Rest Api best practices includes:

1. REST API MUST ACCEPT AND RESPOND WITH JSON
2. SHOULD GO WITH ERROR STATUS CODES
3. DON'T USE VERBS IN URLS
4. USE PLURAL NOUNS TO NAME A COLLECTION
5. WELL COMPILED DOCUMENTATION
6. RETURN ERROR DETAILS IN THE RESPONSE BODY
7. USE RESOURCE NESTING
8. USE SSL/TLS
9. SECURE YOUR API

Http methods Best Practices:

1. GET: We already know that GET must be safe and idempotent. Web clients know this as well and do not expect side effects when repeating GET requests. Use GET only for retrieving the representation of resources or other information.

2. POST: POST is most commonly used for creating resources. For this purpose, the body of the request must contain the representation of the resource. You can also find it useful in other situations, i.e. when updating several resources at once, on tunneling requests, or when avoiding browser limitations (like query size).

3. PUT: Because POST is not idempotent, avoid using it when updating a resource and use PUT instead, as shown below (this is a common mistake). Though you may return a body in the response, it is not required by the HTTP 1.1 specification.

4. DELETE: As you can understand from its name, this method is used for deleting resources. The interesting thing is that you might not always be able to implement it as idempotent. If DELETE is idempotent, your service has to be aware of all the deleted resources and return the same HTTP status, for example 204 No Content, even if the resource has been previously deleted.

5. OPTIONS: Use it to query the server about the HTTP methods available for resource manipulation or to ping the service. Be aware that the request does not contain a body, but it's free to return the representation of a resource in the response.

6. HEAD: Use this method to look for a particular resource on the server. It usually returns the same headers as GET. The HTTP specification requires that nor request neither response contain a body.

JSON Best Practices:

1. 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  ✓
```

2. 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. ✓
```

3. 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 do
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>"
```

4. 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:

API Request best practices:

1. 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

2. 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 Response best practices:

1. 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.

2. 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.

3. 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 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 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.

- 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.

Web Security best Practices:

1. 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.

2. 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.

3. 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.

4. 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.

5. 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

6. 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.

7. 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.
    - Authorization: Allowing the user to access routes, services, and resources
    - Information Exchange: Way of securely transmitting information between parties.

8. Adopt a DevSecOps Approach
9. Implement a Secure SDLC Management Process
10. Address Open-Source Vulnerabilities
11. Automate
12. Be Aware of Your Own Assets
13. Risk Assessment
14. Security Training for Developers
15. Manage Containers Properly
16. Limit User Access to Data
17. Update and Patch Regularly
18. Ensure Access to Log Data
19. Encrypt Your Data
20. Use Pentesting
21. Ensure Accurate Input Validation
22. Aim for Permanent Fixes

50 Interview Question on Web Back-End Development:

1. What Is CAP Theorem?
2. What REST stands for?

3. What are NoSQL databases? What are the different types of NoSQL databases?
4. What do you understand by NoSQL databases? Explain.
5. What is SQL injection?
6. What is meant by Continuous Integration?
7. How to mitigate the SQL Injection risks?
8. Name some performance testing steps
9. Name the difference between Acceptance Test and Functional Test
10. What are some advantages of using Go?
11. What are the advantages of Web Services?
12. What are the benefits of using Go programming?
13. What does Containerization mean?
14. Why Would You Opt For Microservices Architecture?
15. Name some Performance Testing best practices
16. What Do You Mean By High Availability (HA)?
17. What Is ACID Property Of A System?
18. What Is Sticky Session Load Balancing? What Do You Mean By "Session Affinity"?
19. What are disadvantages of REST web services?
20. What are the DRY and DIE principles?
21. What are the difference between Clustered and a Non-clustered index?
22. What are the differences between Continuous Integration, Continuous Delivery, and Continuous Deployment?
23. What is the difference between Monolithic, SOA and Microservices Architecture?
24. What is the difference between JOIN and UNION?
25. What is the difference between WHERE clause and HAVING clause?
26. Explain what is the API Gateway pattern
27. How does SSL/TLS work?
28. How does B-trees Index work?
29. What do you understand by Distributed Transaction?
30. What is GOD class and why should we avoid it?
31. What is Spike Testing?
32. What is BASE property of a system?
33. What's the difference between Faking, Mocking, and Stubbing?
34. What's the difference between principles YAGNI and KISS?
35. When to use Redis or MongoDB?
36. Why layering your application is important? Provide some bad layering example.
37. Are you familiar with The Twelve-Factor App principles?
38. How would you implement SSO for Microservice Architecture?
39. Name some best practices for better RESTful API design
40. What is your favorite programming language? And why?
41. What do you understand by NoSQL databases? Explain.
42. What is the difference between software architecture and software design?
43. What is JavaScript, and why is it used?
44. What is your approach to debugging?
45. How would you find the most expensive queries in an application?

46. What is the difference between an acceptance test and a functional test?
47. Which sorting algorithm should you use and when?
48. What is the meaning of “high cohesion” and “loose coupling”?
49. Why should you use microservices architecture?
50. Explain the difference between cohesion and coupling?