Designing an API

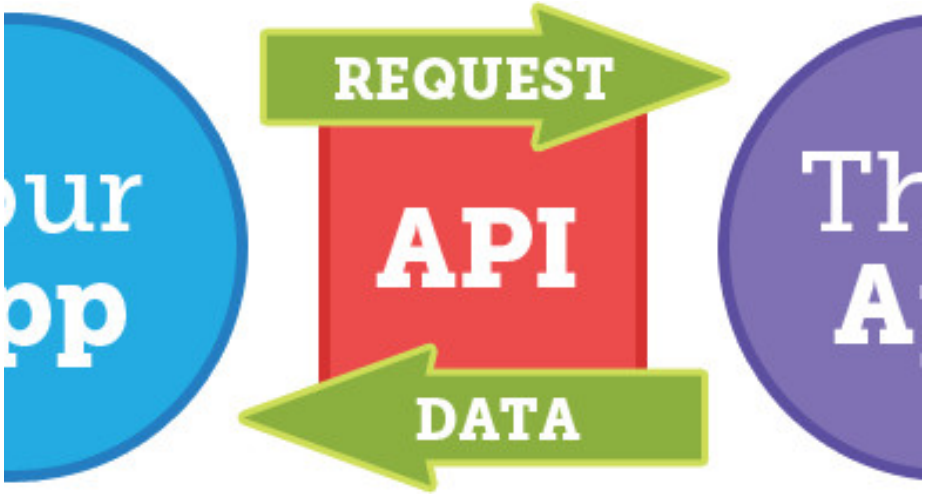
James Dey

Deytalytics Articles

-
- Cloud Architecture
- Data Architecture
- Enterprise Architecture
- Solution Architecture

Designing an API

Date Thu 24 November 2016 By Deytalytics Ltd Category Data Architecture. Tags data integration REST API



What is an API?

An API defines a controlled way in which you can interact with a service. It restricts the methods by which you can access a service, the parameters that

can be passed, and the messages that can be sent to a service as a request, and the message that is returned as a response.

Why do we need APIs?

Service orientated architecture splits systems and applications in to a logical set of services. So, for example, in order to access your facebook page, you need to be able to logon. What happens behind the scenes is that the username and password is passed to an authentication service via a defined API. Similarly, when you view your posts, the facebook web page has called a post retrieval service via an API, in order to retrieve the posts that you've made from a back end data store. The advantages of logically splitting systems and applications in to services include:-

1. It's easier to understand how the system works
2. It's easier to maintain the code, as you've split what would otherwise be a very large piece of code in to smaller parts
3. You can have developers work independently on a particular service.
4. You can re-write a particular service, or move the service to different hardware without affecting the system.
5. You can apply appropriate levels of security to different services

An API is effectively a contract. In the same way that when you're dealing with a human interaction, there are rules that define what information you need to provide and what you expect as a response. For example, if you make a cash payment in a store for a basket of items, you:-

1. Hand over the shopping items.
2. Hand over money for a particular value and currency
3. And wait for a receipt or verbal acknowledgement that the transaction is complete.

In order for payment services to take an online payment, it has to emulate that same transaction, so it requires identification of the particular shopping basket and the value and currency to be paid as part of the order completion service. The payment request service will request the customer to make payment. The customer then has to enter information about his payment method (e.g. debit card or paypal) and the payment authorisation service will respond to state whether the card payment worked properly (after calling a service to authorise payment from the cardholder's bank). Since this set of services has to work as smoothly as possible, an API will restrict the methods, parameters and messages which can be passed to the service, and the valid responses and messages that can be returned.

What are the steps in designing an API?

1. You will first need to understand the business service that you want to emulate. So in the example above, the business service is a shopping service.
2. From there, you need to consider how to split that business service in to a logical set of technical services e.g. add_shopping_item, finalise_shopping_basket, take_payment.
3. You will need a data architect to define the attributes associated to each object for which you wish to store information. For example, shopping item, shopping basket, payment method, payment card, currency, transaction date, payment instruction etc.
4. The data architect will also need to define the relationships between these data objects so that information that relates to each other can be retrieved later. For example, the payment instruction involves a payment card (f card is the payment method), a currency, a transaction date and a payment method. The payment instruction also relates to the shopping basket. The shopping basket contains many shopping basket items.
5. The data architect will also need to specify which attributes should have a restricted set of values. For example, there will only be restricted set of payment methods e.g. paypal, credit card, debit card and the list of currencies should be restricted to those defined by the International Standards Organisation (ISO).
6. The security architect will need to be consulted to ensure that legally sensitive data is not passed over the network in an unencrypted form, or stored in it's raw form in a data store or backup file.
7. The security architect will also need to approve the authentication method that is used to ensure that only authorised users can obtain access to certain data. For example, to gain access to your facebook posts, you first have to use the OAUTH authentication service to authenticate yourself via a username & password. You are then given a unique access code which needs to be supplied in subsequent API calls in order to retrieve sensitive data. (You may have noticed that it is common nowadays to register for a web site using your social media login. Invariably, you are also asked as to whether you agree to share certain private data with the web site owner. Since you've authenticated and accepted that the website should be authorised, Facebook will update it's internal authorisation tables to allow the website access to the data that you agreed to share).
8. The API designer will need to consider the methods by which information can be sent to the service and the parameters (fields and acceptable values) that need to be supplied.
9. The API designer will need to also define the response messages and any

status codes to help the calling application understand whether the service call was successful or not.

10. The API will need to be documented in order that there's a clear understanding to 3rd party application developers as to how the technical services interact with each other in order to complete the business service, and the what the methods, parameters and messages are that need to be passed to and received from each technical service, and how to respond to status codes.

What style should an API be written in?

The predominant style for writing modern web services is known as REST. It uses the http protocol methods (POST, PUT, DELETE & GET) to interact with a technical service via it's API. The advantage of this is that most programming languages already support the http style and status codes, so it's easier to write technical services.

The programmer can interact with any web-based service by identifying resources which are defined using the uniform resource identifier (URI). For example, this web page might have a URI of http://www.deytaltyics.com/designing-an-api.

The application programmer can retrieve information from the service via the API using the GET method

For example:-

```
GET http://api.deytalytics.com/retrieve_shopping_basket?basket_id=100001
```

would call the retrieve_shopping_basket service and request the contents of shopping basket 100001.

Typically, the API designer will respond with a http status code which tells the calling application whether the service call succeeded, failed, timed out etc. plus a response message in JSON or XML format.

In a similar fashion data can be stored via a service API call using the POST method e.g.

```
POST http://api.deytalytics.com/add_shopping_item?basket_id=100001 {item_id:1}
```

might add a particular product to a shopping basket.

What is the best way of specifying a complete set of REST service APIs?

A popular way of specifying REST service APIs is to use the REST API Modelling Language (RAML).

Using the modelling language has the benefits of:-

1. Ensuring that your design contains all of the necessary elements required for the API
2. That the API can be tested
3. The API is well documented
4. The API can be published to API repositories for discovery by 3rd party providers.

Comments

-
-
- Deytalytics.com
-
- Linkedin
-
- Cloud Architecture
- Data Architecture
- Enterprise Architecture
- Solution Architecture
-
- cloud
- github pages
- aws
- azure
- gcp
- website
- python
- cms
- fintech
- alexa
- openbanking

- data standards
- financial services
- business glossary
- iso20022
- data integration
- REST
- API
- semantic web
- rdf
- owl
- web 3.0
- data modelling
- data vault
- erwin
- powerdesigner
- uml
- relational
- e-r
- data architecture
- digital transformation
- business benefits
- enterprise architecture
- digital marketing
- strategy