API documentation examples section

Fig. 3: Screenshots of auto generated API documentation from SpyREST

header is found, Spy also marks the example with "requiresAuth: true" so that in the generated API documentation it can be visually indicated.

SpyREST allows users to sign up and get an API key to be used for generating API documentation. The "apiToken" and "userId" fields are populated when a valid SpyREST API key is provided in the "x-spy-rest-api-key" request header. This provides credibility to the documentation generated by SpyREST since the official API developers can be identified by their SpyREST account.

To record data for an example call to an API that uses a SSL/TLS connection, the Spy needs to perform as an intentional man-in-the-middle style proxy since the data would be encrypted otherwise. We consider the security risks for this approach to be minimal when the example API calls are intended to be recorded for public viewing through the documentation. For APIs where this is not acceptable, the Spy can be installed with the required SSL certificates on their trusted infrastructure to overcome this issue.

The Spy component is built using nodejs[7] and open source libraries and SpyREST currently supports JSON based APIs only.

The Database: MongoDB[8], a widely used document oriented database, is used as the database. MongoDB is chosen because of its scalability features that could be used by SpyREST to automatically generate documentation of large number of RESTful APIs in a shared platform.

The Web App: The web app component further processes and presents auto generated documentation that is captured by Spy in the database. It also allows the users to edit or add custom content to the auto generated documentation. Additionally, the web app includes administration features such as user registration and SpyREST API key management. The features of the web app are discussed later in this paper in greater detail. The web app is written using the Ruby on Rails[9] framework primarily because of the first author's past experience using this framework.

The source code for both the Spy and Web app components are released as open source projects. Even though nodejs, MongoDB and Ruby on Rails are used for SpyREST implementation, the ideas that SpyREST realized are not dependent on these specific technologies.

[7]https://nodejs.org/

[8]https://www.mongodb.org/

[9]http://rubyonrails.org/

C. SpyREST Features

Now that we have explained the three SpyREST components and how they work, in the next paragraphs we discuss how SpyREST satisfies the aforementioned requirements.

Automated RESTful API documentation: The work flow for automated RESTful API documentation can be explained by the following steps: 1) API developer uses the Spy HTTP proxy to run example API calls, 2) The Spy records example API calls, 3) The Spy extracts meta information about the API call, 4) The Web app displays the auto generated documentation, 5) API developers can optionally customize the auto generated documentation.

Fig 3 shows screen shots (content and presentation adapted for brevity) of fragments from of an auto generated documentation solely based on an example API call to URL https://api.github.com/search/repositories?q=tetris+language:assembly&sort=stars&order=desc with a custom header "x-spy-rest-description" to provide the short description for the example.

The auto-generated API documentation features a summary section. This section shows a breadcrumb containing the hierarchy of the API objects related to each API action to help users easily navigate the API documentation as a tree. This section also includes three tables that display the structure of query parameters, and request and response payloads. In addition to the structure, automated type detection is used to show the data type for each field in these tables. Example values are also automatically populated for query parameters.

The examples section on the API documentation shows all the recorded API examples for a given API action. For each example, it shows a description, the request URL, query parameters, and request and response headers and bodies. Because Spy filters out the "authorization" request header before saving the examples in the database, the documentation rendered by the web app excludes the secrets. As discussed before, the Spy also computes a "strippedResponseBody" by truncating large arrays in API responses to contain only two samples per array. The web app displays this "strippedResponseBody" to limit the verbosity on the auto-generated API documentation. Not shown in Fig. 3 for brevity, each example in the documentation also includes shows the SpyREST user that ran the API example and the time when it was generated.

Example based: As discussed earlier, SpyREST generated RESTful API documentation includes both the structure of the API objects and concrete examples for different use cases. The API examples are annotated with user provided descriptions through the custom headers that are otherwise hard to automatically infer. Because the Spy computes a hash digest for deduplication, replays of the same API example only updates the auto-generated documentation with the latest information allowing API developers to update the API documentation automatically when the API changes during development time. To record the examples in SpyREST, any REST API client can be used as long as it supports using a HTTP proxy to call the API.

Executable documentation: Because SpyREST keeps a copy of the API request information including URL, headers and request body, it can also recreate the example API calls that can be executed by users of the generated API documentation. Currently, it automatically generates executable test cases that can be run using cURL[10]. cURL is a command line based tool that can be used as a REST API client. The examples section on Fig. 3 shows a link titled "Try with cURL", on click of which the user gets a panel with am executable API call example reconstructed from the recorded data as shown in Fig. 4.

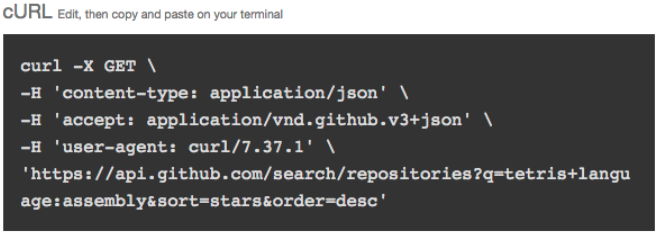


Fig. 4: Executable Examples in SpyREST using cURL

This panel containing the executable API example is editable on the user interface. This allows the users to modify the API call if they need to customize the auto-generated API call. For example, to change the value of "order" query parameter from "desc" to "asc" as shown in Fig. 4, a user can edit the text in this panel before executing the API example in cURL. Although not currently implemented, similar to cURL, SpyREST can generate executable tests targeting other REST API clients and programming languages since the required information to reconstruct an executable test is already available in the database.

Version awareness: SpyREST allows API developers to publish the documentation for multiple version of their RESTful APIs. To organize API documentations and examples under multiple versions, SpyREST has an automatic version detection algorithm that parses the "accept" request header or the URL. Table I shows some examples of auto-detected versions. To cover the commonly used versioning schemes, the version detection logic tokenizes the "accept" header and URL to find segments that are prefixed with a "v" followed by numbers. In case this heuristic fails to detect a version, the examples are recorded under "Default" version. If required, the auto version detection algorithm can be suppressed by specifying the version in the custom "x-spy-rest-version" header when running example API calls through the Spy in.

Type	Example Value	Detected version
"accept" header	application/vnd.ex.ca.v3+json	v3
"accept" header	application/vnd.ex.ca.v3.1+json	v3.1
URL	/v2/x	v2
URL	/v2.1/x	v2.1
URL	/v2.1-pre/users	v2.1-pre

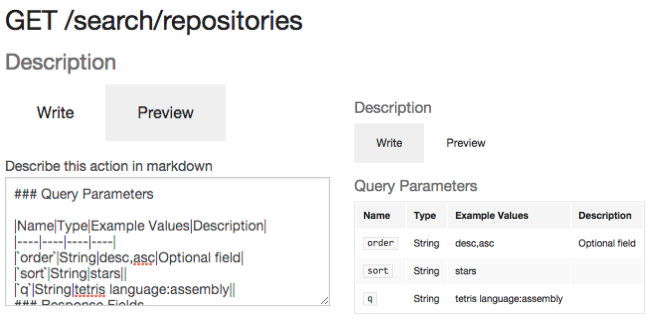
TABLE I: Examples of auto detected versions

Customizable: SpyREST allows API developers to modify and add custom free-form contents to the auto generated summary section of the documentation using Markdown[11] syntax. Markdown was chosen because of its popularity as

[10] http://curl.haxx.se/

[11] http://en.wikipedia.org/wiki/Markdown

a documentation and code sharing tool among software developers on platforms such as StackOverflow and GitHub. Because SpyREST solely relies on example calls to APIs to auto-generate the API documentations, custom content can be used to provide overview information and explain complicated business rules about the APIs that are not derivable from simply synthesizing the examples. We consider this as a pragmatic solution to augment manual effort with the largely automated solution to the RESTful API documentation effort. The summary section on the API documentation page has a button next to the "Description" title that opens up the Markdown editor with preview support as shown in Fig. 5. As shown in this example, a user can annotate the auto-generated documentation for the "order" query parameter as being an "optional field". The custom edits are persisted in the database and are not overridden when the API examples are replayed unless the user decides to revert back to automated summaries.



(a) Editing auto-generated summary (b) Previewing customized content

Fig. 5: Screen shots showing customization feature

Reusable: Because SpyREST only relies on a HTTP proxy to auto-generate RESTful API documentation, it can be used to document RESTful APIs that are developed using any underlying technology. To feed SpyREST with data, any REST API client can be used as long as it supports connecting through a HTTP proxy server. This technology agnostic feature allows SpyREST to be a reusable RESTful API documentation tool.

SpyREST is offered as a SaaS tool on the internet that can be used to auto-generate and publish documentation of multiple RESTful APIs. For APIs where a SaaS solution is not acceptable, SpyREST can also be installed and used as a self-hosted solution. Because SpyREST is an open-source application, users can customize it to support any unique requirements that are not supported by SpyREST.

Collaborative: SpyREST allows the users to comment and discuss API related questions right next to the documentation so that crowdsourced documentations can coexist with the officially published API documentation. The current implementation of commenting features on SpyREST is based on disqus[12] as shown in the Fig. 6. Each API documentation page on SpyREST features its own discussion thread to help users easily locate relevant information.

[12]https://disqus.com/

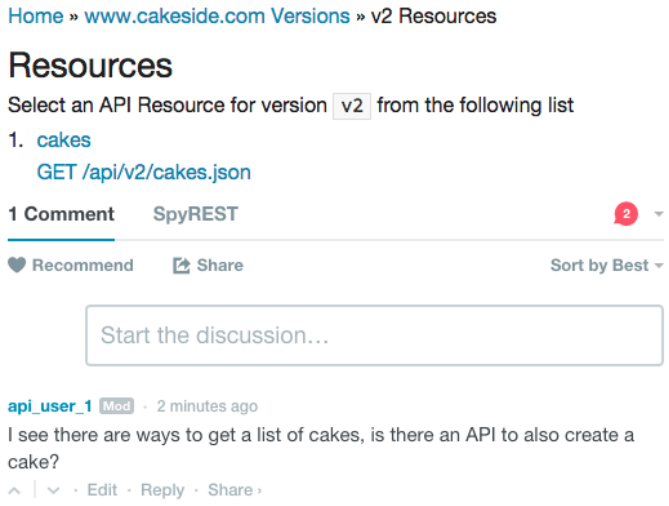


Fig. 6: Screen shot showing commenting within SpyREST

IV. SpyREST In Action

Now that we have discussed the different features offered by SpyREST, in this section we discuss a case study of using SpyREST to auto generate the documentation of one API action from the GitHub API. GitHub currently publishes a manually generated documentation of their API at: https://developer.github.com/v3/.

We start by looking at the manually written notifications list API available at https://developer.github.com/v3/activity/notifications/#list-your-notifications. Using this API, a GitHub user can fetch notifications about different activities related to their account. To auto-generate the documentation for this API using SpyREST, we need to make some example API calls via the Spy proxy. To make the example API calls, in this demonstration, we use a REST client using a ruby program as follows:

```
1 class Github
2   include HTTParty
3
4   base_uri 'https://api.github.com'
5   headers('Accept' => 'application/vnd.github.v3+
        json',
6     'User-Agent' => 'curl/7.37.1',
7     'content-type' => 'application/json',
8     'Authorization' => "token GITHUB_API_TOKEN'"
9   )
10
11  host, port = 'spyrest.com', 9081
12  http_proxy host, port
13 end
14
15 describe 'Notifications' do
16
17   it 'List all notifications for the current user'
       do
18     response = Github.get '/notifications'
19     assert_equal 200, response.code
20   end
21
22   it 'List all notifications including read ones for
         the current user' do
23     response = Github.get '/notifications?all=true'
24     assert_equal 200, response.code
25   end
```

```
26
27  it 'List notifications only where the current user
        is directly participating or mentioned' do
28    response = Github.get '/notifications?
          participating=true'
29    assert_equal 200, response.code
30  end
31
32  it 'List notifications since a time' do
33    response = Github.get '/notifications?since
          =2014-01-01T00:00:00Z'
34    assert_equal 200, response.code
35  end
36
37 end
```

To auto-generate the API documentation for notifications API in GitHub, we first setup the base URL and required HTTP request headers on lines 4-6 inside the `Github` class. Next, we setup the proxy connection to go through the Spy on line 12. Then we provide four example API calls each describing a different use case for the `/notifications` action on lines 15-37 that use the `GitHub` class. These examples are written as automated tests using the RSpec testing framework[13] so that in addition to generating the API documentation, API developers can use the same code as functional tests for the API and vice versa. With this setup, when the tests are run, it executes the example API calls and SpyREST automatically generates the API documentation for the `/notifications` action.

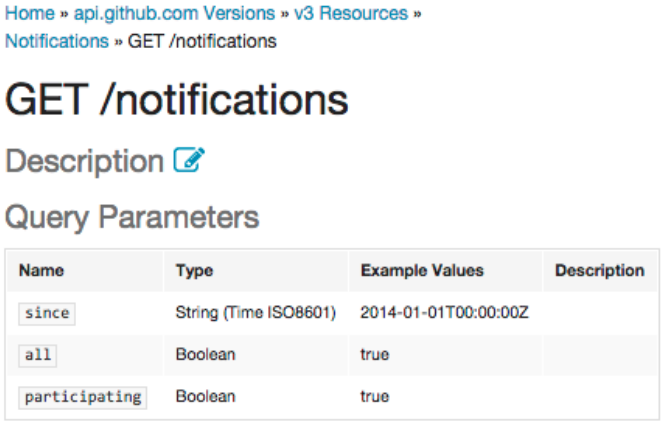


Fig. 7: Screen shot showing URL query parameters

Fig. 7 shows a screen shot of the auto generated summary table for the URL query parameters associated with the `api.github.com > v3 > notifications > GET /notifications` API action. This table merges the different parameters such as `all`, `participating` and `since`, from the four example API calls to the `/notifications` API and displays the auto-detected data type such as ISO8601 formatted time, as well as example values that are captured. Similarly, SpyREST also automatically displays the structure of the API request and response bodies, if any, on separate tables using a similar visualization as shown in Fig. 3. A user can edit this auto-generated summary information on a Markdown editor provided on the user interface to provide further details.

[13]http://rspec.info/

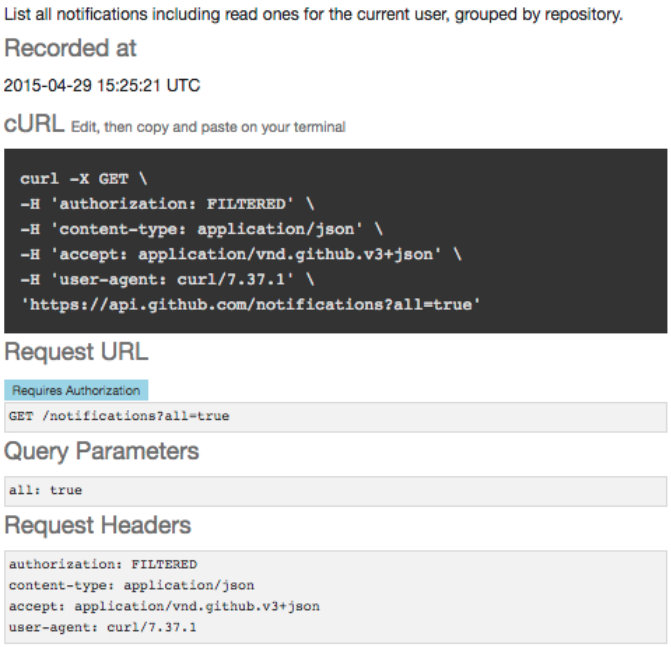


Fig. 8: Screen shot of documentation with an example use

In addition to showing the structure of the API objects, SpyREST also displays the actual data collected from the four example APIs on the auto generated documentation. Fig. 8 shows a fragment of the documentation for the example API call generated from line 22 on the test code. On line 8 of our test code, we set the required `authorization` header for GitHub. The Spy does not save this secret value and the auto-generated documentation displays `authorization: FILTERED` in the Request Headers section. Because of the presence of this `authorization` header, SpyREST also automatically includes a "Requires Authorization" message on the documentation. SpyREST also auto generates the cURL command to allow users to execute this example API call without writing any code. As discussed earlier, SpyREST features a commenting section on every API documentation page to allow users to collaborate and find relevant information on the APIs.

In this case study, we have shown how a total of 29 lines of executable test code, excluding blank lines, has been used to auto-generate the documentation of the `/notifications` action for the Github API. We consider the effort required for writing the test code to be simpler than the effort that'd be required to manually write an API documentation with similar content. Moreover, when the API changes, a replay of the test code will automatically update the documentation.

Even though Ruby and RSpec test framework has been used in this demonstration, any RESTful API client can be used instead. An automated way to run the example APIs are preferred since it is easier to repeat and maintain than a manual approach. We recommend using a test framework because it combines the benefits of both automated testing and documentation from the same effort.

V. Discussion

There is a lack of tool support for automatically generating documentation of RESTful APIs. We presented SpyREST to fill this need. SpyREST demonstrates a novel idea of using HTTP proxies to automatically record and synthesize the recorded data from example API calls to provide a pragmatic solution for automatic RESTful API documentation.In the previous section, we demonstrated a case study showing how to use SpyREST to auto generate documentation for a GitHub API. Similar to this demonstration, RESTful APIs for different applications can be documented from executable code that are easier to maintain over a manual editing process.

The requirements for SpyREST are derived from analyzing the existing literature and industry practices as well as our own experiences developing RESTful APIs and their documentation. In the literature, several papers provided lists of recommendations for API documentation based on studying the existing tools and techniques and feedback collected from API developers. We found a lack of available tool support to generate RESTful API documentation following those recommendations and worked on SpyREST to provide a solution. As a result, SpyREST features such as automated documentation with executable examples, customizable contents, collaboration, version awareness and reusability provide a ready to use tool support for RESTful API documentation following the recommendations.

The novel approach taken by SpyREST over the existing tools from the academia and the industry offers some unique benefits. For example, tools that rely on user provided comments on source code and code inspection, such as JavaDoc, Jadeite, etc. require the user to write formatted comments that are often applicable to a single programming language. To include example usage of the API, the comments need example code that may need to be maintained elsewhere since the comments are not executable. The verbose nature of RESTful API documentation caused by the different parts such as URL parameters, headers, request and response body, and examples makes it difficult to write and maintain as comments in source code. The documentation generated by these comments can go stale if the changes in the source code is not reflected on the comments, since manual effort is needed to keep the two in sync. SpyREST offers a language agnostic solution so that RESTful APIs written in any programming language can be documented using a single tool.

Another key feature of SpyREST is the shared platform for publishing RESTful API documentation so that API developers do not need to maintain a whole website on their own to publish their API documentation. A similar shared platform is provided by several other tools such as Swagger, Blueprint, etc. We identify the key advantage of SpyREST over these tools to be the fact that SpyREST relies on example API calls instead of relying on a custom API specification that are required by these tools. In the literature, several authors have published specifications describing for RESTful APIs. To generate these API specifications, a manual effort is required since there is no automated tools available to do so. SpyREST does not rely on any custom API specification, so the manual effort can be largely avoided. When APIs are documented following these specifications, the documentation produced is machine readable that can be used by API clients to automatically infer some business rules. Although not implemented in the current version, the data captured by SpyREST can be exported to conform to these API specifications. This would allow the API developers to take the exported API specifications in one of these formats so that the manual effort required can be minimized.

Compare with existing literature Compare with existing tools

Still requires manual, why?

12. Provide recommendations (no more than two) for further research. Do not offer suggestions which could have been easily addressed within the study, as this shows there has been inadequate examination and interpretation of the data.

0.3 How is it different from the existing tools?

Swagger RESTdesc RAML API Blueprint

API call – manual/bespoke implementation – API Specification – API documentation API call – automated record – synthesize – API documentation

0.4 Recap of the features

0.3 Limitations ¿ Custom Search ¿ JSON Only ¿ SSL/TLS ¿ Version difference

VI. Conclusion

Acknowledgment

References

[1] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, 2000.

[2] M. Robillard, "What makes apis hard to learn? the answers of developers," *Software, IEEE*, vol. PP, no. 99, pp. 1–1, 2011.

[3] M. Robillard and R. DeLine, "A field study of api learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011. [Online]. Available: http://dx.doi.org/10.1007/s10664-010-9150-8

[4] A. Kuhn and R. DeLine, "On designing better tools for learning apis," in *Search-Driven Development - Users, Infrastructure, Tools and Evaluation (SUITE), 2012 ICSE Workshop on*, June 2012, pp. 27–30.

[5] D. Hoffman and P. Strooper, "{API} documentation with executable examples," *Journal of Systems and Software*, vol. 66, no. 2, pp. 143 – 156, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121202000559

[6] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming q&a in stackoverflow," in *28th IEEE International Conference on Software Maintenance, ICSM 2012, Trento, Italy, September 23-28, 2012*, 2012, pp. 25–34. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/ICSM.2012.6405249

[7] C. Parnin and C. Treude, "Measuring api documentation on the web," in *Proceedings of the 2Nd International Workshop on Web 2.0 for Software Engineering*, ser. Web2SE '11. New York, NY, USA: ACM, 2011, pp. 25–30. [Online]. Available: http://doi.acm.org/10.1145/1984701.1984706

[8] C. Chen and K. Zhang, "Who asked what: Integrating crowdsourced faqs into api documentation," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE Companion 2014. New York, NY, USA: ACM, 2014, pp. 456–459. [Online]. Available: http://doi.acm.org/10.1145/2591062.2591128

[9] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live api documentation," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 643–652. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568313

[10] E. Stepalina, "Saas support in software documentation systems," in *Software Engineering Conference (CEE-SECR), 2010 6th Central and Eastern European*, Oct 2010, pp. 192–197.

[11] J. Stylos, B. A. Myers, and Z. Yang, "Jadeite: Improving api documentation using usage information," in *CHI '09 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '09. New York, NY, USA: ACM, 2009, pp. 4429–4434. [Online]. Available: http://doi.acm.org/10.1145/1520340.1520678

[12] M. Maleshkova, C. Pedrinaci, and J. Domingue, "Investigating web APIs on the world wide web," 2010, pp. 107–114.

[13] T. Espinha, A. Zaidman, and H.-G. Gross, "Web API growing pains: Stories from client developers and their code," 2014, pp. 84–93.

[14] P. Danielsen and A. Jeffrey, "Validation and interactivity of web api documentation," in *Web Services (ICWS), 2013 IEEE 20th International Conference on*, June 2013, pp. 523–530.

[15] R. Verborgh, T. Steiner, D. Van Deursen, S. Coppens, J. G. Vallés, and R. Van de Walle, "Functional descriptions as the bridge between hypermedia apis and the semantic web," in *Proceedings of the Third International Workshop on RESTful Design*, ser. WS-REST '12. New York, NY, USA: ACM, 2012, pp. 33–40. [Online]. Available: http://doi.acm.org/10.1145/2307819.2307828

[16] J. Mangler, P. Beran, and E. Schikuta, "On the origin of services using riddl for description, evolution and composition of restful services," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, May 2010, pp. 505–508.

[17] J. Kopecky, K. Gomadam, and T. Vitvar, "hrests: An html microformat for describing restful web services," in *Proc. of International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, vol. 1, 2008, pp. 619–625.

[18] N. Li, C. Pedrinaci, M. Maleshkova, J. Kopecky, and J. Domingue, "Omnivoke: A framework for automating the invocation of web apis," in *Semantic Computing (ICSC), 2011 Fifth IEEE International Conference on*, Sept 2011, pp. 39–46.

[19] B. A. Myers, S. Y. Jeong, Y. Xie, J. Beaton, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D. K. Busse, "Studying the documentation of an api for enterprise service-oriented architecture," *J. Organ. End User Comput.*, vol. 22, no. 1, pp. 23–51, Jan. 2010. [Online]. Available: http://dx.doi.org/10.4018/joeuc.2010101903