1) HTTP response status codes

When the client raises a request to the server through an API, the client should know the feedback, whether it failed, passed or the request was wrong. HTTP status codes are bunch of standardized codes which has various explanations in various scenarios. The server should always return the right status code.
The following are the important categorization of HTTP codes:

2xx (Success category)

These status codes represent that the requested action was received and successfully processed by the server.

- **200 Ok** The standard HTTP response representing success for GET, PUT or POST.
- **201 Created** This status code should be returned whenever the new instance is created. E.g on creating a new instance, using POST method, should always return 201 status code.
- **204 No Content** represents the request is successfully processed, but has not returned any content.
- DELETE can be a good example of this.
- The API DELETE /companies/43/employees/2 will delete the employee 2 and in return we do not need any data in the response body of the API, as we explicitly asked the system to delete. If there is any error, like if employee 2 does not exist in the database, then the response code would be not be of 2xx Success Category but around 4xx Client Error category.

3xx (Redirection Category)

- **304 Not Modified** indicates that the client has the response already in its cache. And hence there is no need to transfer the same data again.

4xx (Client Error Category)

These status codes represent that the client has raised a faulty request.

- **400 Bad Request** indicates that the request by the client was not processed, as the server could not understand what the client is asking for.
- **401 Unauthorized** indicates that the client is not allowed to access resources, and should re-request with the required credentials.
- **403 Forbidden** indicates that the request is valid and the client is authenticated, but the client is not allowed access the page or resource for any reason. E.g sometimes the authorized client is not allowed to access the directory on the server.
- **404 Not Found** indicates that the requested resource is not available now.
- **409 Conflict** The request could not be completed due to a conflict with the current state of the resource (it will be used depending on project requirements).
- **410 Gone** indicates that the requested resource is no longer available which has been intentionally moved (it will be used depending on project requirements).

- **422 Unprocessable Entity** - The server understands the content type of the request entity and the syntax of the request entity is correct but was unable to process the contained instructions..
- **429 Too Many Requests** - When a request is rejected due to rate limiting(it will be used depending on project requirements).

5xx (Server Error Category)

- **500 Internal Server Error** indicates that the request is valid, but the server is totally confused and the server is asked to serve some unexpected condition.
- **503 Service Unavailable** indicates that the server is down or unavailable to receive and process the request. Mostly if the server is undergoing maintenance.

For More error Please also look into :
https://cloud.google.com/storage/docs/json_api/v1/status-codes

2) Use RESTful URLs and actions

The key principles of REST involve separating your API into logical resources. These resources are manipulated using HTTP requests where the method (GET, POST, PUT, PATCH, DELETE) has specific meaning.

- GET /tickets - Retrieves a list of tickets
- GET /tickets/- Retrieves a specific ticket
- POST /tickets - Creates a new ticket
- PUT /tickets/ - Updates ticket #ticket id
- PATCH /tickets/ - Partially updates ticket #ticket id
- DELETE /tickets/ - Deletes ticket #ticket

Dealing with relations

- GET /tickets/:id/messages - Retrieves list of messages for #ticket_id
- GET /tickets/:id/messages/:msg_id - Retrieves message #msg_id for #ticket_id
- POST /tickets/:id/messages - Creates a new message in #ticket_id
- PUT /tickets/:id/messages - Updates message for ticket #ticket_id
- PATCH /tickets/:id/messages - Partially updates message #msg_id
- DELETE /tickets/:id/messages - Deletes message for ticket #ticket_id

3) SSL everywhere - all the time

All HTTP calls should be redirected to HTTPS apache/nginx should be configured to redirect

4) Result filtering, sorting & searching

Limit the result set
To define the maximum amount of datasets in the result, use limit.
/tickets?limit=50
To define the offset of the datasets in the result, use offset.
/tickets?offset=20&limit=50
Be aware that in order to use offset you always have to specify a limit too.
Please refere the below URL for complex sort, Pagination and filtering options
https://www.thrinacia.com/blog/post/rest-api-sorting-paging-and-filtering

5) Add HTTP Caching

HTTP caching should be added to specific request which is based on business logics.
Eg : Request to get country, city, state etc

6) Errors handling

Below is a sample response used in custom error handling .
Example in exception, pdo error, file permission error etc

```
{
        "error": [
        {
        "param": "email",
        "msg": "\"\" must be valid email"
        }       ]
}
```

8. Version your API

Make the API Version mandatory and do not release an unversioned API.
Eg : /blog/api/v1

9) Rate limiting or throttle

Rate limiting or throttle should it be handled in server side (Apache) or app

10) Code commenting :

Example Below :

```
/**
  * @description Saves message to log table
  * @param string $msg - the message to be saved
  * @return bool - success or failure
*/
function postLog($msg)
{

}
```

11) Naming conventions :

Should use underscore for request and response & table

12 CRUD Example Below :

GET => https://127.0.0.1/v1/tickets

Response :

HTTP/1.1 **200**

Content-Type: application/json


```
{
        "data" :[{
                "id": 1,
                "message": "message",
                "created_at": "2017-11-03 18:41:06",
                "updated_at": "2017-11-03 18:41:06"
        },
        {
                "id": 2,
                "message": "message",
                "created_at": "2017-11-03 18:41:06",
                "updated_at": "2017-11-03 18:41:06"
        }]
}
```

GET => https://127.0.0.1/v1/tickets/1

Response :

HTTP/1.1 **200**
Content-Type: application/json

"data" :{
   "id": 1,
   "message": "message"
}

POST => https://127.0.0.1/v1/tickets

Request :
{
   "message": "message"
}

Response :
HTTP/1.1 **201**
Content-Type: application/json

"data" :{
"id": 1
}

PUT => https://127.0.0.1/v1/tickets

Request :

"data" :{
   "id": "1",
   "message": "updated message"
}
Response :
HTTP/1.1 **200**
Content-Type: application/json

"data" :{
   "message": "updated message",
   "updated_at": "2017-11-16T12:57:16.300Z"
}

PATCH => https://127.0.0.1/v1/tickets

Request :

"data" :{
   "id": "1",
   "message": "updated message"
}
Response :
HTTP/1.1 **200**
Content-Type: application/json

"data" :{
   "message": "updated message",
   "updated_at": "2017-11-16T12:57:16.300Z"
}

DELETE => https://127.0.0.1/v1/tickets

Request :

"data" :{
   "id": "1"
}

Response :
HTTP/1.1 **204**

Exception Cases :

POST => https://127.0.0.1/v1/tickets

Request :

"data" :{
   "unavailable_field": "sample field"
}
Response :
HTTP/1.1 **400**
Content-Type: application/json

{
	"error": [
	{
	"param": "email",

```
        "msg": "\"\" must be valid email"
        },
        {
        "param": "password",
        "msg": "The password length must be between 6 and 25 characters"
        },
        {
        "param": "first_name",
        "msg": "Field cannot be blank"
        },
        {
        "param": "first_name",
        "msg": "\"\" must contain only letters (a-z)"
        },
        {
        "param": "last_name",
        "msg": "Field cannot be blank"
        },
        {
        "param": "last_name",
        "msg": "\"\" must contain only letters (a-z)"
        },
        {
        "param": "social_type",
        "msg": "\"\" must be an integer number"
        },
        {
        "param": "verified",
        "msg": "\"\" must be an integer number"
        }
        ]
}
```

GET => https://127.0.0.1/v1/tickets/2000

Response :

HTTP/1.1 **404**

Content-Type: application/json


```
{}
```

Reference : https://reqres.in/

13 JSON Web Token structure :

In its compact form, JSON Web Tokens consist of three parts separated by dots (.), which are:
Header
Payload
Signature

Therefore, a JWT typically looks like the following.
xxxxx.yyyyy.zzzzz

Encoded JWT
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE1MjEwMjQ4NTAsImV4cCI6MTgzNjM4NDg1MCwiZGF0YSI6eyJ1c2VySWQiOjExfSwiaXNzIjoibG9jYWxob3N0In0.dIdev5hUG0D3TUTSeLy30fZRzp9pHE_zWKH1-DkbwTo

Decoded JWT:
HEADER
```
{
  "typ": "JWT",
  "alg": "HS256"
}
```
PAYLOAD:
```
{
  "iat": 1521024850,
  "exp": 1836384850,
  "data": {
          "userId": 11
  },
  "iss": "localhost"
}
```

SIGNATURE:
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Reference : https://jwt.io/introduction/
For more status code in header refer the URL https://httpstatuses.com/