RESTful APIs:

A **RESTful API** is an architectural style for an application program interface (**API**) that uses HTTP requests to access and use data. That data can be used to GET, PUT, POST and DELETE data types, which refers to the reading, updating, creating and deleting of operations concerning resources.

A **REST API** is a way for two computer systems to communicate over HTTP in a similar way to web browsers and servers.

RESTFul API Specification:

RAML:

RESTful API Modeling Language is a YAML-based language for describing RESTful APIs. It provides all the information necessary to describe RESTful or practically RESTful APIs. Although designed with RESTful APIs in mind, RAML is capable of describing APIs that do not obey all constraints of REST.

Examples-

* POST/api/v1/foos
* PUT/api/foos/{id}
* DELETE/api/v1/foos/{id}
* GET/api/v1/foos?name={name}&ownerName={ownerName}

OAS:

The OpenAPI Specification, originally known as the Swagger Specification, is a specification for machine-readable interface files for describing, producing, consuming, and visualizing RESTful web services. API specifications can be written in YAML or JSON. The format is easy to learn and readable to both humans and machines.

Non RESTFul APIs:

These are API’s which don’t follow all of the REST architecture constraints

* **RPC**: Remote Procedure Call (RPC) RPC is the earliest, simplest form of API interaction. It is about executing a block of code on another server, and when implemented in HTTP or AMQP it can become a Web API.Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer on a network without having to understand the network's details.A procedure call is also sometimes known as a function call or a subroutine call.
* **SOAP:** SOAP is a messaging protocol specification for exchanging structured information in the implementation of web services in computer networks. SOAP APIs is largely based on HTTP and XML. It is written in Web Service Description Language (WSDL)
* **GraphQL APIs:** GraphQL is a query language and server-side runtime for application programming interfaces (APIs) that prioritizes giving clients exactly the data they request and no more. ... As an alternative to REST, GraphQL lets developers construct requests that pull data from multiple data sources in a single API call
* **Async API:** Asynchronous APIs are application programming interfaces that return data for requests either immediately or at a later time, respectively. Asynchronous requests are useful in maintaining functionality in an application rather than tie up application resources waiting on a request. Protocols supported by AsyncAPI: Advanced Message Queuing Protocol (AMQP), Message Queuing Telemetry Transport (MQTT), WebSocket API, Kafka, JMS, STOMP

Richardson Maturity Model: Richardson used three factors to decide the maturity of a service i.e. [URI](https://restfulapi.net/resource-naming/), [HTTP Methods](https://restfulapi.net/http-methods/) and [HATEOAS](https://restfulapi.net/hateoas/) (Hypermedia). The more a service employs these technologies – more mature it shall be considered.

In this analysis, Richardson described these maturity levels as below:

[**Level Zero**](https://restfulapi.net/richardson-maturity-model/#level-zero): Level zero of maturity does not make use of any of URI, HTTP Methods. These services have a single URI and use a single HTTP method (typically POST). For example, most Web Services (WS-\*)-based services use a single URI to identify an endpoint, and HTTP POST to transfer SOAP-based payloads, effectively ignoring the rest of the HTTP verbs.

[**Level** **One**](https://restfulapi.net/richardson-maturity-model/#level-one): Level one of maturity makes use of URIs out of URI, HTTP Methods. These services employ many URIs but only a single HTTP verb – generally HTTP POST. They give each resource in their universe a URI. A unique URI separately identifies one unique resource – and that makes them better than level zero.

[**Level Two**](https://restfulapi.net/richardson-maturity-model/#level-two)**:** Level two of maturity makes use of URIs and HTTP out of URI, HTTP Methods. Level two services host numerous URI-addressable resources. Such services support several of the HTTP verbs on each exposed resource – Create, Read, Update and Delete (CRUD) services. Here the state of resources, typically representing business entities, can be manipulated over the network. Here service designer expects people to put some effort into mastering the APIs – generally by reading the supplied documentation. Level 2 is the excellent use-case of REST principles, which advocate using different verbs based on the HTTP request methods, and the system can have multiple resources.

[**Level Three**](https://restfulapi.net/richardson-maturity-model/#level-three)**:** Level three of maturity makes use of all three, i.e. URIs and HTTP.This level is the most mature level of Richardson’s model, which encourages easy discoverability. This level makes it easy for the responses to be self-explanatory. The service leads consumers through a trail of resources, causing application state transitions as a result.

Anypoint Design Centre: Anypoint Design Center™ gives you the tools you need to build connectors, implement data and application flows, and dramatically simplify API design, reuse, and testing.

Anypoint Design Center is a development environment that consists of two tools:

* [API Designer](https://docs.mulesoft.com/design-center/design-create-publish-api-specs)

In API Designer, we can create API specifications in RESTful API Modeling Language (RAML) 0.8 or 1.0, or according to OpenAPI specification (OAS) 2.0 or 3.0. We can also create API fragments in RAML. After you create a specification or fragment, you can publish it to Anypoint Exchange, so that it can be used by anyone in your MuleSoft organization.

* [Flow Designer](https://docs.mulesoft.com/design-center/about-designing-a-mule-application)

In Flow Designer, you create Mule applications to integrate systems into workflows.

API Designer: API Designer provides a web-based interface for designing, documenting, testing, and sharing API specifications and fragments.

**Designing:**

We can use either the text editor or the visual editor to create API specifications in RESTful API Modeling Language (RAML) version 0.8 or 1.0, or according to OpenAPI specification (OAS) version 2.0 or 3.0.

With the text editor, you can:

* Design according to best practices with a context-aware shelf that auto-populates with the appropriate methods, resources, parameters, and security components.
* Include RAML API fragments, such as security schemas and data types, from Exchange or write your own.
* Import existing API specifications.

With the visual editor, we can:

* Create a specification for use cases in which you do not need to include RAML fragments.
* Scaffold a specification that you can complete and customize in the text editor.

**Documenting:**

As we add descriptions in your specifications and fragments, you can view them in auto-generated, interactive documentation in the right-hand pane of the text editor.

**Testing**

We can use the mocking service to:

* Preview an API’s functionality before writing a single line of code.
* Send requests to the specification and receive defined responses.
* Use behavioral headers to simulate different scenarios, such as timeouts and errors.

**Sharing**

W can publish an API specification to Anypoint Exchange for use in API Manager, Anypoint Studio, or Flow Designer. You can also add an API specification directly to API Manager.

A specification writer uses API Designer to design API specifications and RAML fragments. RAML specifications can use assets that are hosted in Exchange. The specification writer can publish iterations and final versions to Anypoint Exchange.

From Exchange, the specification can be imported into Anypoint Studio and Flow Designer as a RESTful connector, or into API Manager. Anypoint Studio can even import a specification directly from API Designer.

## Visual Editor: We can use API Designer’s visual API editor to scaffold API specifications in RAML 1.0 or in OAS 2.0 or 3.0 (JSON). We do so by populating language-neutral forms.

## Procedure

1. On the **Projects** page in Design Center, click **Create** and then click **Create API Specification**.
2. In the **New API Specification** dialog, name your project. You can change the name later, if you want to.
3. Select the option **Visual editor** to use the visual API editor.

*Result:*

The visual API editor opens.

As you make progress, you can run the mocking service to simulate calls to your API specification. For more information, view the topic "Simulate Calls to an API", which is listed in the **See Also** section at the end of this topic.

If at any time you want to edit the specification directly or customize it (import RAML fragments and more), click **Edit RAML** or **Edit OAS** to switch to the API editor. After you switch, however, you cannot switch back to the visual API editor to continue creating the API specification.

If we are satisfied with our API specification, we can publish it to Exchange in RAML 1.0

**API Fragments:** An API fragment is a portion of an API specification, which is why understanding it starts at the API specification level. Instead of starting every project from scratch, you can reuse fragments and APIs to accelerate project delivery.

**Data Modelling:** It is the process of creating data models.  
It occurs at three levels

* Physical Model: It is a schema how data is stored in database
* Conceptual Model: Identifies the high level, user view of data
* Logical Model: It is between Physical and Conceptual Model and allows for a logical representation of data to be separated from physical storage.

**Canonical Data Model (CDM): Canonical data models** are a type of **data model** that aims to present **data** entities and relationships in the simplest possible form in order to integrate processes across various systems and databases

**Enterprise Data Model (EDM):** Enterprise Data Model is an integrated view of the data and related processes that consumes and store data across the organizations.

**Bounded Context:** is a central pattern in Domain-Driven Design. It is the focus of DDD's strategic design section which is all about dealing with large models and teams.It also acts as the conceptual foundation for the design of the software itself - how it's broken down into objects or functions. The **bounded context** concept originated in Domain-Driven Design (DDD) circles. It promotes an object-model-first approach to a service, defining a data model that a service is responsible for and is "bound to." In other words, the service owns this data and is responsible for its integrity and mutability.

* **Mirror a Backend System:** Creates an exact copy of the selected folders and files from the source.
* Unique in this is when we delete a file from the source, that file will eventually be deleted on the mirror backup
* First time we run a mirror backup it will take the longest time
* Increased data protection

Two ways of mirroring

* Disk : RAID -  ensure that the data is always available
* Server: the same information is quickly accessed at various sites