Understanding HTTP Verbs

- HTTP verbs tell the server what to do with the data identified by the URL. The HTTP method is supplied in the request line and specifies the operation that the client has requested. If you're going to follow the REST architecture and the HTTP protocol, you must choose from the verbs available in that protocol, the primary or most-commonly-used HTTP verbs are POST, GET, PUT, and DELETE.
- These correspond to create, read, update, and delete (or CRUD) operations.

- **GET**:
- This method is used to retrieve a representation of a resource. A GET request is considered safe because as HTTP specifies, these requests are used only to read data and not change it.
- So in short there are no side effects and GET requests can be re-issued without worrying about the consequences.
- The disadvantage of GET requests is that they can only supply data in the form of parameters encoded in the URI or as cookies in the cookie request header.
- For example: Displaying your registered account details on any website has no effect on the account. If you are using Internet Explorer you can refresh a page and the page will show information that resulted from a GET, without displaying any kind of warning. We have also other HTTP components like PROXIES that automatically retry GET requests if they encounter a temporary network connection problem.
- Tasks that you can do with GET requests are:
    - You can cache the requests
    - It can remain in the browser history
    - You can bookmark the requests
    - You can apply length restrictions with GET requests
    - It is used only to retrieve data

- **POST** :
- POST is used when the processing you wish to do on the server should be repeated or when creating a new resource or for a POST to the parent when the service associates the new resource with the parent or for assigning an ID, etcetera. It is used to create new resources. For some resources, it may be used to alter the internal state. For others, its behavior may be that of a remote procedure call. If you try to refresh a page while using a POST request, then you will get an error like page can't be refreshed. Why? Because the request cannot be repeated without explicit approval by the user.
- Example: The best example of a POST request I must say is when the user submits a change and then uses a 302 redirection (Code redirection details you will find later on this article) to change to a GET that displays the result of the action as the new updated value.
- Some important points about POST requests:
    - Data will be re-submitted
    - It cannot be bookmarked
    - It cannot be cached
    - Parameters are not saved in browser history
    - No restrictions. Binary data is also allowed
    - Data is not displayed in the URL

- **PUT :**
- PUT is used to create a resource, not in a general case but when the resource ID is chosen by the client instead of by the server, then only PUT is used. Simply PUT updates data in the repository, replacing any existing data with the supplied data.
- For example: Suppose we wanted to change the image that we have something already on the server, So, we just upload a new one using the PUT verb.
- Some good points of PUT requests are:
    - It completes as possible

    - It's as seamless as possible
    - Easy to use via HTML
    - It should integrate well with servers that already support PUT

- **DELETE:**
- A DELETE request is as simple as its name implies; it just deletes, it is used to delete a resource identified by a URI and on successful deletion, it returns HTTP status 200 (OK) along with a response body. The importent and unique thing about a DELETE request is that the client cannot be guaranteed that the operation has been carried out, even if the status code returned from the origin server indicates that the action has been completed successfully.
- Some HTTP infrastructures do not support the DELETE method. In such cases, the request should be submitted as a POST with an additional X-HTTP-Method-Override header set to DELETE.
- Example DELETE request: DELETE /data/myDataApp/myorder/-/takenOrdernumber('00777')

Understanding JSON Structure

- JSON stands for JavaScript Object Notation
- JSON is a text format for storing and transporting data
- JSON is "self-describing" and easy to understand
- example is a JSON string: '{"name":"John", "age":30, "car":null}'
- JSON is a lightweight data-interchange format
- JSON is plain text written in JavaScript object notation
- JSON is used to send data between computers
- JSON is language independent *
- The JSON format is syntactically similar to the code for creating JavaScript objects. Because of this, a JavaScript program can easily convert JSON data into JavaScript objects.

- Since the format is text only, JSON data can easily be sent between computers, and used by any programming language.
- JavaScript has a built in function for converting JSON strings into JavaScript objects:JSON.parse()
- JavaScript also has a built in function for converting an object into a JSON string:JSON.stringify()
- **Data Types:**
- In JSON, values must be one of the following data types:
    - a string
    - a number
    - an object (JSON object)
    - an array
    - a boolean
    - *null*
- **JSON Object**
- JSON object literals are surrounded by curly braces {}.
- JSON object literals contains key/value pairs.
- Keys and values are separated by a colon.
- Keys must be strings, and values must be a valid JSON data type
- Each key/value pair is separated by a comma.
- myObj = {"name":"John", "age":30, "car":null};
- access object values by using dot (.) notation
- x = myObj.name;

- **JSON Array**
- Arrays in JSON are almost the same as arrays in JavaScript.
- In JSON, array values must be of type string, number, object, array, boolean or *null*.
- In JavaScript, array values can be all of the above, plus any other valid JavaScript expression, including functions, dates, and *undefined.*
- myArray = ["Ford", "BMW", "Fiat"];
- access array values by index
- myArray[0];