PhUSE 2015

Paper AD08

An API for Your Data

David Brennan, Independent Consultant

ABSTRACT

Serving up your data to web browsers via a web Application Programming Interface (API) opens up many possibilities. Interactive data presentations and graphs can be created using various open-source tools and libraries. This paper presents Mira, an application which creates simple APIs to tabular data.

INTRODUCTION

The use of web APIs has been commonplace for many years, with applications communicating with each other by exchanging data over HTTP in the form of XML or JSON. In recent years, the use of web APIs in the public domain has become more common. A relevant example is openFDA [1], which provides open-source APIs for FDA drug, device, and food data. So, what are web APIs?

From Wikipedia [2]:

> *When used in the context of web development, an API is typically defined as a set of Hypertext Transfer Protocol (HTTP) request messages, along with a definition of the structure of response messages, which is usually in an Extensible Markup Language (XML) or JavaScript Object Notation (JSON) format*

As web APIs have became more prevalent over the years, the number of ways to interact with them has grown and grown. For example, there's a wide range of open-source JavaScript libraries such as jQuery, d3, crossfilter, and DataTables which can be used to create interactive data presentations and graphs. The motivation for creating Mira was the desire to have an easy way to use these tools with arbitrary tables of data.

INTRODUCING MIRA

Mira is a web application developed using the *Ruby on Rails™* web framework [3]. Its goal is to take two-dimensional data tables and provide a simple Web API to their data. It works as follows:

1. Data is uploaded to Mira along with its metadata
2. Using its metadata, Mira imports the data into relational database tables
3. A simple HTTP query interface is created for each database table
4. Various tools are used to access the API to create data presentations and other applications

TABULAR DATA

Mira has been designed to work with a simple data publishing format promoted by the Open Knowledge Foundation [4] called Tabular Data Package [5]. The data itself is assumed to be stored in CSV files and a single JSON file, *datapackage.json,* provides the CSV file metadata, i.e. table and column names, column types, delimiters etc. Once CSV files are uploaded to Mira along with their corresponding *datapackage.json* file, it should have the information it needs to import the data and provide a corresponding API.

As an example, say we have two CSV files, *ADSL.csv* and *ADAE.csv (two ADaM domains, abbreviated for simplicity)*:

ADSL.csv with columns USUBJID, *SEX*, *ARM* and *AGE*:

USUBJID	SEX	ARM	AGE
ABC001-101-01	F	Treatment B	41
ABC001-101-02	F	Placebo	60

ADAE.csv with columns *USUBJID, AETERM, AESTDT* and *AESTDY:*

USUBJID	AETERM	AESTDT	AESTDY
ABC001-101-01	Headache	2014-07-04	-2
ABC001-101-01	Carpel Tunnel	2014-07-19	13

A corresponding *datapackage.json* file might look like the following (some metadata has been excluded for simplicity):

```
{ "name": "Study ABC001",
  "resources": [
  { "path": "ADSL.csv",
    "dialect": { "delimiter": "," },
    "schema": {
      "fields": [
        {  "name": "USUBJID", "type": "string" },
        {  "name": "SEX", "type": "string" },
        {  "name": "ARM", "type": "string" },
        {  "name": "AGE", "type": "integer" }  ]
    }
  },
  { "path": "ADAE.csv",
    "dialect": { "delimiter": "," },
    "schema": {
      "fields": [
        { "name": "USUBJID", "type": "string" },
        { "name": "AETERM", "type": "string" },
        { "name": "AESTDT", "type": "date"    },
        { "name": "AESTDY", "type": "integer" }  ]
    }
  } ]
}
```

It should be easy to deduce from the above *datapackage.json* file that, for example, *ADSL.csv* contains a variable *USUBJID* of type *string*, and that ADAE contains a variable *AESTDY* of type *integer* (both highlighted above in red).

When *ADSL.csv* and *ADAE.csv* are uploaded to Mira with the above *datapackage.json* file, Mira uses the metadata in the *datapackage.json* file to import the data into database tables with columns having the relevant data types.

API ENDPOINTS

Once CSV files have been uploaded to Mira with their corresponding *datapackage.json* file, the data is imported and a HTTP API becomes available. Using the earlier example, where two CSV files, ADSL.csv and ADAE.csv are uploaded, Mira then assigns an ID number, e.g. 123, which becomes part of a root API URL. Assuming Mira is running on *localhost* at port 3000, this root URL would be:

http://localhost:3000/api/projects/123/

Note that Mira uses the term “projects” to describe groups of uploaded files, and appears in the URL as a REST-style resource [6]. Table 1 summarises the API endpoints Mira would provide in our example:

Table 1

API Endpoint URL (host and port not shown)	Description
../api/projects/123	Returns the project metadata
../api/projects/123/tables	Returns the data tables available for querying, returning one JSON object per uploaded table
../api/projects/123/tables/ADSL	Returns ADSL table metadata
../api/projects/123/tables/ADAE	Returns ADAE table metadata
../api/projects/123/tables/ADSL/columns	Returns ADSL column metadata
../api/projects/123/tables/ADAE/columns	Returns ADAE column metadata
../api/projects/123/tables/ADSL/data	Returns the first page of the ADSL data. Query strings can be appended to filter the data.
../api/projects/123/tables/ADAE/data	Returns the first page of the ADAE data. Query strings can be appended to filter the data.

All of the above API requests are made using the HTTP GET method and all responses are in JSON. These API endpoints can be explored using a web browser's address bar. For example, to retrieve ADSL data, the API request URL would be:

```
../api/projects/123/tables/ADSL/data
```

And the JSON response might look like:

```
{ data: [{
              id: 1,
              usubjid: "ABC001-101-01",
              sex: "F",
              arm: "Treatment B",
              age: 41 },
         {    id: 2,
              usubjid: "ABC001-101-02",
              sex: "F",
              arm: "Placebo",
              age: 60 }
       ]
}
```

QUERYING

Data can be filtered by appending a query string to the data API endpoints. Following on from our example data, to request all rows from ADAE where *AETERM* is “Headache” and the *AESTDY* is greater than 0, you would use the following URL:

```
../api/projects/123/tables/ADAE/data?AETERM_eq=Headache&AESTDY_gt=0
```

The query string begins with a “?” character which is then followed by “&”-separated field-value pairs. The field-value pairs are used like this to filter the data and the available query fields depend on the names and types of the columns in the uploaded CSV files. Table 2 outlines the query fields which can be used in the query string, where *[column]* represents a column in an uploaded CSV file :

Table 2

Field-value Pair	Description	Available for Column Types
[column]_eq=value	Return rows where [column] equals value	All column types
[column]_ne=value	Return rows where [column] does not equal value	All column types
[column]_contains=value	Return rows where [column] contains value	Text only
[column]_not_contains=value	Return rows where [column] does not contain value	Text only
[column]_begins=value	Return rows where [column] begins with value	Text only
[column]_not_begins=value	Return rows where [column] does not begin with value	Text only
[column]_ends=value	Return rows where [column] ends with value	Text only
[column]_not_ends=value	Return rows where [column] does not end with value	Text only
[column]_lt=value	Return rows where [column] is less than value	Numeric, dates and times
[column]_le=value	Return rows where [column] is less than or equal to value	Numeric, dates and times
[column]_gt=value	Return rows where [column] is greater than value	Numeric, dates and times
[column]_ge=value	Return rows where [column] is greater than or equal to value	Numeric, dates and times
[column]_blank	Return rows where [column] is blank / missing	All column types
[column]_not_blank	Return rows where [column] is not blank / missing	All column types

ORDERING

Ordering is applied to a query by appending one or more *order=[column]* field-value pairs to the query string, each pair delimited by a comma character. The following would request rows from ADAE ordered by USUBJID and AESTDY:

```
../api/projects/123/tables/ADAE/data?order=USUBJID,AESTDY
```

Ascending order is applied by default. The following request would return rows from ADAE ordered by descending USUBJID:

```
../api/projects/123/tables/ADAE/data?order=USUBJID:desc
```

PAGING

The API returns a default number of rows per page (25 at the time of writing). This is adjusted using the *per_page* query field, and a particular page is requested using the *page* query field. The following would request 50 rows per page from ADAE, and the 4th such page:

```
../api/projects/123/tables/ADAE/data?per_page=50&page=4
```

The maximum allowable *per_page* value is a setting applied by whoever administers the Mira application.

The API's HTTP response includes *Records-Per-Page and Records-Total* headers and paging links are included in the *Link* header [7]. These provide what is needed to implement pagination on the client side.

DATATABLES JAVASCRIPT LIBRARY

DataTables [8] is a plug-in for the jQuery JavaScript library which adds interaction controls to HTML tables, such as paging, filtering and sorting. The APIs made available by Mira are capable of responding to requests from the DataTables JavaScript library. This provides a convenient way to present the uploaded data in the web browser.

API USAGE

Having uploaded CSV files and with a functioning API, various JavaScript libraries can be used to present the data in tables, graphs, maps, dashboards etc. For example, jQuery can be used to read the data, d3.js or Google Charts to create charts and DataTables to create interactive HTML tables. Essentially, Mira provides a simple API to your data which can be used to build other applications. For example, study dashboards and interactive patient profiles, tables and figures could be developed.

OPEN SOURCE

Mira has been developed with open-source software. It uses the *Ruby on Rails™* web framework with a *PostgreSQL™* database. Its source code repository is available on-line [9] and has been open-sourced under the MIT license [10]. The repository's README file contains information on how to set-up an instance of Mira. There is plenty of room for improvement within the application's code-base, so collaborations are welcome.

CONCLUSION

Web APIs are everywhere and there's a wide variety of tools to work with them. Mira provides a simple web API to tabular data, facilitating the use of these tools to create dynamic data presentations and other applications.

REFERENCES

[1] https://open.fda.gov/
[2] https://en.wikipedia.org/wiki/Application_programming_interface#Web_APIs (accessed 19-Aug-2015)
[3] http://rubyonrails.org/
[4] http://data.okfn.org/standards
[5] http://dataprotocols.org/tabular-data-package/
[6] https://en.wikipedia.org/wiki/Representational_state_transfer
[7] http://www.w3.org/wiki/LinkHeader
[8] http://datatables.net/
[9] https://github.com/davbre/mira
[10] http://opensource.org/licenses/MIT

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

david.n.brennan gmail