Definition of an API

An application programming interface is a set of definitions and protocols for building and integrating application software. It can be thought of as a contract between an information provider and an informational consumer. The API is a mediator between the clients and the resources they wish to acquire from a server or database.

REST

REST stands for **Representational State Transfer**. It is a set of *architectural constraints* on the structure of an API rather than a fixed protocol. It is a particular way of implementing client-server interaction over HTTP.

When a request is made from a client to resources via RESTful API, the API transfers a representation of of the state of the resource to the requester or endpoint. The information is delivered via HTTP. The format can be of several types (HTML, XML, plaintext, Python, PHP etc) but is generally JSON because of its broad compatibility with multiple programming languages.

Key constraints

In order to qualify as RESTful, an API must meet the following constraints:

1. **Uniform interface**: Possess a client-server architecture with request manage through HTTP
2. **Client-server decoupling** : The client and server applications must be completely independent of one another. The *only* information the client should know about the server is the URI it uses to request the resource, it can't interact with the server in any other way. Likewise, the server shouldn't modify the client application in any way (contrast for example SSR) other than passing the requested data via HTTP.
3. **Statelessness** Server applications should not be able to store any data related to a client request. The request alone should contain all the information necessary for processing it, without recourse to any specifics of the client application. For example, a specification of POST with a certain JSON body and header authentication will be all that is provided to the server.
4. **Cacheability** Where possible, resources should be cacheable on the client or server side. Server responses must contain information about whether caching is allowed for the delivered resource (you see this in the headers in the DevTools console). The goal here is to improve performance on the client side whilst increasing scaleability on the server side.
5. **Layered system architecture** It may be the case that the data flow between the client and the server is not direct. For instance the request may be funneled through middleware or another program before it is received by the server. Similarly there may be several steps before the

client receives the requested data. Whilst one should not assume a direct correspondence, REST APIs need to be designed so that neither the client nor the server can tell whether it communicates with the end application or an intermediary.

Example

A basic example of a REST API would be a series of methods corresponding to the main HTTP request types.

| HTTP request type | URI | Action | Body ? | |————————- |————————— |————————— |————————— | | GET | /api/customers | Retrieve customers as array | No | | GET | /api/customers/guid | Get a specific customer | No, data comes from GUID | | PUT | /api/customers/guid | Update an existing customer | Yes | | DELETE | /api/customers/1 | Delete a customer | No, data comes from GUID | | POST | /api/customers | Create a new customer | Yes |