APIs CheatSheet

CODE HELP

In this Cheatsheet, we will cover the basics of API. We will provide examples to help you understand how API's work and how to use them in your own web development projects. Whether you are a beginner or an experienced developer, this PDF can serve as a useful reference guide.

An API, or Application Programming Interface, is a set of rules and protocols for building and interacting with software applications. It allows different software systems to communicate with each other, enabling them to share data and functionality. This allows developers to access the functionality of a certain software application or system without having to understand its underlying code or implementation.

An example of an API is the Facebook API, which allows developers to access and interact with the functionality of the Facebook platform, such as posting status updates, retrieving user information, and managing ad campaigns. Another example is the Google Maps API, which allows developers to embed maps and location-based functionality in their own websites and apps.

How an API Works:

APIs act as a bridge between applications and web servers, processing data transfer between systems. When a client application initiates an API call, also known as a request, it is sent to the web server via the API's Uniform Resource Identifier (URI) and includes a request verb, headers, and sometimes a request body. The API then processes the request and may make a call to an external program or web server for the requested information.

The server responds with the requested data, which the API then forwards to the initial requesting application. This process of requests and responses all happens through the API. Unlike user interfaces which are designed for human use, APIs are designed for use by computers or applications.

REST API: (Representational State Transfer)

REST (Representational State Transfer) is a type of web architecture and a set of constraints to be used when creating web services. RESTful API (Application Programming Interface) is an API that conforms to the REST architectural style and constraints, and it is typically used to make requests to retrieve or update data on a web server. A RESTful API uses HTTP requests to POST (create), PUT (update), GET (read), and DELETE (delete) data. A RESTful API also returns a response in a standard format, typically JSON or XML, and uses standard HTTP status codes to indicate the status of the request. RESTful APIs are popular because they are simple to understand and easy to use, and they work well with the HTTP protocol that the internet is built on. Additionally, RESTful APIs are often faster and more lightweight than their SOAP (Simple Object Access Protocol) counterparts because they use smaller message formats. RESTful API's have become a popular way for systems to expose databases through HTTP(S) following CRUD operations (Create, Read, Update, Delete), and return JSON or XML as responses, it's also widely used in microservices, mobile and web applications, IoT, and many more.

REST requires that a client make a request to the server in order to retrieve or modify data on the server.

A request generally consists:

- An HTTP verb, which defines what kind of operation to perform.
- A header, which allows the client to pass along information about the request.
- A path to a resource.
- An optional message body containing data.

CRUD: (Create Read Update Delete)

CRUD stands for Create, Read, Update, and Delete, which are the four basic operations that can be performed on data in a database. These operations are often used to interact with databases through APIs (Application Programming Interfaces).

- **Create:** This operation is used to add new data to the database. It is typically performed using the **HTTP POST** method and is used to create new resources.

- **Read:** This operation is used to retrieve data from the database. It is typically performed using the **HTTP GET** method and is used to read existing resources.

- **Update:** This operation is used to modify existing data in the database. It is typically performed using the **HTTP PUT** method and is used to update existing resources.

- **Delete:** This operation is used to remove data from the database. It is typically performed using the **HTTP DELETE** method and is used to delete resources.

CRUD operations are typically implemented in RESTful APIs, but they can also be used in other types of APIs. These operations can be used to expose the data in a database to other systems, such as a mobile app or a web application, that can use the data to provide a service to end-users.

HTTP Verbs

In the context of API (Application Programming Interface), HTTP verbs (or methods) are used to specify the type of action that the API client wants to perform on a resource. The most commonly used HTTP verbs in API are:

- **GET:** This verb is used to retrieve information from the server. It is the most common verb used to make a request to an API. It is considered safe, meaning that it should not have any side effects on the server or the data it serves.

- **POST:** This verb is used to submit information to the server. It is typically used to create new resources.

- **DELETE:** This verb is used to delete resources.

- **PUT:** This verb is used to update existing resources. It replaces the entire resource, unlike the PATCH verb which modifies only the fields sent in the request.

- **PATCH:** This verb is used to partially update a resource. It is used to modify only the fields sent in the request and it's usually used when you don't want to replace all the resource's attributes.

- **PUT vs PATCH:** PUT method uses the request URI to supply a modified version of the requested resource which replaces the original version of the resource, whereas the PATCH method supplies a set of instructions to modify the resource.

- **HEAD:** This verb is similar to GET, but it only returns the headers of the response, without the body.

- **OPTIONS:** This verb is used to retrieve the allowed actions on a resource.

- **CONNECT:** Connect request establishes a tunnel to the server identified by a specific URI. A good example is SSL tunnelling.

- **TRACE:** The Trace method performs a message loop-back test along the path to the target resource, to provide a useful debugging mechanism. It allows clients to view whatever message is being received at the other end of the request chain so that they can use the info for testing or diagnostic functions.

These verbs are part of the HTTP protocol, and are commonly used in RESTful (Representational State Transfer) APIs, which are built on top of the HTTP protocol and are designed to be easily consumed by web and mobile applications.

HTTP Status Codes

HTTP status codes are three-digit numbers returned by an API (Application Programming Interface) to indicate the status of a request. These codes provide information about whether a request was successful or not, and they are an important part of the API development.

The first digit of the status code specifies one of five standard classes of responses.

Standard Classes:

- **1xx:** Informational responses.
- **2xx:** Successful responses.
- **3xx:** Redirection responses.
- **4xx:** Client error responses.
- **5xx:** Server error responses.

1xx: Informational responses

It indicates that the request was received and understood by the server and it's continuing the process.

- **100:** Continue.
- **101:** Switching Protocols.
- **102:** Processing.
- **103:** Early Hints.

2xx: Successful responses

It indicates that the action requested by the client was received, understood, and accepted.

- **200:** OK.
- **201:** Created.
- **202:** Accepted.
- **203:** Non-Authoritative Information.
- **204:** No Content.

3xx: Redirection responses

Many of these 3xx status codes are used in URL redirection or it indicates the client must take additional action to complete the request.

- **301:** Moved Permanently.
- **302:** Found.
- **304:** Not Modified.
- **305:** Use Proxy.
- **307:** Temporary Redirect.
- **308:** Permanent Redirect.

4xx: Client error responses

This status code is intended for situations in which the error seems to have been caused by the client.

- **400:** Bad Request.
- **401:** Unauthorized.
- **403:** Forbidden.
- **404:** Not Found.
- **406:** Not Acceptable.
- **408:** Request Timeout.

5xx: Server error responses

It indicates that the server has encountered a situation where it doesn't know how to handle a request.

- **500:** Internal Server Error
- **501:** Not Implemented
- **502:** Bad Gateway
- **503:** Service Unavailable
- **504:** Gateway Timeout
- **505:** HTTP Version Not Supported

These are just a few examples of the many HTTP status codes that can be returned by an API. It's important to understand the meaning of each code and to handle them appropriately in the client application. The HTTP status codes are standardized and are used as a way for the server to communicate with the client about the outcome of a request.