REST (Representational State Transfer) is an architectural style for designing networked applications. When building RESTful APIs, it's essential to adhere to certain standards and best practices to ensure consistency, scalability, and interoperability. Here are some REST API standards and best practices:

1. Use HTTP Methods:
   * Use HTTP methods (GET, POST, PUT, DELETE, PATCH, etc.) to perform CRUD (Create, Read, Update, Delete) operations on resources.
   * Use GET for retrieving data, POST for creating resources, PUT/PATCH for updating resources, and DELETE for deleting resources.
2. URI Structure:
   * Design meaningful and hierarchical URIs that represent resources.
   * Use nouns (plural) to represent resources, e.g., **/users** for a collection of users.
   * Use sub-resources for related data, e.g., **/users/{id}/orders** to get a user's orders.
   * Avoid using verbs in URIs; let HTTP methods convey the action.
3. Versioning:
   * Include a version number in the API's URI to ensure backward compatibility, e.g., **/v1/users**.
4. Status Codes:
   * Use appropriate HTTP status codes to indicate the outcome of the request (e.g., 200 OK, 201 Created, 400 Bad Request, 404 Not Found).
   * Provide clear and consistent error messages in the response body.
5. Request and Response Formats:
   * Use JSON as the default format for both request and response bodies.
   * Include proper headers to specify the content type (e.g., **Content-Type: application/json**).
6. Pagination and Filtering:
   * Implement pagination for large collections of resources using query parameters like **page** and **per\_page**.
   * Allow filtering, sorting, and searching using query parameters, e.g., **?filter=name&sort=asc**.
7. Authentication and Authorization:
   * Use industry-standard authentication mechanisms like OAuth 2.0 or API keys.
   * Implement role-based access control (RBAC) or permissions to manage authorization.
8. Error Handling:
   * Use consistent error response structures with details about the error (error code, message, and optionally, additional information).
   * Provide human-readable error messages and error codes.
9. Use Hypermedia:
   * Include hypermedia links (HATEOAS - Hypermedia as the Engine of Application State) to guide clients through the API.
   * Hypermedia links can help clients discover available actions dynamically.
10. Idempotent Operations:
    * Ensure that idempotent operations (e.g., PUT, DELETE) can be safely retried without causing unintended side effects.
11. Rate Limiting:
    * Implement rate limiting to prevent abuse of the API and ensure fair usage.
12. Version Control:
    * Use a version control system (e.g., Git) to track changes in your API code.
13. Documentation:
    * Create comprehensive API documentation with examples, usage guidelines, and sample requests.
    * Consider using tools like Swagger or OpenAPI to generate documentation.
14. Testing:
    * Thoroughly test your API using unit tests, integration tests, and real-world scenarios.
    * Set up automated testing and monitoring to detect issues promptly.
15. Security:
    * Implement security best practices to protect against common threats, like SQL injection, XSS, and CSRF attacks.
    * Regularly update and patch dependencies.
16. Versioning:
    * Plan for API versioning to accommodate changes and updates while maintaining backward compatibility.

These standards and best practices help create well-designed and maintainable RESTful APIs that are easier to understand, consume, and extend. Always consider the specific requirements of your project and team when designing your REST API.