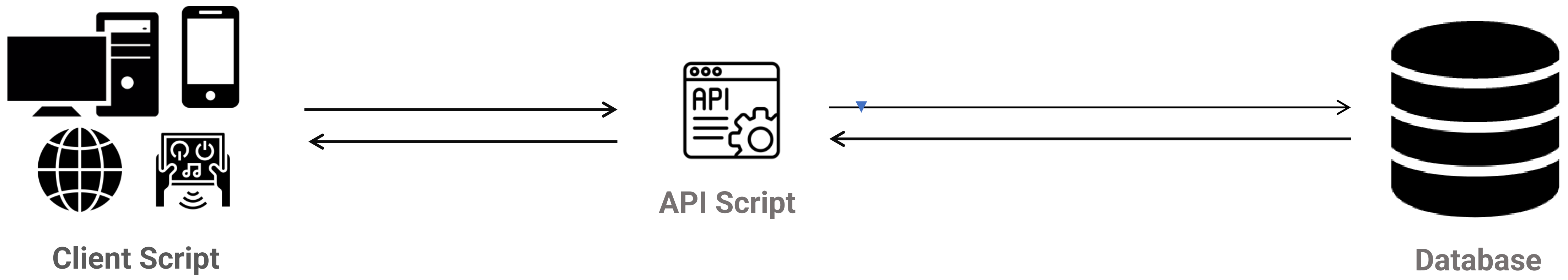
# Make A note on web Back-End DEVELOPMENT

## Assignment Topic

- Introduction of Rest API
- Rest API Best Practices
- JSON Best Practices
- Http methods Best Practices
- Request-Response Best Practices
- Web Security Practices
- 50 Interview Question Back-end Development

# Introduction to Rest API

## Representational State Transfer



## Best practices of json

### **JavaScript Object Notation (JSON)**

- JSON is a lightweight data-interchange format that is completely language independent.
- It was derived from JavaScript, but many modern programming languages include code to generate and parse JSON-format data
- The official Internet media type for JSON is application/json.
- It was designed for human-readable data interchange.
- The filename extension is .json.

## Best practices of json

### <u>Uses of JSON</u>

- It is used while writing JavaScript based applications that includes browser extensions and websites.
- JSON format is used for serializing and transmitting structured data over network connection.
- It is primarily used to transmit data between a server and web applications.
- Web services and APIs use JSON format to provide public data.

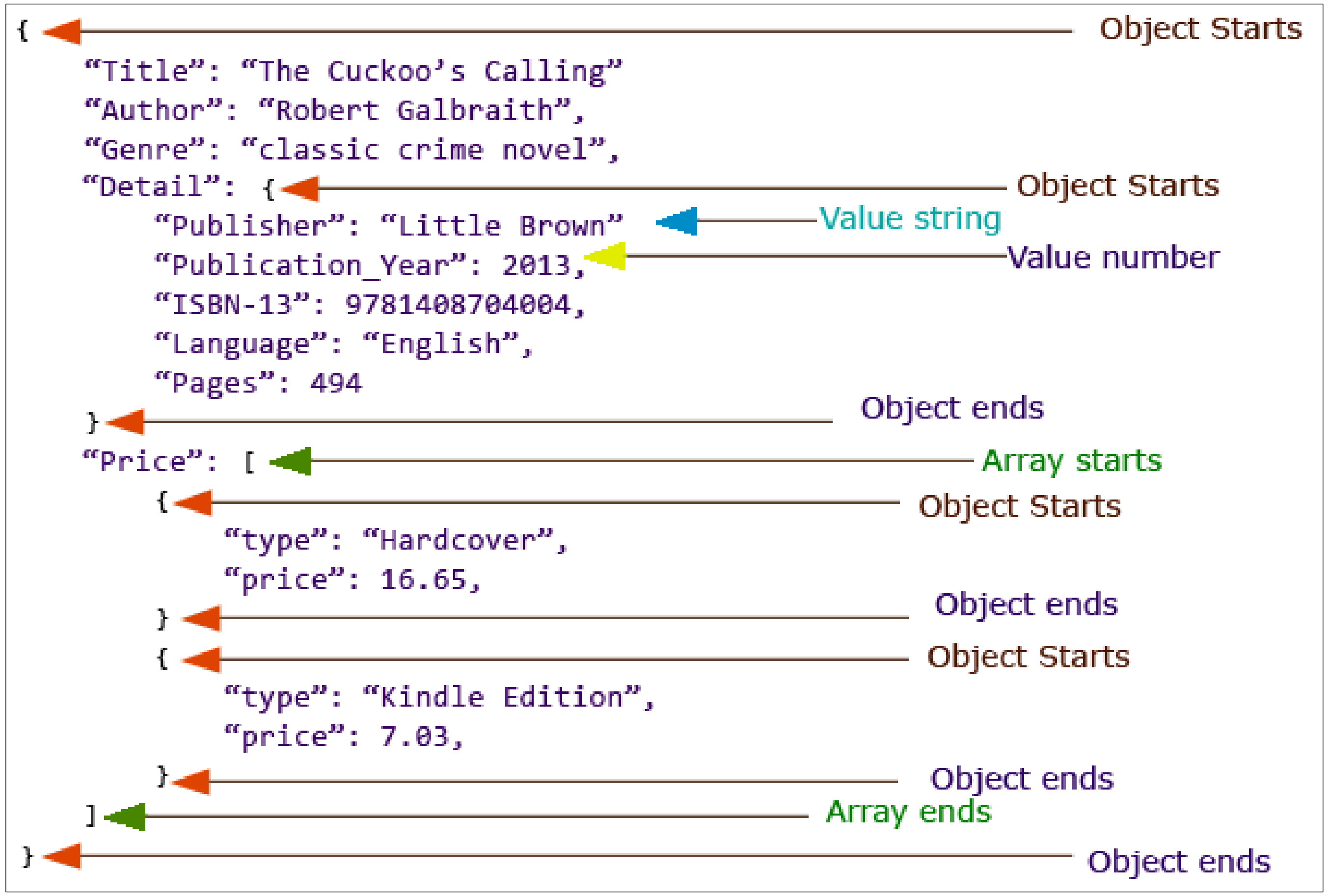
## Best practices of json

**Characteristics of JSON**

- JSON is easy to read and write.
- It is a lightweight text-based interchange format.
- JSON is language independent.

# Best practices of json

## Understanding JSON Structure



```
{
    "Title": "The Cuckoo's Calling"
    "Author": "Robert Galbraith",
    "Genre": "classic crime novel",
    "Detail": {
        "Publisher": "Little Brown"
        "Publication_Year": 2013,
        "ISBN-13": 9781408704004,
        "Language": "English",
        "Pages": 494
    }
    "Price": [
        {
            "type": "Hardcover",
            "price": 16.65,
        }
        {
            "type": "Kindle Edition",
            "price": 7.03,
        }
    ]
}
```

## Best practices of json

**JSON - Data Types**

| Type | Description |
|---|---|
| Number | Double- precision floating-point format in JavaScript |
| String | Double-quoted Unicode with backslash escaping |
| Boolean | True or False |
| Array | An ordered sequence of values |
| Value | It can be a string, a number, true or false, null etc |
| Object | An unordered collection of key:value pairs |
| Whitespace | Can be used between any pair of tokens |
| null | Empty |

# Best practices of json

Bad Special Characters & Solution

| Characters | Replace with |
|---|---|
| Backspace | \b |
| Form feed | \f |
| Newline | \n |
| Carriage return | \r |
| Tab | \t |
| Double quote | \" |
| Backslash | \\ |

## Best practices of json

- Always enclose the **Key : Value** pair within double quotes

```
{'id': '1','name':File} is not right ✗

{"id": 1,"name":"File"} is okay  ✓

{"id": "1","name":"File"} is the best  ✓
```

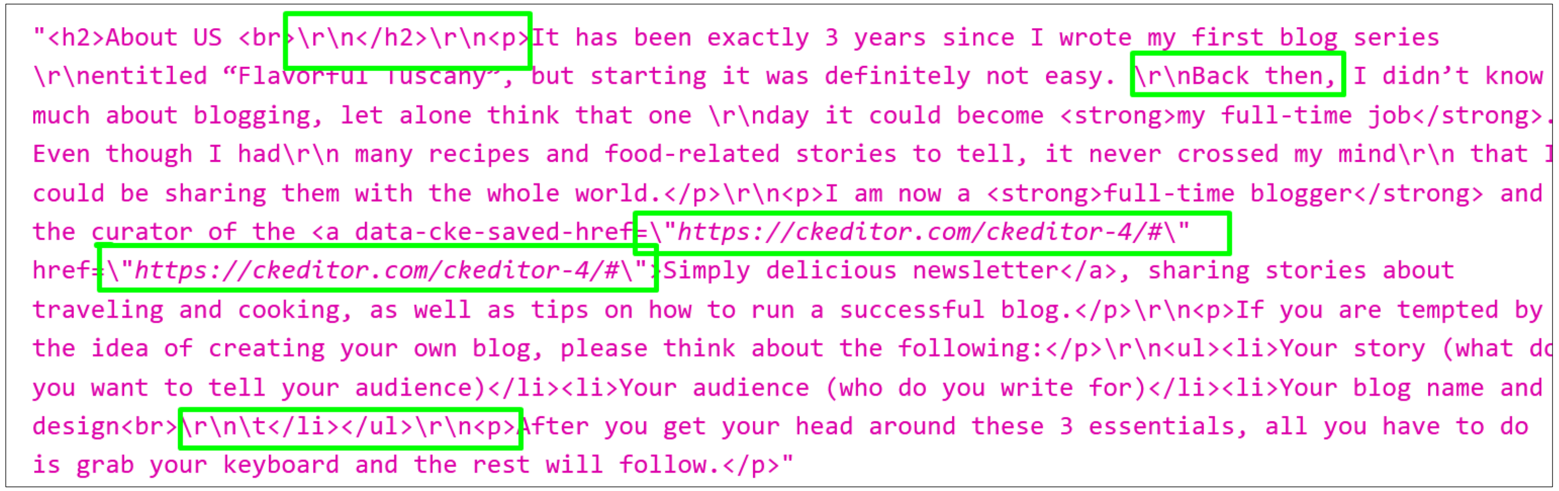
# Best practices of json

- Never use Hyphens in your Key fields

```
{"first-name":"Rachel","last-name":"Green"}  is not right. ✗

{"first_name":"Rachel","last_name":"Green"} is okay ✓

{"firstname":"Rachel","lastname":"Green"} is okay ✓

{"firstName":"Rachel","lastName":"Green"} is the best. ✓
```

# Best practices of json

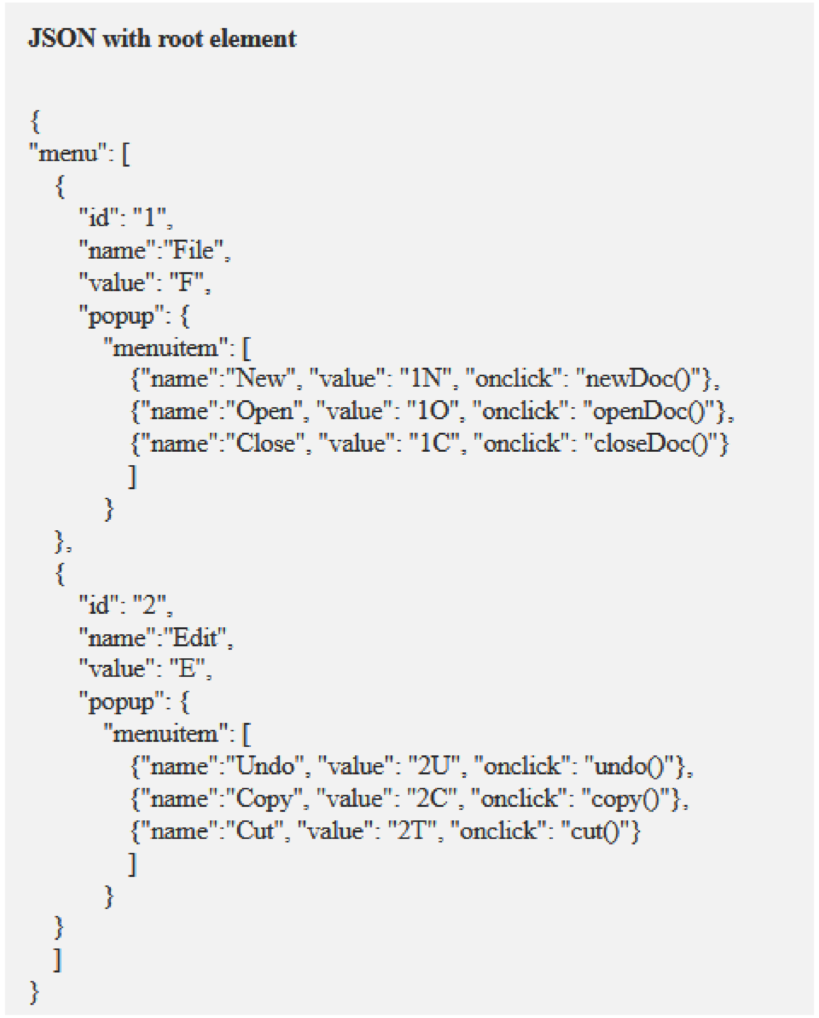
- Bad Special Characters And Solution



```
"<h2>About US <br>\r\n</h2>\r\n<p>It has been exactly 3 years since I wrote my first blog series
\r\nentitled “Flavorful Tuscany”, but starting it was definitely not easy. \r\nBack then, I didn’t know
much about blogging, let alone think that one \r\nday it could become <strong>my full-time job</strong>.
Even though I had\r\n many recipes and food-related stories to tell, it never crossed my mind\r\n that I
could be sharing them with the whole world.</p>\r\n<p>I am now a <strong>full-time blogger</strong> and
the curator of the <a data-cke-saved-href=\"https://ckeditor.com/ckeditor-4/#\"
href=\"https://ckeditor.com/ckeditor-4/#\">Simply delicious newsletter</a>, sharing stories about
traveling and cooking, as well as tips on how to run a successful blog.</p>\r\n<p>If you are tempted by
the idea of creating your own blog, please think about the following:</p>\r\n<ul><li>Your story (what d
you want to tell your audience)</li><li>Your audience (who do you write for)</li><li>Your blog name and
design<br>\r\n\t</li></ul>\r\n<p>After you get your head around these 3 essentials, all you have to do
is grab your keyboard and the rest will follow.</p>"
```

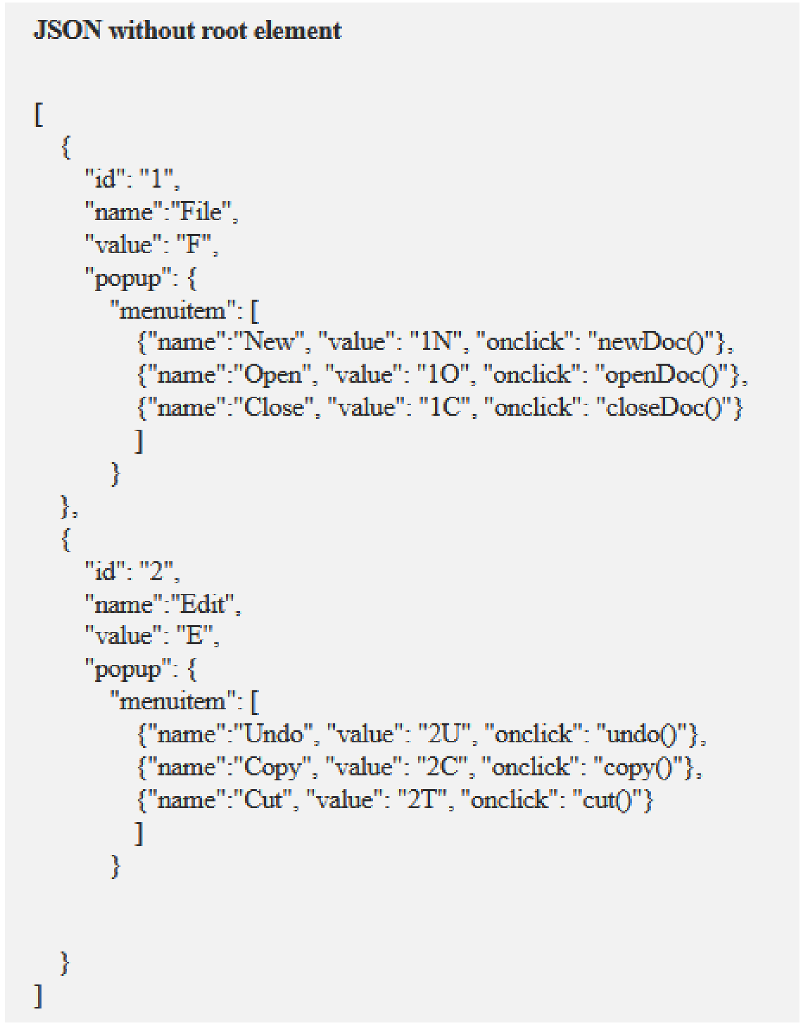
## Best practices of json

- Always create a Root element.



**JSON with root element**

```
{
"menu": [
   {
      "id": "1",
      "name":"File",
      "value": "F",
      "popup": {
         "menuitem": [
            {"name":"New", "value": "1N", "onclick": "newDoc()"},
            {"name":"Open", "value": "1O", "onclick": "openDoc()"},
            {"name":"Close", "value": "1C", "onclick": "closeDoc()"}
            ]
         }
   },
   {
      "id": "2",
      "name":"Edit",
      "value": "E",
      "popup": {
         "menuitem": [
            {"name":"Undo", "value": "2U", "onclick": "undo()"},
            {"name":"Copy", "value": "2C", "onclick": "copy()"},
            {"name":"Cut", "value": "2T", "onclick": "cut()"}
            ]
         }
   }
   ]
}
```



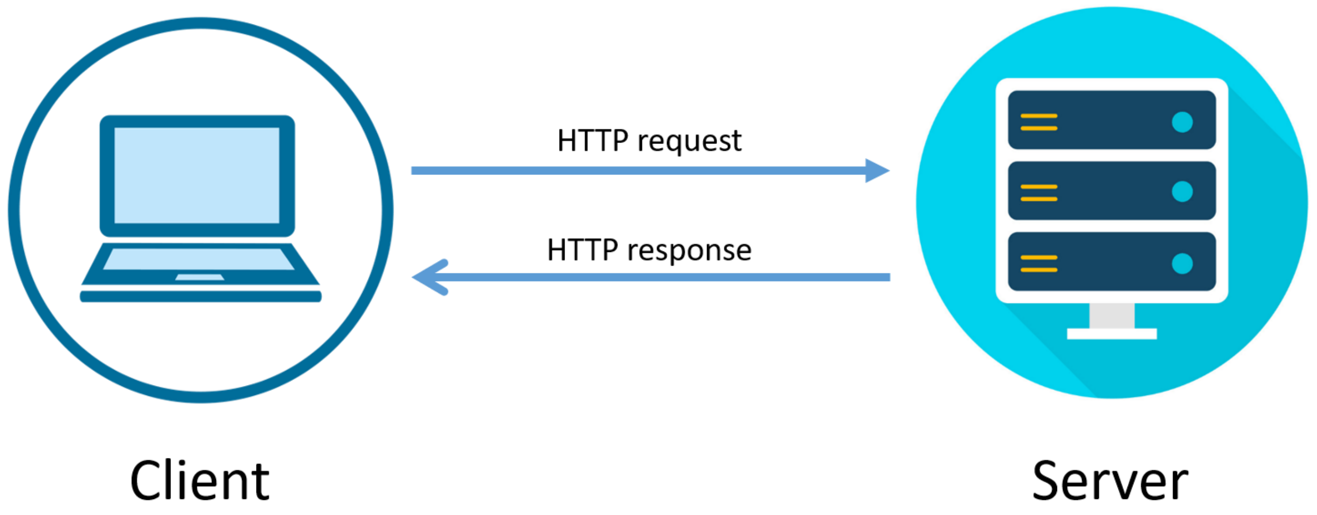
**JSON without root element**

```
[
   {
      "id": "1",
      "name":"File",
      "value": "F",
      "popup": {
         "menuitem": [
            {"name":"New", "value": "1N", "onclick": "newDoc()"},
            {"name":"Open", "value": "1O", "onclick": "openDoc()"},
            {"name":"Close", "value": "1C", "onclick": "closeDoc()"}
            ]
         }
   },
   {
      "id": "2",
      "name":"Edit",
      "value": "E",
      "popup": {
         "menuitem": [
            {"name":"Undo", "value": "2U", "onclick": "undo()"},
            {"name":"Copy", "value": "2C", "onclick": "copy()"},
            {"name":"Cut", "value": "2T", "onclick": "cut()"}
            ]
         }


   }
]
```

## Request response Best Practices

### <u>HTTP/HTTPS Request Response Communication</u>

- In request/response communication mode.
- One software module sends a request to a second software module and waits for a response.
- The First software module performs the role of the client.
- The second, the role of the server,
- This is called client/server interaction.



## Request response Best Practices

HTTP Client:.

- Browser Level Client
- Application Level Client

# Request response Best  Practices

## Browser Level Client

- Browser is the primary HTTP Client responsible for load the web application.

## Request response Best Practices

### Application Level Client :

- HTTP Client is an application library used in client side application to generate request and receive response.
- HTTP Client's libraries varies from platform to platform

| HTTP Client Library | Platform | Language |
|---|---|---|
| Volly | Native Android | Java |
| Retrofit | Native Android | Java |
| Rest Sharp | ASP.NET | C# |
| Axios | Mobile/Web/Desktop | JavaScript |
| cURL | Web | PHP |
| Alamofire | Native IOS | Swift |

# Request response Best Practices

## POSTMAN Http Client

Postman is an HTTP Client application, used to test request-response communication.
Postman is widely used for API testing and generating documentation.

- Quickly and easily send REST, SOAP, and GraphQL requests directly within Postman.
- Generate and publish beautiful, machine-readable API documentation.
- Checking performance and response times at scheduled intervals.
- Communicate the expected behavior of an API by simulating endpoints and their responses

## Request response Best Practices

<u>HTTP Request</u>

HTTP Request is the first step to initiate web request/response communication. Every request is a combination of request header, body and request URL.

Http Request Segments:

| Request Area | Standard Data Type |
|---|---|
| **Body** | Simple String, JSON, Download, Redirect, XML |
| **Header** | Key Pair Value |
| **URL Parameter** | String |

# Request response Best practices

HTTP Request Methods:

| Method Name | Responsibilities |
|---|---|
| **GET()** | The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data. |
| **Head()** | Same as GET, but transfers the status line and header section only. |
| **POST()** | A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms. |
| **PUT()** | Replaces all current representations of the target resource with the uploaded content. |
| **DELETE()** | Removes all current representations of the target resource given by a URI. |

## Request response  Best Practices

Request Compare GET vs. POST:

| Key Points | GET | POST |
|---|---|---|
| BACK button/Reload | Harmless | Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted) |
| Bookmarked | Can be bookmarked | Cannot be bookmarked |
| Cached | Can be | Never |
| Encoding type | application/x-www-form-urlencoded | application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data |
| History | Parameters remain in browser history | Parameters are not saved in browser history |
| Restrictions on data length | Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters) | No restrictions |

## Request response Best Practices

Request Compare GET vs. POST:

| Key Points | GET | POST |
|---|---|---|
| Restrictions on data type | Only ASCII characters allowed | No restrictions. Binary data is also allowed |
| Security | GET is less secure compared to POST because data sent is part of the URL. Never use GET when sending passwords or other sensitive information! | POST is a little safer than GET because the parameters are not stored in browser history or in web server logs |
| Visibility | Data is visible to everyone in the URL | Data is not displayed in the URL |

## Request response Best Practices

**Http Request Throttling:**

Throttle Request refers to a process in which a user is allowed to hit the application maximum time in per second or per minute.  Throttling is also known as request rate limiting.

- Essential component of Internet security, as DoS attacks can tank a server with unlimited requests.
- Rate limiting also helps make your API scalable by avoid unexpected spikes in traffic, causing severe lag time.

## Request response Best Practices

**HTTP Response:**

Http response is the final step of request-response communication. Every response is a combination of response header, body an d cookies.

**Http Response Segments:**

| Response Area | Standard Data Type |
|---|---|
| **Body** | Simple String, JSON, Download, Redirect, XML |
| **Header** | Key Pair Value |
| **Cookies** | Key Pair Value |

## Request response Best Practices

**HTTP Response status messages**

| Code | Meaning | **Description** |
|---|---|---|
| 200 | OK | The request is OK (this is the standard response for successful HTTP requests) |
| 201 | Created | The request has been fulfilled, and a new resource is created |
| 202 | Accepted | The request has been accepted for processing, but the processing has not been completed |
| 203 | Non-Authoritative Information | The request has been successfully processed, but is returning information that may be from another source |
| 204 | No Content | The request has been successfully processed, but is not returning any content |
| 205 | Reset Content | The request has been successfully processed, but is not returning any content, and requires that the requester reset the document view |
|  |  |  |

## Request response Best Practices

**HTTP Response status messages**

| Code | Meaning | Description |
|---|---|---|
| 206 | Partial Content | The server is delivering only part of the resource due to a range header sent by the client |
| 400 | Bad Request | The request cannot be fulfilled due to bad syntax |
| 401 | Unauthorized | The request was a legal request, but the server is refusing to respond to it. |
| 403 | Forbidden | The request was a legal request, but the server is refusing to respond to it |
| 404 | Not Found | The requested page could not be found but may be available again in the future |
| 405 | Method Not Allowed | A request was made of a page using a request method not supported by that page |

## Request response Best Practices

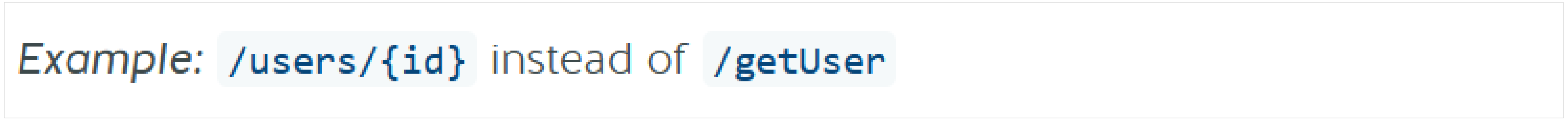
**HTTP Response status messages**

| Code | Meaning | Description |
|---|---|---|
| 408 | Request Timeout | Request Timeout |
| 500 | Internal Server Error | A generic error message, given when no more specific message is suitable |
| 502 | Bad Gateway | The server was acting as a gateway or proxy and received an invalid response from the upstream server |
| 503 | Service Unavailable | The server is currently unavailable (overloaded or down) |
| | | |
| | | |

## Request response Best Practices

**URIs as resources as nouns:**

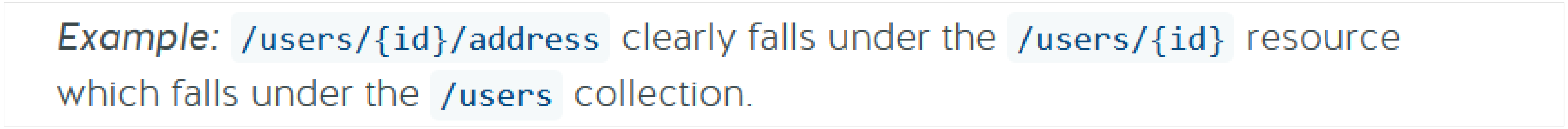
One of the most recognizable characteristics of REST is the predominant use of nouns in URIs. Restful URIs should not indicate any kind of CRUD (Create, Read, Update, and Delete) functionality. Instead, REST APIs should allow you to manipulate a resource.



*Example:* `/users/{id}` instead of `/getUser`

## API End Naming Best Practice

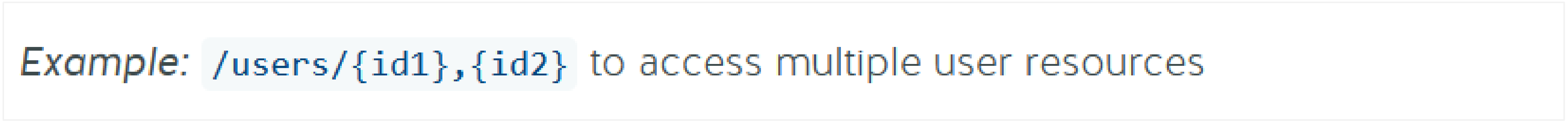
Forward slashes for hierarchy:
As shown in the examples above, forward slashes are conventionally used to show the hierarchy between individual resources and collections


*Example:* `/users/{id}/address` clearly falls under the `/users/{id}` resource which falls under the `/users` collection.

# API End Naming Best Practice

**Punctuation for lists:**

When there is no hierarchical relationship (such as in lists), punctuation marks such as the semicolon, or, more frequently, the comma should be used.

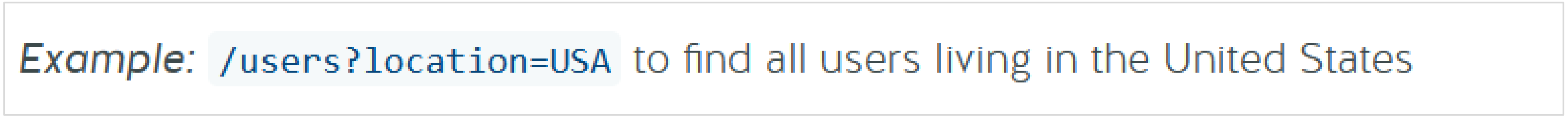


*Example:* `/users/{id1},{id2}` to access multiple user resources

## API End Naming Best Practice

**Query parameters where necessary:**

In order to sort or filter a collection, a REST API should allow query parameters to be passed in the URI.

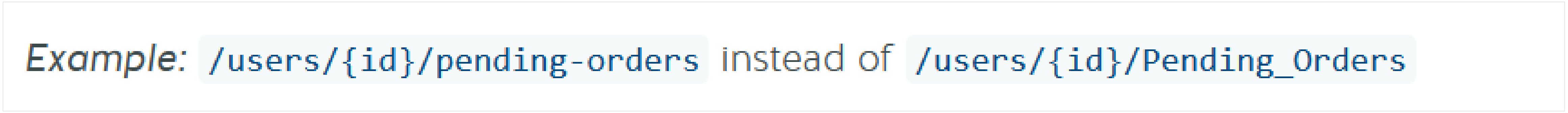


*Example:* `/users?location=USA` to find all users living in the United States

# API End Naming Best Practice

**Lowercase letters and dashes:**

By convention, resource names should use exclusively lowercase letters. Similarly, dashes (-) are conventionally used in place of underscores (_).

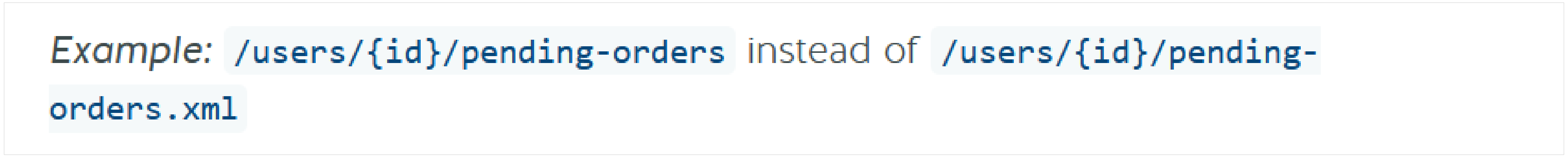


*Example:* `/users/{id}/pending-orders` instead of `/users/{id}/Pending_Orders`

## API End Naming Best Practice

No file extensions:

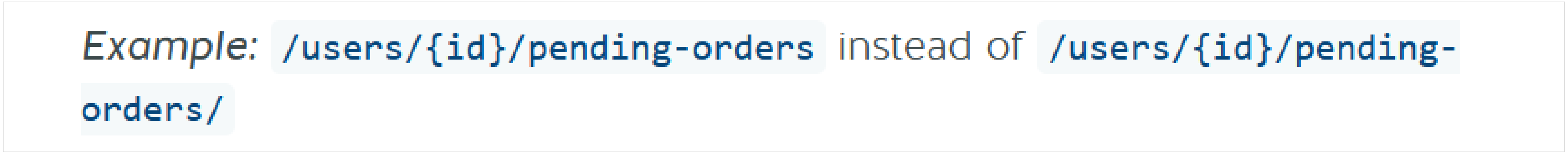
Leave file extensions (such as .xml) out of your URIs. We’re sorry to say it, but they’re ugly and add length to URIs. If you need to specify the format of the body, instead use the Content-Type header



*Example:* `/users/{id}/pending-orders` instead of `/users/{id}/pending-orders.xml`

## API End Naming Best Practice

**No trailing forward slash:** Similarly, in the interests of keeping URIs clean, do not add a trailing forward slash to the end of URIs.



*Example:* `/users/{id}/pending-orders` instead of `/users/{id}/pending-orders/`

## API Response Best Practices

### Response Header:

- Provide proper http response status code.
- Provide proper content type, file type if any.
- Provide cache status if any.
- Authentication token should provide via response header.
- Only string data is allowed for response header.
- Provide content length if any.
- Provide response date and time.
- Follow request-response model described before.

## API Response Best Practices

### Response Body:

- Avoid providing response status, code, message via response body
- Use JSON best practices for JSON response body.
- For single result, can use String, Boolean directly.
- Provide proper JSON encode-decode before writing JSON Body.
- Follow discussion on JSON described before.

## API Response Best Practices

### Response Cookies:

- A Restful API may send cookies just like a regular Web Application that serves HTML
- Avoid using response cookies as it is violate stateless principle.
- If required use cookie encryption, decryption and other policies

## API Request Handling Best Practices

### **When use GET():**

- GET is used to request something from server with less amount of data to pass.
- When nothing should change on the server because of your action.
- When request only retrieves data from a web server by specifying parameters
- Get method only carries request url & header not request body.

### When use POST():

- POST should be used when the server state changes due to that action.
- When request needs its body, to pass large amount of data.
- When want to upload documents , images , video from client to server

## API Request Handling Best Practices

Request Body:

- Request body should be structured in JSON Array/ Object pattern
- Request body hold multipart/ form-data like images, audio, video etc
- Request body should not hold any auth related information.
- Request body should associated with specific request data model, setter getter can used for this

Request Header:

- Request header should carry all security related information, like token, auth etc.
- Only string Key:Pair value is allowed for header .
- Request header should provide user agent information of client application.
- If necessary CSRF/ XSRF should provide via header.
- Request header should associated with middleware controller, where necessary

## API Request Handling Best Practices

### API Controller Best Practices

- The controllers should always be as clean as possible. We shouldn’t place any business logic inside it.
- Controllers should be responsible for accepting http request
- Consider API versioning
- Use async/await if at all possible.
- Follow solid principles to manage controller classes.
- Mention which method is responsible for GET() and which for POST().
- Controller should be only responsible for calling model, return response , redirect to action etc.

## API Request Handling Best Practices

### API Middleware Controller Best Practices

Middleware is a special types of controller executed after request but before in response. It is a type of filtering mechanism to ensure API securities and more. Middleware acts as a bridge between a request and a response.

**Middleware Uses:**

- Use to implement API key, user-agent restriction, CSRF, XSRF security, token based API authentication.
- Use to implement API request rate limit.
- Logging of incoming HTTP requests.
- Redirecting the users based on requests.
- Middleware can inspect a request and decorate it, or reject it, based on what it finds.
- Middleware is most often considered separate from your application logic.
- Middleware gives you enough freedom to create your own security mechanism.

# REST API Security Best Practices

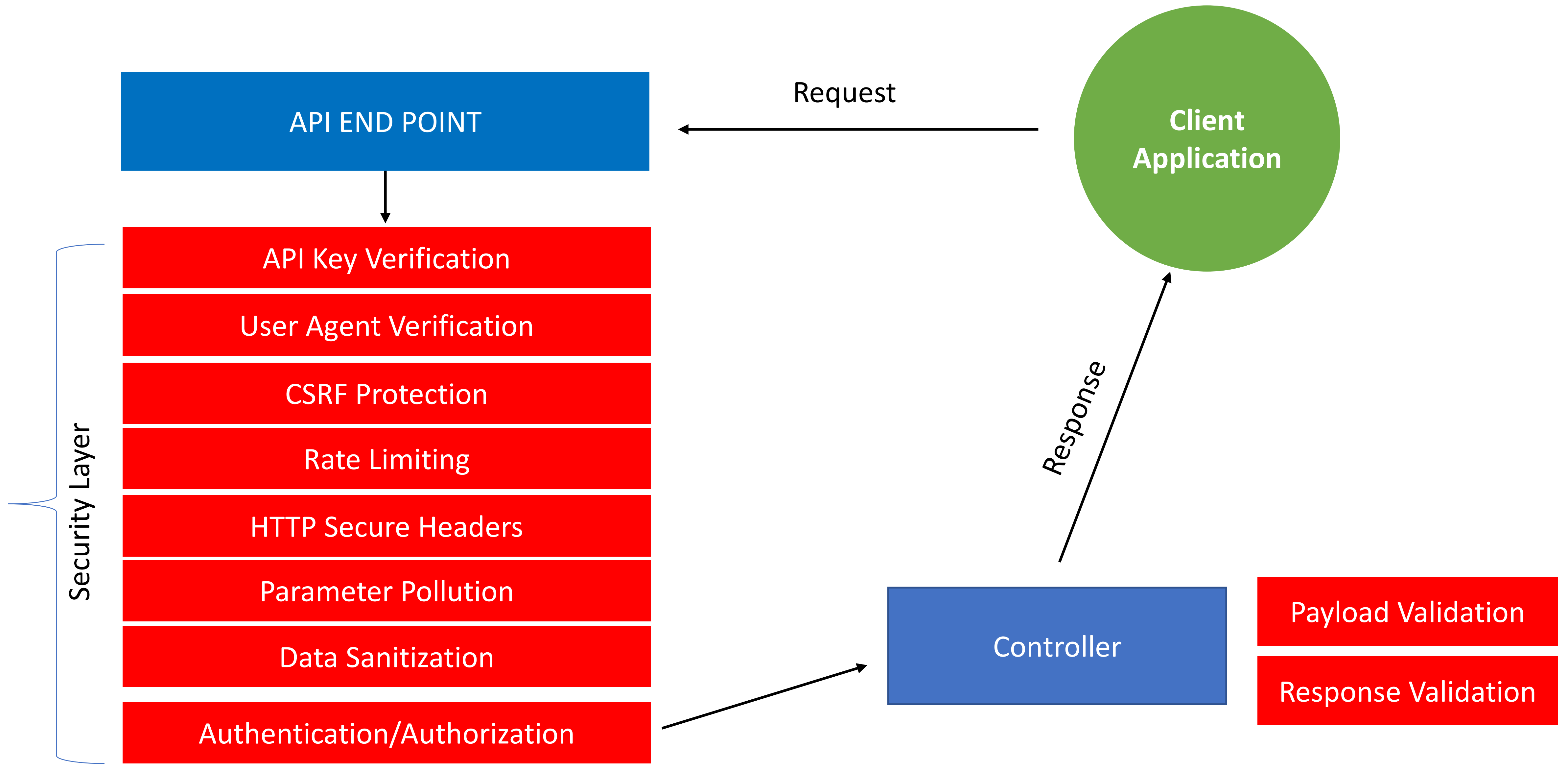
## Security Practices May Varies

1. May varies from application to application
2. May varies from developer to developer
3. May varies from environment to environment
4. May varies from use case to use case

## But we have to know

1. The best practices
2. Know about the security layers
3. Security placement

# REST API Security



# REST API Security Best Practices

## Output Validation

**Output Header:**

- Provide proper http response status code.
- Provide proper content type, file type if any.
- Provide cache status if any.
- Authentication token should provide via response header.
- Only string data is allowed for response header.
- Provide content length if any.
- Provide response date and time.
- Follow request-response model described before.

**Output Body:**

- Avoid providing response status, code, message via response body
- Use JSON best practices for JSON response body.
- For single result, can use String, Boolean directly.
- Provide proper JSON encode-decode before writing JSON Body.
- Follow discussion on JSON described before.

## REST API Security Best Practices

**Request Rate limit- Throttling**

We need to make sure our APIs are running as efficiently as possible. Otherwise, everyone using your database will suffer from slow performance. Performance isn't the only reason to limit API requests, either. API limiting, which also known as rate is limiting, is an essential component of Internet security, as DoS attacks can tank a server with unlimited API requests. Rate limiting also helps make your API scalable. If your API blows up in popularity, there can be unexpected spikes in traffic, causing severe lag time.

| Language | Platform | Library name | Library Sources |
|---|---|---|---|
| **C#** | ASP.NET | WebApiThrottle, MvcThrottle | Nuget package manager |
| **PHP** | Laravel | Laravel Karnel Default | Packagist |
| **JS** | Node/Express JS | express-rate-limit | NPM |

# REST API Security Best Practices

## CSRF/XSRF Protection

Cross-site request forgery attacks (CSRF or XSRF for short) are used to send malicious requests from an authenticated user to a web application.

- Use request-response header to pass CSRF token
- CSRF token should be unique for every session
- For self API CSRF token works well.

| Language | Platform | Library name | Library Sources |
|---|---|---|---|
| **C#** | ASP.NET | AntiCSRF | Nuget package manager |
| **PHP** | Laravel | Laravel Default | Packagist |
| **JS** | Node/Express JS | npm  i csrf | NPM |

## REST API Security Best Practices

**User Agent Protection**

User agent is a request header property, describe client identity like operating system, browser details, device details etc. Moreover every web crawler like Google crawler, Facebook crawler has specific user-agent name.

- Using user agent we can prevent REST API from search engine indexing, social media sharing.
- Can stop subspecies request from who is hiding his identity.
- We can add user agent along with REST API usage history.
- We can add device/OS usage restriction.

## REST API Security Best Practices

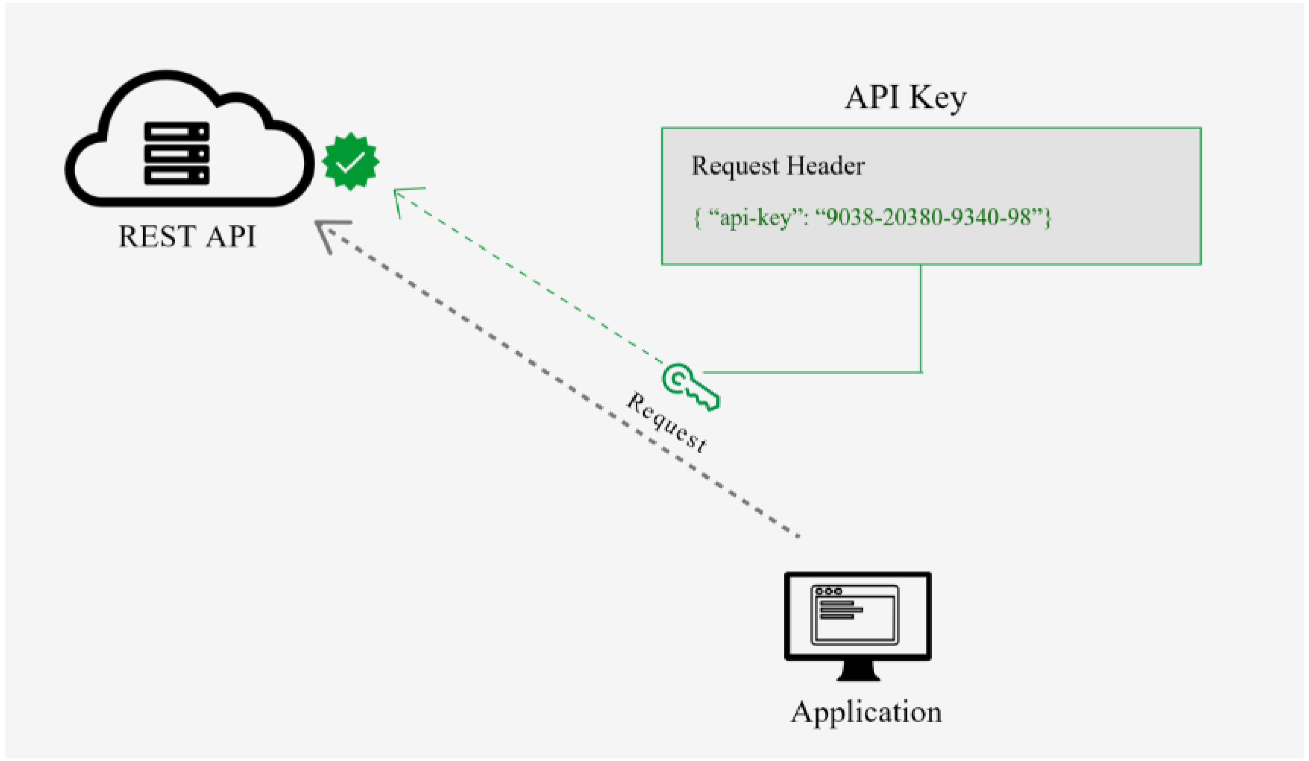
| Platform | Example User Agent Like |
|---|---|
| **Android web browser** | Mozilla/5.0 (Linux; Android 6.0.1; RedMi Note 5 Build/RB3N5C; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/68.0.3440.91 Mobile Safari/537.36 |
| **IOS web browser** | Mozilla/5.0 (iPhone; CPU iPhone OS 12_3_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.1.1 Mobile/15E148 Safari/604.1 |
| **Windows** | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36 |
| **Mac** | Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/11.1.2 Safari/605.1.15 |
| **Google BOT** | Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html) |
| **Facebook BOT** | facebookexternalhit/1.0 (+http://www.facebook.com/externalhit_uatext.php) |

## REST API Security Best Practices

**API Key**

- This is the most straightforward method and the easiest way for auth
- With this method, the sender places a **username:password/ ID / Keys** into the request header.
- The credentials are encoded and decode to ensure safe transmission.
- This method does not require cookies, session IDs, login pages, and other such specialty solutions

```
Authorization: Basic bG9sOnNlY3VyZQ==
```



# REST API Security Best Practices

**Bearer Authentication/ Auth 2.0**

Bearer authentication (also called token authentication) is an HTTP authentication scheme that involves security tokens called bearer tokens, passes through request-response header. In General JSON Web Tokens JWT used for this purposes.

| Language | Platform | Library name | Library Sources |
|---|---|---|---|
| **C#** | ASP.NET | JwtBearer, jose-jwt | Nuget package manager |
| **PHP** | Laravel | firebase / php-jwt | GitHub |
| **JS** | Node/Express JS | npm i jsonwebtoken | NPM |

## REST API Security Best Practices

**JWT (JSON WEB TOKEN):**

- Compact and self-contained way for securely transmitting information between parties as a JSON object.
- Information can be verified and trusted because it is digitally signed.

**USES:**

- **Authorization:** Allowing the user to access routes, services, and resources
- **Information Exchange**: Way of securely transmitting information between parties.

## REST API Security Best Practices

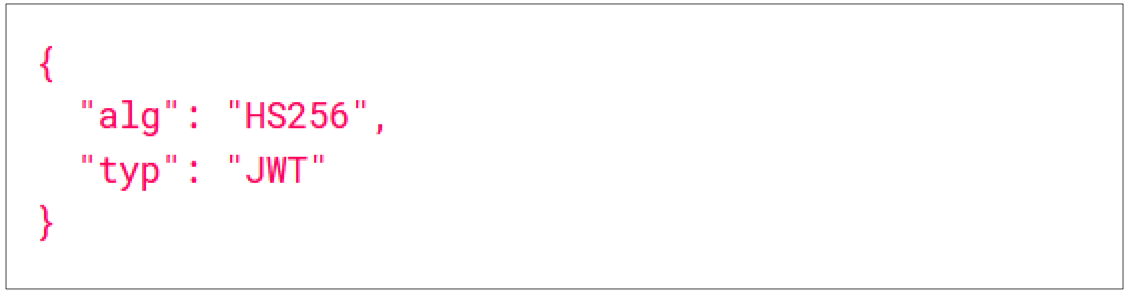
**JSON WEB TOKEN STRUCTURE :**

- Header
- Payload
- Signature

## REST API Security Best Practices

**JSON WEB TOKEN HEADER :**

- Type of the token
- Signing algorithm



```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

# REST API Security Best Practices

## JSON WEB TOKEN PAYLOAD:

- Registered claims:  **iss** (issuer), **exp** (expiration time), **sub** (subject), **aud** (audience)
- Public claims: These can be defined at will by those using JWTs.
- Private claims: These are the custom claims created to share information between parties.

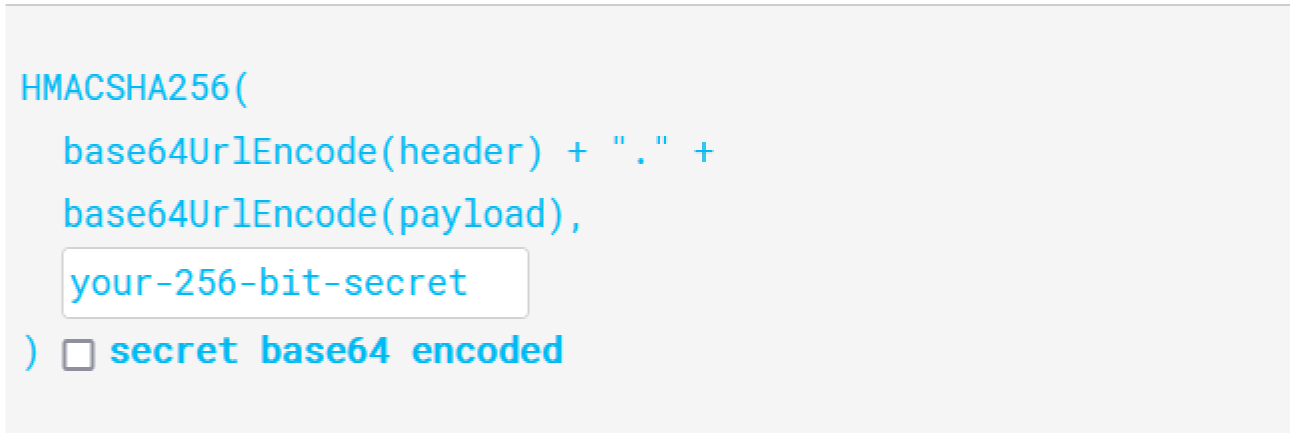
```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022,
  "exp":1576239022
}
```

# REST API Security Best Practices

**JSON WEB TOKEN SIGNATURE**

To create the signature part -

- Take the encoded header
- Take the encoded payload, a secret
- The algorithm specified in the header

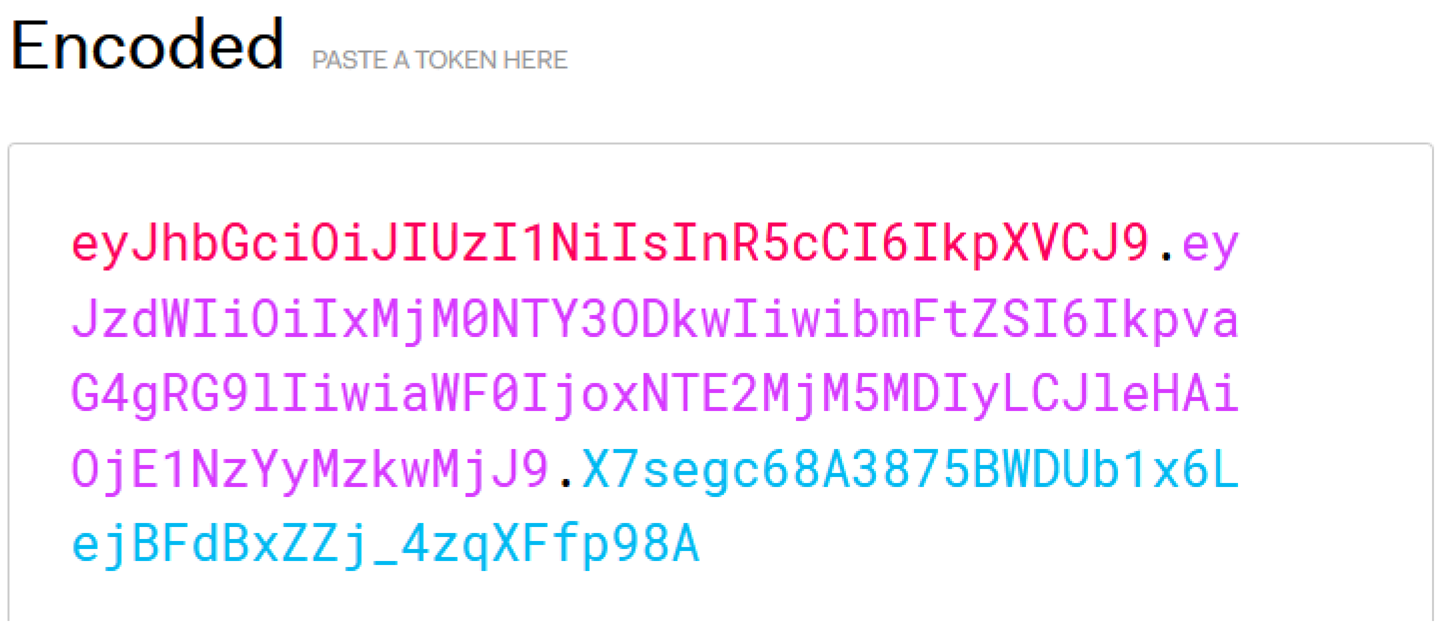


```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

# REST API Security Best Practices

**JSON WEB TOKEN**

**Putting all together**



Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJleHAi
OjE1NzYyMzkwMjJ9.X7segc68A3875BWDUb1x6L
ejBFdBxZZj_4zqXFfp98A
```

## General Questions

1.Why are you interested in this position?

2.How did  you hear about our company?

3. What can you tell us about yourself?

4. How would you describe your work or management style?

5. Do you prefer to work individually or as a team member?

6.How do you stay current with back-end development trends?

7.In your last position, what were your primary job responsibilities?

8.Where do you see yourself professionally in five years?

9.What words would your coworkers use to describe you?

10.What do you find most satisfying about this type of work?

## Questions about experience and background

1. How did you get into coding?

2. What's your greatest strength as a coder?

3. If you were in charge of a tech company, how would you manage its developers?

4. Tell me about a time when someone criticized your work and explain how you responded.

5. How do you deliver negative feedback to members of your development team?

6. Have you ever worked on a team project where you felt you were doing most of the work? How did you manage that?

7. Tell me about the coding accomplishment you’re most proud of?

## Questions about experience and background

8. What's the most challenging decision you've faced in your career?

9. What’s your favorite programming language, and why?

10. What’s your experience with GoTo, and do you prefer structured programming?

## in-depth questions

1. How would you explain the difference between design and architecture?

2. **What are the seven layers in the OSI system model?**

3. **What’s a reverse proxy?**

4. **What's the difference between threads and processes?**

5. **Define and explain these nine server response error codes: 200, 201, 204, 301, 400, 401, 404, 409 and 500.**

## Interview Question

* What is meant by replication in MongoDB?

* How does React work?

*What is Mongoose?

*What do you know about Asynchronous API?

*What is CallBack Hell?

*Explain higher order components(HOC)?

*How does node prevent blocking code?

*What is a ReactDOM?

* **What is Key?**

* **Explain stub in Node.js**

*** What are the features of NodeJS?**