TechClass

Open UAS

Course Project:

API Design for a service provider

Independent Test Organization REST API Design Instances

Submitted by:
Arnel Imperial

Instructor:
Farhad Eftekhari

A project submitted to TechClass and credentialing to Metropolia Open UAS

In partial fulfillment of the Requirements for the Subject

Mastering the Fundamentals of RESTful API Design

Date: March 30, 2019

Project links:

- ***REST-API app (BaseURI): https://dataindustries.herokuapp.com/api***
- ***REST-API app (Documentation):*** https://dataindustries.herokuapp.com
- ***Repository: https://github.com/arnelimperial/RESTful-API-Design***

Table of Contents

I - API Base URL

The API design was simulated in an independent test type of organization named ***Data Industries*** as service provider. It has a baseurl: ***https://dataindustries.herokuapp.com/api***

The organization provides independent testing and verification of environmental matrices such as water, wastewater, soil, air, hazardous waste, agricultural produced as well as raw materials and commodities for compliance and certification.

II – API Consumers

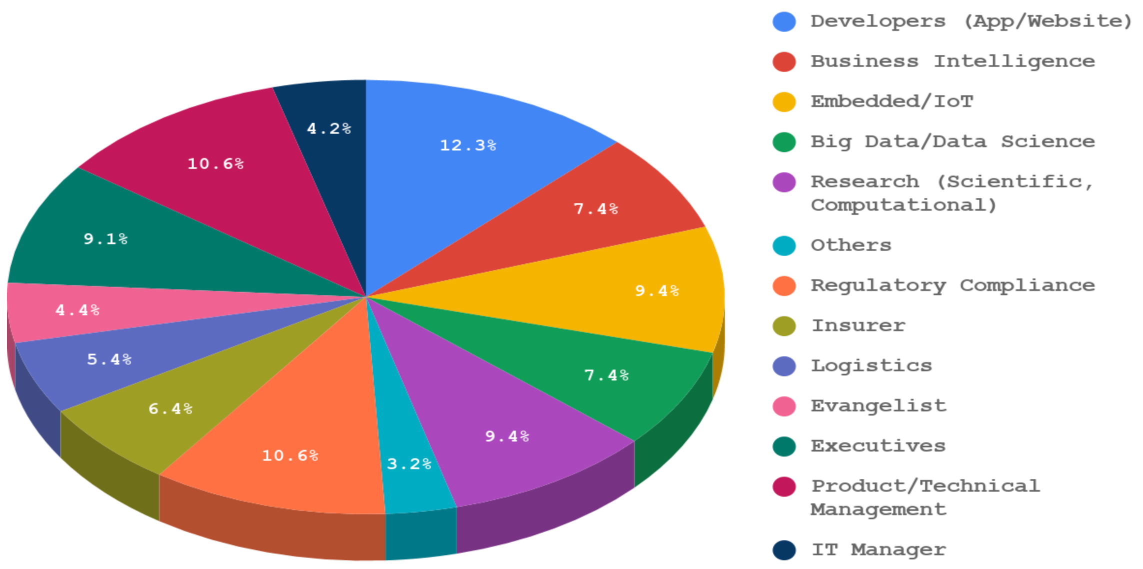


Fig.1: Demography of Data Industries API consumers

A consumer is a way to register as an interested party for a container. As a third-party testing firm, it is deemed to develop a collection of data from the results of services and an agreement to its clients to publish the results as open data. The use of REST API can leverage the dissemination of data for public use on certain resources.

Those people are categorized as Tech- savvy individuals or groups working in an organization. They are usually developers (applications, software, websites etc.), product developers, decision-makers, data wranglers, entrepreneur, technical management, regulatory bodies, technology evangelist and researcher in the academe or industry.

Generally, they have different sets of skills and motives on how to exploit resources by using internet technology combined with their respective specialization in the company from which they work. Actually, as long if someone knows how to interact with API and knows how to use it, that individual can be included as a consumer. In this scenario, the API provider (Data Industries) is also the main consumer of its API and the rest are the external partnership, collaboration, development and value propositions from different organization and sectors of industry.

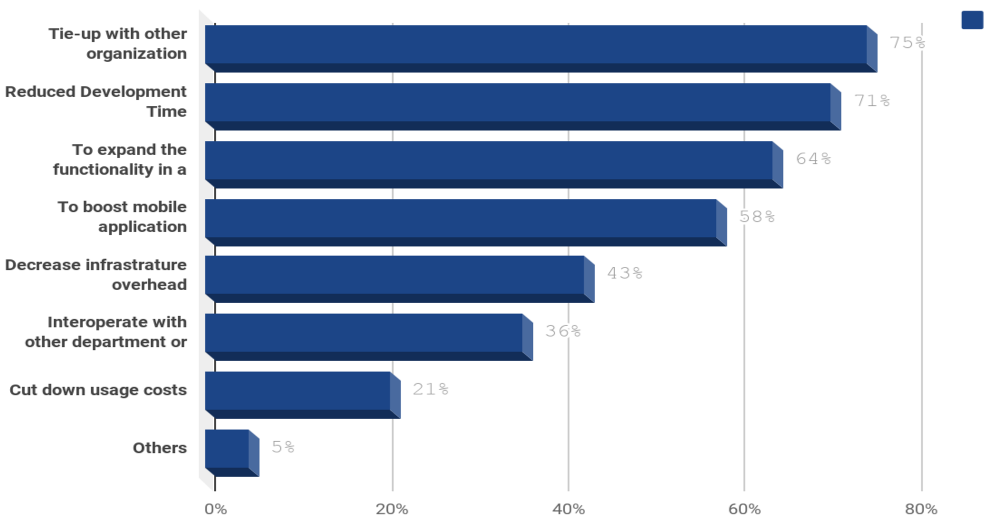


Fig.2: Rationality of using Data Industries API

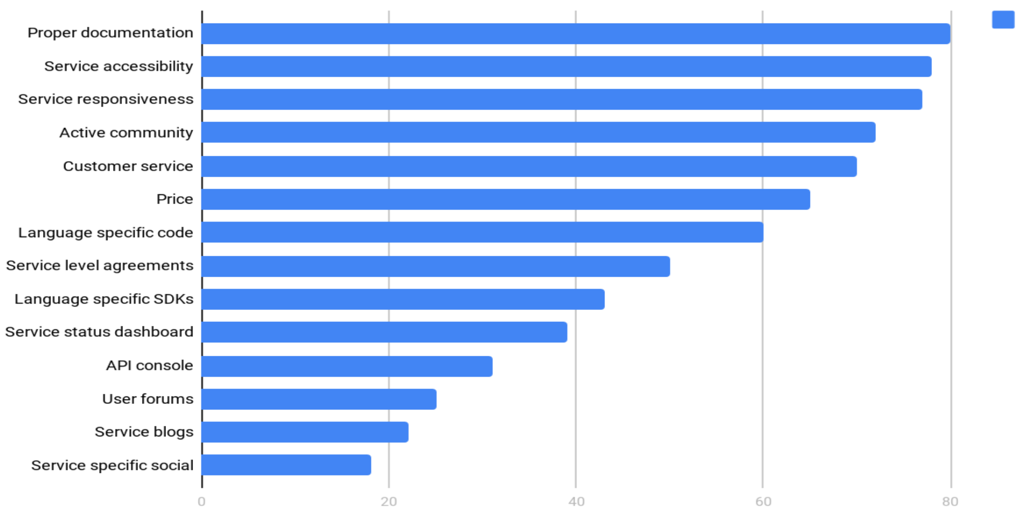


Fig.3: Factors to consider for customers experience

III - API Style

The deciding factor to consider REST to be implemented in the API project is to target first the goal (Table 2) of the management and at the same time make a comparison of different API style (Table 1) to determine which one is aligned to our needs.

Why REST for Data Industries?

Fig.4: Decision for using REST API

- *SOAP has been preferred for services within the enterprise and REST has been preferred for services that are exposed as public APIs. Data Industries wants its API and data resources to be public to be used not only for producing data but to transform data into a piece of useful information.*

- *Less duplication (HTTP already represents operations like DELETE, PUT, GET etc.*
- *More standardized - HTTP operations are well understood and operate consistently. Creating clients, developing APIs, the documentation is much easier to understand.*

- *REST permits many different data formats. JSON usually is a better fit for data and parses much faster. REST allows better support for browser clients due to its support for JSON.*
- *Exchanging data over the web that only deal with code, no extensive standards to follow, and no overhead.*

Table 1: Comparison of different API styles

Distinction	RPC	SOAP	REST
Style	Can use to request a service from a program located in	- Basically SOA based architecture that can used for	Designed for web based communication assuming only

	another computer in a network without having to understand network details.	middleware interoperability. - A distributed computing style implementation.	point-to-point communication. A web style (Client Server) implementation.
Data Format	Permits XML and JSON. Can allow to use HTML if Javascript or server code is involved.	Only XML	JSON, XML, HTML, RAML
Security	Encryption standards: Dieffie Hellman and DES.	Supports WS-Security and SSL.	HTTPS and SSL.
Functionality	- Masked remote function calls as local. - Request/reply paradigm for message passing. Client/Server model.	- Function driven: transferred structured data. - Service oriented: invoke service by RPC method.	- Data driven: access resource for data. - Resource oriented: uses URI and methods to expose resources.
Documentation	Requires extensive documentation.	- More complex documentation. - WSDL- Web Service Definitions.	- Easier to understand. - Supplemented by Hypermedia.
Bandwidth	Lightweight.	More resources required.	Lightweight: fewer resources.
Payload handling	Requires users to know the procedure names	- Requires strict communication contracts and needs to know everything before making transactions. - Exposes operation/methods call.	- Returning data without exposing methods. - Needs know knowledge of API.
Data Cache		Can not be cached.	Cachable.
Reliability	- Reliable data compressed. - Procedure calls preserved business logic which apt to the application. - Enable the application to run in local and distributed environment.	- Can ensure ACID transaction or Atomicity, Consistency, Isolation, Durability. - Guarantees that all database transaction are processed reliably. - Has Successful/retry logic built-in.	- Limited single HTTP transaction. - Cannot implement Two-Phase Commit (2PC). - REST expects clients to retry if something fails.
Generalization	- Optimized for specific task. - Fine grained control. - Fits proprietary mission-critical apps. - Costly planning and implementation. - It is not Object-oriented paradigm: invokes functions in servers as opposed to the methods on objects. - Distribution transparency problems.	- Highly specified and official standards must be followed. - For enterprise: high reliability and high security system.	- Low technical barrier to entry. - Massively scalable and extensible. - Performance penalty: hard to customize. - Fits for open resource-oriented apps. - Can be use to all systems apart from where high security and high reliability is critical. - Has no official standard for REST. - Easier planning and implementation

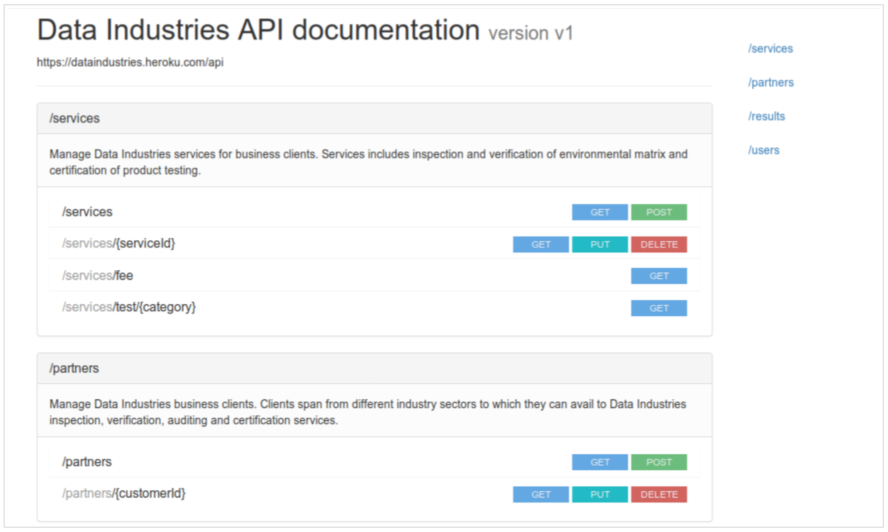
Table 2: Conformance of different API style to Data Industries project requirements

Requirements	RPC	SOAP	REST
Requirements for scalable and extensible application.	✓	✓	✓
Platform and language agnostic application.	X	X	✓
Easy to build and fast deployment.	✓	X	✓
Responsive UI.	✓	✓	✓
Not mission-critical level in terms of security requirements.	X	X	✓

IV - API Blue Print

- **Link to** API Documentation **and** RAML file.

RESTful API Modeling Language (RAML), a vendor-neutral, open-specification language built on YAML 1.2 and JSON for describing RESTful APIs was implemented to design the first version of the API. The **RAML 2 HTML** documentation generator https://github.com/raml2html/raml2html was use to generate the HTML file to be use in presenting the documentation in the browser.



Data Industries API documentation version v1

https://dataindustries.heroku.com/api

/services

Manage Data Industries services for business clients. Services includes inspection and verification of environmental matrix and certification of product testing.

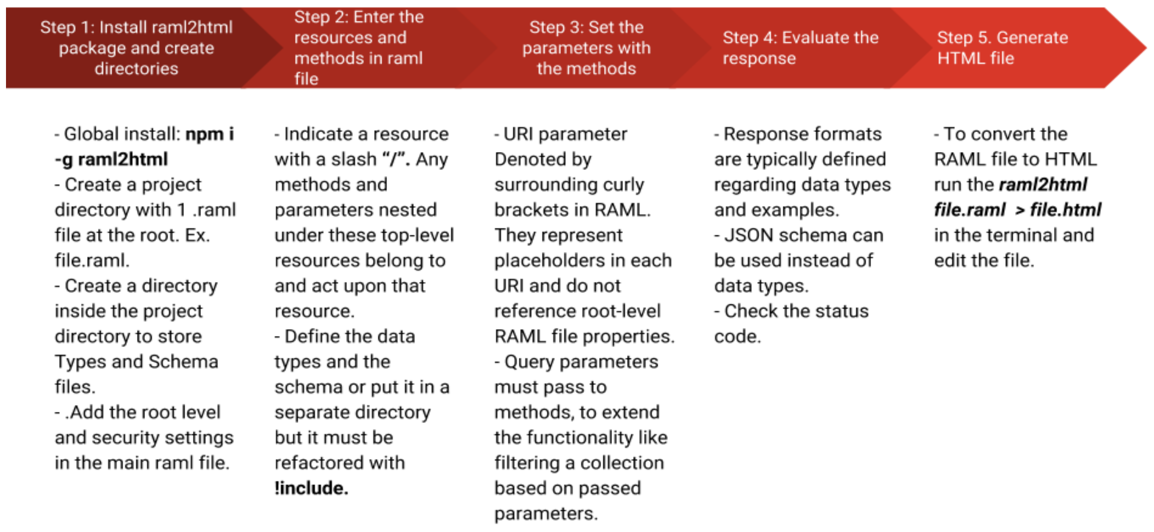
/services	GET POST
/services/{serviceId}	GET PUT DELETE
/services/fee	GET
/services/test/{category}	GET

/partners

Manage Data Industries business clients. Clients span from different industry sectors to which they can avail to Data Industries inspection, verification, auditing and certification services.

/partners	GET POST
/partners/{customerId}	GET PUT DELETE

Fig.5: Screenshot of Data Industries API Documentation designed in RAML



Step 1: Install raml2html package and create directories	Step 2: Enter the resources and methods in raml file	Step 3: Set the parameters with the methods	Step 4: Evaluate the response	Step 5. Generate HTML file
- Global install: **npm i -g raml2html** - Create a project directory with 1 .raml file at the root. Ex. file.raml. - Create a directory inside the project directory to store Types and Schema files. - .Add the root level and security settings in the main raml file.	- Indicate a resource with a slash **"/".** Any methods and parameters nested under these top-level resources belong to and act upon that resource. - Define the data types and the schema or put it in a separate directory but it must be refactored with **!include.**	- URI parameter Denoted by surrounding curly brackets in RAML. They represent placeholders in each URI and do not reference root-level RAML file properties. - Query parameters must pass to methods, to extend the functionality like filtering a collection based on passed parameters.	- Response formats are typically defined regarding data types and examples. - JSON schema can be used instead of data types. - Check the status code.	- To convert the RAML file to HTML run the ***raml2html file.raml > file.html*** in the terminal and edit the file.

Fig.6: Steps in designing Data Industries API with RAML

V - API Value Chain

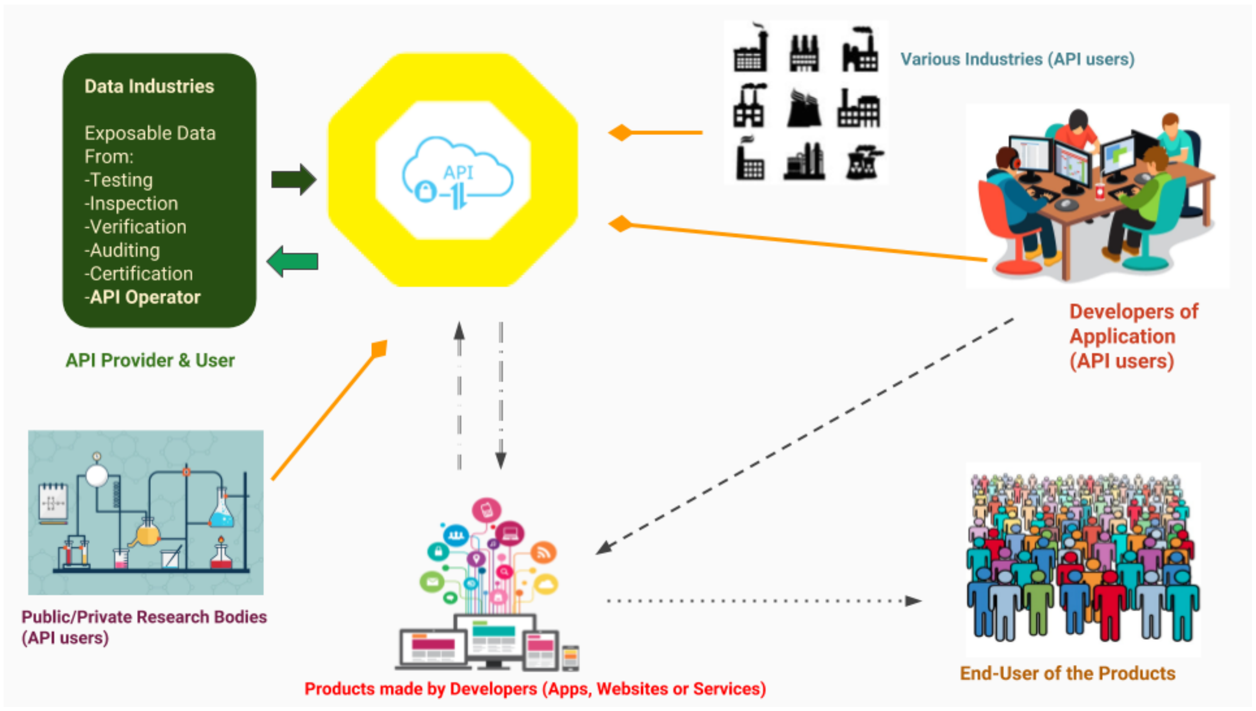

Data Industries
Exposable Data From:
-Testing
-Inspection
-Verification
-Auditing
-Certification
-API Operator
API Provider & User
API
Various Industries (API users)
Developers of Application (API users)
Public/Private Research Bodies (API users)
Products made by Developers (Apps, Websites or Services)
End-User of the Products

Fig. 7: Data Industries activity model to create services with the aid of API

VI. API Features

Table 3. Data Industries API Features and Specification

Features/Specification	Description
API Design and Documentation	API was designed and documented in RAML 1.0. HTML documentation generator using Raml2html.
Development Environment	It is coded in JavaScript and developed in NodeJS run time environment.
Database	NoSQL Database: MongoDB(local machine) and MLAB (MongoDB Hosting).
Server/Router framework	Restify frameworks as the main server of the application.
Object-Relational Mapping: Type casting, validation and query building	Mongoose for MongoDB Object modeling tooling.
Error handling	Set of constructors that can be used to new up Error objects with default status codes by Restify-error module.
Cross-Origin Resource Sharing Error handling	Handling CORS errors using Restify-cors-middleware.
Authorization	Implementation of JSON Web Token (JWT) by Jsonwebtoken package.
Validation of users input based on model schema	Object schema description language and validator for JavaScript objects using Joi.
Authentication	Validators of JSON Web Tokens using Restify-JWT-Community middleware.
Timestamp for created objects	The created At and updated At date attribute that gets auto-assigned to the most recent create/update timestamp using Mongoose-timestamp plugin.
Aggregation of different schema	Joi validation for Mongoose models provided by Joigoose module.
Searchability	The API can be search and deployed in the cloud with Heroku.
Sharing of other data format	It can serve static files. Description
Encryption	Encryption of user information (password etc.) handled by Bcryptjs library.
Revision system	Version control with Git

VII – Minimum Viable Product (MVP)

Developing a developer portal a website that provides Data Industries API with all the documentation connected to it.

Table 4: Proposal for developing MVP for a Developer Portal

Key Components	Description	Plan of Action
Integration	Shows what Data Industries API are about, how they work, how developers can start onboarding and where they can find resources.	Implement quick start or overview pages in details, illustrative use cases and partner showcases of real-life integration successes such as blog posts. Provides timely and appropriate links to Getting Started, API reference documentation, and other Docs.
Decision-Maker Documentation	Addressing the needs of developers and business people with information and guidance about the value of API in their organization through appropriate business description.	Include an API Explorer that allows the possible consumer to play around with API and get a first feeling before any registration. Include “Try-it-Out” or Sandbox functionality are some of the options
API Business Model	Pricing model innovations, new ways to monetize API with explanation, shareable value propositions,	Provides specifics on the opportunity from a monetary and technical perspective to developers who may still be questioning the value of building on Data Industries. Including a memorable video about value propositions of Data Industries
Active Developers Community	Community sections where developers can share knowledge and build connections.	Including community blogs, developers forum, upcoming events and social media connections.
Policies and Terms of Use	Specify the relation between customer and API supplier and building trust through a genuine approach through topics in policies and terms of use.	Make this data accessible and easy to navigate by including sidebar summaries.
Maintenance support	Creating by clearly and creatively indicating their availability and reliability through great release notes and other maintenance support.	Provides Support pages that address the current known problems of the consumers
Digital Experience/Design	Understandable and well-structured usability, aesthetics, contents and optimized experiences consistently across different digital touchpoint.	No cluttering of information and links with category. Having a simple user interface that serves multiple user audiences.

VIII – Other Version

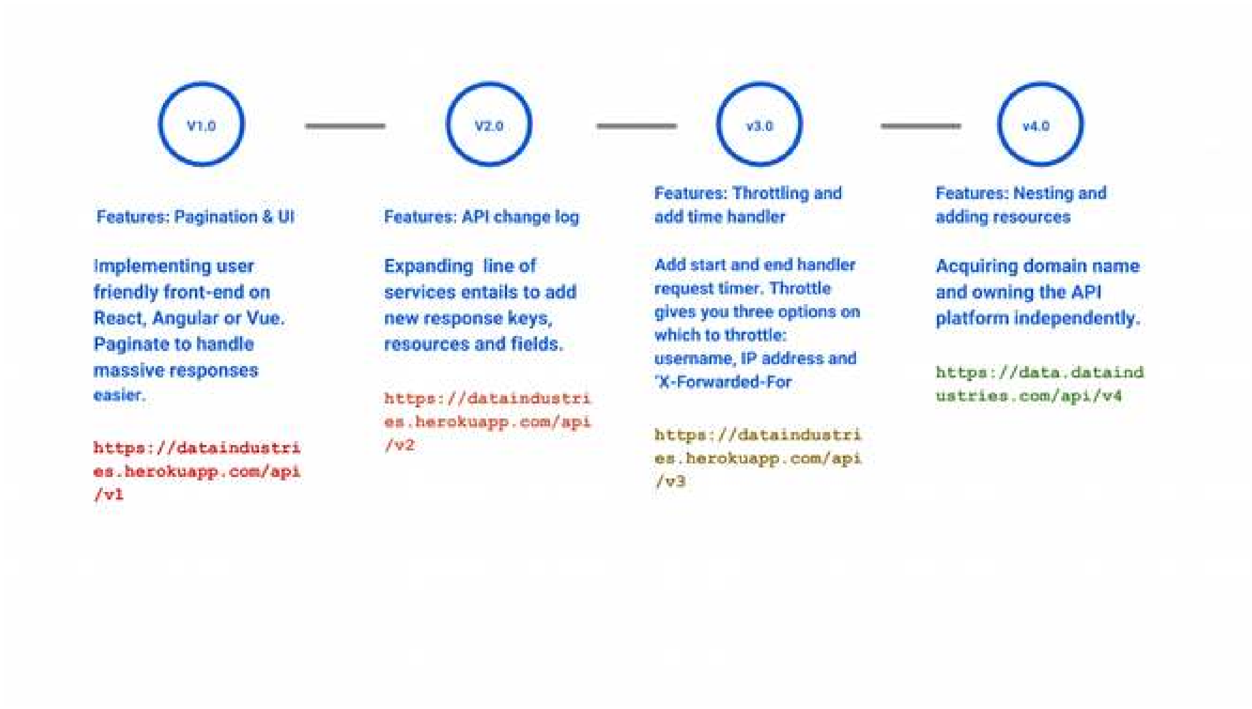

V1.0
V2.0
v3.0
v4.0
Features: Pagination & UI
Implementing user friendly front-end on React, Angular or Vue. Paginate to handle massive responses easier.
https://dataindustries.herokuapp.com/api/v1
Features: API change log
Expanding line of services entails to add new response keys, resources and fields.
https://dataindustries.herokuapp.com/api/v2
Features: Throttling and add time handler
Add start and end handler request timer. Throttle gives you three options on which to throttle: username, IP address and 'X-Forwarded-For
https://dataindustries.herokuapp.com/api/v3
Features: Nesting and adding resources
Acquiring domain name and owning the API platform independently.
https://data.dataindustries.com/api/v4

Fig.8: Upcoming API features with the corresponding versions

IX – Resources

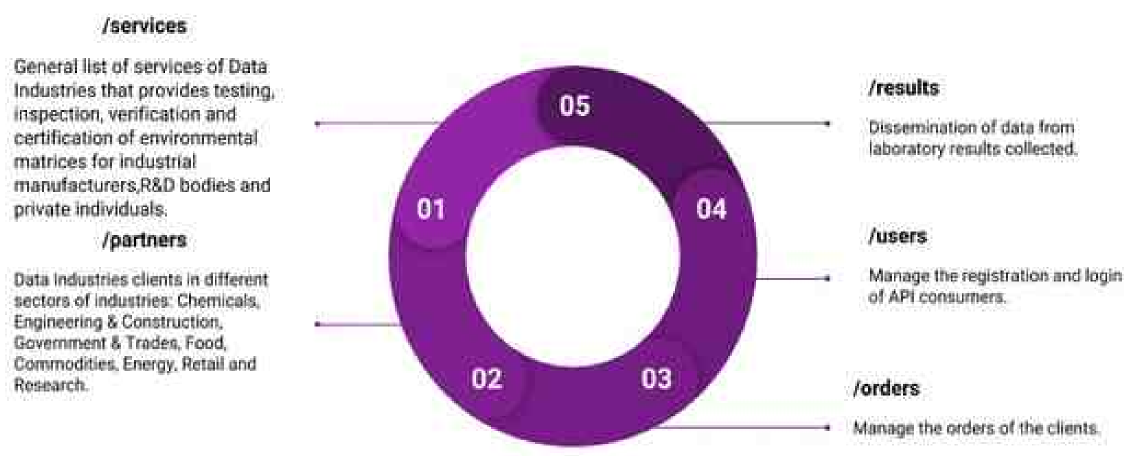

/services
General list of services of Data Industries that provides testing, inspection, verification and certification of environmental matrices for industrial manufacturers,R&D bodies and private individuals.
/partners
Data Industries clients in different sectors of industries: Chemicals, Engineering & Construction, Government & Trades, Food, Commodities, Energy, Retail and Research.
05
01
04
02
03
/results
Dissemination of data from laboratory results collected.
/users
Manage the registration and login of API consumers.
/orders
Manage the orders of the clients.

Fig.9: Data Industries API resources

X – URI, Endpoints, Parameters and HTTP Methods

Base URI: https://dataindustries.herokuapp.com/

Table 5: Data Industries API Definition: URI, Endpoint, Methods and Parameters

Resources	URL	Endpoint	HTTP Method	URI Param.	Query Param.	Results
/services	https://dataindustries.herokuapp.com/api/services	*/api/services*	GET			Get all services
/services	https://dataindustries.herokuapp.com/api/services	*/api/*services	POST			Add new service
/services	https://dataindustries.herokuapp.com/api/services	*/api/*services	PUT	*/services/*{id}		Update service by Id
/services	https://dataindustries.herokuapp.com/api/services	*/api/*services	DELETE	*/services/*{id}		Delete service by Id
/services	https://dataindustries.herokuapp.com/api/services	*/api/*services	GET		*/services/*fee?fee=123	Return all services w/ certain fee.
/services	https://dataindustries.herokuapp.com/api/services	*/api/*services	GET	*/services/*test/{category}		Return all services based on their category
/partners	https://dataindustries.herokuapp.com/api/partners	*/api/*partners	GET			Get all customers info.
/partners	https://dataindustries.herokuapp.com/api/partners	*/api/*partners	POST			Create customer info.
/partners	https://dataindustries.herokuapp.com/api/partners	*/api/*partners	GET	/api/partners/{id}		Return customer by Id
/partners	https://dataindustries.herokuapp.com/api/partners	*/api/*partners	PUT	*/api/*partners/{id}		Update customer by Id
/partners	https://dataindustries.herokuapp.com/api/partners	*/api/*partners	DELETE	*/api/*partners/{id}		Delete customer.
/orders	https://dataindustries.herokuapp.com/api/orders	*/api/*orders	GET			Return all orders info.

/orders	https://dataindustries.herokuapp.com/api/orders	/*api*/orders	POST			Make orders.
/orders	https://dataindustries.herokuapp.com/api/orders	*/api/orders*	GET	/api/orders/{id}		Return order info. by Id
/orders	https://dataindustries.herokuapp.com/api//orders	*api/orders*	PUT	*/api/orders/*{id}		Update order info by Id
/orders	https://dataindustries.herokuapp.com/api/orders	/*api*/orders	DELETE	*/api/orders/*{id}		Delete order.
/users	https://dataindustries.herokuapp.com/api/users	/*api*/users	GET			Return all users info.
/users	https://dataindustries.herokuapp.com/api/users	/*api*/users	POST			Create user account
/users	https://dataindustries.herokuapp.com/api/users	*/api/users*	GET	/api/users/{id}		Return user info. by Id
/users	https://dataindustries.herokuapp.com/api//users	*/api/users*	PUT	*/api/users/*{id}		Update user info by Id
/users	https://dataindustries.herokuapp.com/api/users	/*api*/users	DELETE	*/api/users/*{id}		Delete user
/results	https://dataindustries.herokuapp.com/api/results	/*api*/results	GET			Return all results info.
/results	https://dataindustries.herokuapp.com/api/results	/*api*/results	POST			Record the result
/results	https://dataindustries.herokuapp.com/api/results	*/api/results*	GET		*results/name?name=field*	Return result of analysis
/results	https://dataindustries.herokuapp.com/api/results	*/api/results*	PUT	*/api/results/*{id}		Update result info by Id
/results	https://dataindustries.herokuapp.com/api/results	/api/results	DELETE	*/api/results/*{id}		Delete a record.

XI – Validation

The validation of object constraints from the given resources was established with the implementation of NodeJS packages:

- Mongoose: Validation of schema for consistency when inserting/updating/finding documents from collections.
- Joi: Validation of data types at the request level.
- Joigoose: To run the Mongoose and Joi concurrently.
- Restify-errors: Module for exposing HTTP status code.

Example of validation is illustrated using **/services** resource for this purpose. The details of validation can be reviewed in the API Documentation.

Fig.10: HTTP method validation for GET /*api*/servises

API Definition: GET /api/services
Description: Return all services.
Response:
Data Types:
- testName: string
- testMethod: string
- fieldOfTesting: string
- sampleType: string
- testFee: number
Body:
Media type: application/Json
Success status: 200 – OK
Error status: 404 – Not Found

Fig.11: HTTP method validation for POST /*api*/services

API Definition: POST /api/services
Description: Create new service.
Request Body:
Media Type: Application/Json
Properties:
testName: required(string)
testMethod: required(string)
fieldOfTesting: required(string)
sampleType: required(string)
testFee: required(number)
Ex:

```
{
    "testName": "Arsenic",
    "testMethod": "SM 200.7",
    "fieldOfTesting": "Environmental Toxicology",
    "sampleType": "Wastewater",
    "testFee": 94.44
}

Response:
Body: Application/Json
Success status: 201 – Create Success
Error status: 500 – Internal Server Error
Ex:
{
    "code": "Internal",
    "message": "Service validation failed: testName: Path `testName` is required.,
testMethod: Path `testMethod` is required., fieldOfTesting: Path `fieldOfTesting` is
required., sampleType: Path `sampleType` is required., testFee: Path `testFee` is required."
}
```

Fig.12: HTTP method validation for PUT /*api*/services/{serviceId)

API Definition: PUT /api/*services/*{serviceId}
Description: Update a service by Id.
URI Parameters
serviceId: *required(string)*
Request Body:
Media Type: Application/Json
Properties
testName: *required(string)*
testMethod: *required(string)*
fieldOfTesting: *required(string)*
sampleType: *required(string)*
testFee: *required(integer)*
Ex:

```
{
  "fieldOfTesting": "Microbiology",
  "testFee": 100.55
}
```

Response:
Body: Application/Json
Success Status: 200 – OK
Error Status: 404 – Not Found
Ex:

```
{
    "code": "ResourceNotFound",
    "message": "There is no service with the id of 5c8fdabb8fa9da33c35009"
}
```

Fig.13: HTTP method validation for PUT /*api*/service/{serviceId}

API Definition: DELETE *a/pi/*service/{serviceId}
Description: Delete service by Id.
URI Parameters:
serviceId: *required(string)*
Response:
Media Type: Application/Json
Success Status: 204 – No Content
Error Status: 404 – Not Found

Table 6: HTTP method properties of Data Industries REST API

Resources	URL	Endpoint	HTTP Method	URI Param.	Query Param.	Property
/services	https://dataindustries.herokuapp.com/api/services	*/api/services*	GET			**Idempotent/ Safe**
/services	https://dataindustries.herokuapp.com/api/services	*/api/*services	POST			**Unsafe**
/services	https://dataindustries.herokuapp.com/api/services	*/api/*services	PUT	*/services/*{id}		**Idempotent/ Unsafe**
/services	https://dataindustries.herokuapp.com/api/services	*/api*services	DELETE	*/services/*{id}		**Idempotent/ Unsafe**
/services	https://dataindustries.herokuapp.com/api/services	*/api/*services	GET		*/services/* fee? fee=123	**Idempotent/ Safe**
/services	https://dataindustries.herokuapp.com/api/services	*/api/*services	GET	*/services/*test/ {category}		**Idempotent/ Safe**
/partners	https://dataindustries.herokuapp.com/api/partners	*/api/*partners	GET			**Idempotent/ Safe**
/partners	https://dataindustries.herokuapp.com/api/partners	*/api/*partners	POST			**Unsafe**
/partners	https://dataindustries.herokuapp.com/api/partners	*/api/*partners	GET	/api/partners/{id}		**Idempotent/ Safe**
/partners	https://dataindustries.herokuapp.com/api/partners	*/api/*partners	PUT	/*api*/partners/{id}		**Idempotent/ Unsafe**
/partners	https://dataindustries.herokuapp.com/api/partners	*/api/*partners	DELETE	/*api*/partners/{id}		**Idempotent/ Unsafe**
/orders	https://dataindustries.herokuapp.com/api/orders	*/api/*orders	GET			**Idempotent/ Safe**
/orders	https://dataindustries.herokuapp.com/api/orders	*api/*orders	POST			**Unsafe**
/orders	https://dataindustries.herokuapp.com/api/orders	*api/orders*	GET	/api/orders/{id}		**Idempotent/ Safe**
/orders	https://dataindustries.herokuapp.com/api//orders	*api/orders*	PUT	*/api/orders/*{id}		**Idempotent/ Unsafe**
/orders	https://dataindustries.herokuapp.com/api/orders	*/api/*orders	DELETE	*/api/orders/*{id}		**Idempotent/ Unsafe**
/users	https://dataindustries.herokuapp.com/api/users	*/api/*users	GET			**Idempotent/ Safe**
/users	https://dataindustries.herokuapp.com/api/users	*api*users	POST			**Unsafe**
/users	https://dataindustries.herokuapp.com/api/users	*/api/users*	GET	/api/users/{id}		**Safe/ Idempotent**

/users	https://dataindustries.herokuapp.com/api//users	*/api/users*	PUT	*/api/users/*{id}		**Unsafe/ Idempotent**
/users	https://dataindustries.herokuapp.com/api/users	*/api/*users	DELETE	*/api/users/*{id}		**Unsafe/ Idempotent**
/results	https://dataindustries.herokuapp.com/api/results	*/api/*results	GET			**Safe/ Idempotent**
/results	https://dataindustries.herokuapp.com/api/results	*/api/*results	POST			**Unsafe**
/results	https://dataindustries.herokuapp.com/api/results	*/api/results*	GET		*results/ name? name=fie ld*	**Safe/ Idempotent**
/results	https://dataindustries.herokuapp.com/api/results	*/api/results*	PUT	*/api/results/*{id}		**Unsafe/ Idempotent**
/results	https://dataindustries.herokuapp.com/api/results	/api/results	DELETE	*/api/results/*{id}		**Unsafe/ Idempotent**

XII – Security Concerns

Table 7: Security Features of Data Industries API

Security Techniques	**Project application**	**Status of Implementation**
Authorization *Authorization is a mechanics of confirming that you have access to something. Gaining access to a resource because the permissions configured on it allow you access is authorized.*	JSON Web Token was use to create access token to access data from the resources and it is implemented with jsonwebtoken nodejs package.	The user must first create a POST request in https://dataindustries.herokuapp.com/api/register: email, password, name, company name and occupation are the required field to save the credentials in the database. Encryption of password was implemented using Bcryptjs package.
Authentication *Authentication is the technique of checking who you are by means of your credentials and identification. Signing in to a PC with a user name and password or logging to a Unix server using ssh client, or access the server using POP3 and SMTP email client are forms of authentication.*	The of JSON Web token was validated using Restify-JWT-Community.	The user must login to https://dataindustries.herokuapp.com/api/auth and send a POST request with your registered email and password to retrieve the token.
Delegation *Delegation allows an existing website to handle developer sign-in/sign-up and subscription of products instead of using the built-in functionality in the developer portal.*	Not yet implemented.	Not yet intact in the application.

XIII – Versioning Format

URI versioning format:
BaseURI: https://dataindustries.herokuapp.com/api/v1
Version: v1.0

Table 8: Data Industries API version 1.0 definition

Resources	**URL**	**Endpoint**	**HTTP Method**	**URI Param.**	**Query Param.**	**Results**
/services	https://dataindustries.herokuapp.com/api/v1/services	*/api/v1/services*	GET			Get all services
/services	https://dataindustries.herokuapp.com/api/v1/services	/api/v1/services	POST			Add new service
/services	https://dataindustries.herokuapp.com/api/services	/api/v1/services	PUT	*/services/*{id}		Update service by Id
/services	https://dataindustries.herokuapp.com/api/v1/services	/api/v1/services	DELETE	*/services/*{id}		Delete service by Id
/services	https://dataindustries.herokuapp.com/api/v1/services	/api/v1/services	GET		*/services/*fee?fee=123	Return all services w/ certain fee.
/partners	https://dataindustries.herokuapp.com/api/v1/partners	*/api/v1/*partners	GET			Get all customers info.
/partners	https://dataindustries.herokuapp.com/api/v1/partners	*/api/v1/*partners	POST			Create customer info.
/partners	https://dataindustries.herokuapp.com/api/v1/partners	*/api/v1/*partners	GET	/api/partners/{id}		Return customer by Id
/partners	https://dataindustries.herokuapp.com/api/v1/partners	*/api/v1/*partners	PUT	*/api/*partners/{id}		Update customer by Id
/partners	https://dataindustries.herokuapp.com/api/v1/partners	*/api/v1/*partners	DELETE	*/api/*partners/{id}		Delete customer.
/orders	https://dataindustries.herokuapp.com/api/v1/orders	*/api/v1/*orders	GET			Return all orders info.

/orders	https://dataindustries.herokuapp.com/api/v1/orders	*/api/v1/*orders	POST			Make orders.
/orders	https://dataindustries.herokuapp.com/api/v1/orders	*/api/v1/*orders	GET	/api/orders/{id}		Return order info. by Id
/orders	https://dataindustries.herokuapp.com/api/v1/orders	*/api/v1/*orders	PUT	*/api/orders/*{id}		Update order info by Id
/orders	https://dataindustries.herokuapp.com/api/v1/orders	*/api/v1/*orders	DELETE	*/api/orders/*{id}		Delete order.
/users	https://dataindustries.herokuapp.com/api/v1/users	*/api/v1/*users	GET			Return all users info.
/users	https://dataindustries.herokuapp.com/api/v1/users	*/api/v1/*users	POST			Create user account
/users	https://dataindustries.herokuapp.com/api/v1/users	*/api/v1/*users	GET	/api/users/{id}		Return user info. by Id
/users	https://dataindustries.herokuapp.com/api/v1/users	*/api/v1/*users	PUT	*/api/users/*{id}		Update user info by Id
/users	https://dataindustries.herokuapp.com/api/v1/users	*/api/v1/*users	DELETE	*/api/users/*{id}		Delete user
/results	https://dataindustries.herokuapp.com/api/v1/results	*/api/v1/*results	GET			Return all results info.
/results	https://dataindustries.herokuapp.com/api/v1/results	*/api/v1/*results	POST			Record the result
/results	https://dataindustries.herokuapp.com/api/v1/results	*/api/v1/*results	GET		*results/name?name=field*	Return result of analysis
/results	https://dataindustries.herokuapp.com/api/v1/results	*/api/v1/*results	PUT	*/api/results/*{id}		Update result info by Id
/results	https://dataindustries.herokuapp.com/api/v1/results	*/api/v1/*results	DELETE	*/api/results/*{id}		Delete a record.