Questions for the self-check:

1.Explain the difference between terms: REST and RESTful. What are the six constraints?

REST is a set of architectural principles or constraints that define a lightweight, stateless, and scalable approach for designing networked applications, while RESTful refers to the when a web service keeps to these principles when building.  
The six constraints of REST are:

1.**Stateless** (The server should not store any information about the client's state between requests.)

2.**Client-Server** (REST separates the client and server, enabling them to evolve independently.)

3.**Uniform Interface** (REST enforces a uniform and consistent way to interact with resources, using a limited set of well-defined methods (HTTP methods like GET, POST, PUT, DELETE) and standard resource identifiers (URIs or URLs))

4.**Layered System** (REST allows you to use a layered system architecture where you deploy the APIs on server A, and store data on server B and authenticate requests in Server C, for example. A client cannot ordinarily tell whether it is connected directly to the end server or an intermediary along the way.)

5.**Cacheable** (Responses from the server should be explicitly marked as cacheable or non-cacheable)

6.**Statelessness** (REST encourages the server to be stateless, meaning that each request from the client to the server must contain all the information required to understand and process the request)

2. HTTP Request Methods (the difference) and HTTP Response codes. What is idempotency?  Is HTTP the only protocol supported by the REST?

HTTP is a fundamental protocol for communication on the World Wide Web, and it defines a set of request methods and response codes used for interactions between clients (usually web browsers or applications) and servers.  
methods => POST, GET, PUT, PATCH, and DELETE.

* response codes =>   
  1xx: Informational
* 2xx: Successful (e.g., 200 for OK)
* 3xx: Redirection (e.g., 302 for Found)
* 4xx: Client Errors (e.g., 404 for Not Found)
* 5xx: Server Errors (e.g., 500 for Internal Server Error)

Idempotency in the context of HTTP methods means that if a request can be made multiple times with the same input and the result will be the same. GET, PUT, and DELETE are examples of idempotent methods.

While HTTP is the most used protocol in RESTful web services, it is not the only one. REST is an architectural style that can work with other protocols as well.

3. What are the advantages of statelessness in RESTful services?

* Statelessness simplifies the design of RESTful services and allows them to be highly scalable.
* Stateless services work well with caching mechanisms. Since responses can be cached based on the request, multiple clients can benefit from cached data, reducing the load on the server and improving response times. Stateless design ensures that cached responses can be safely reused for identical requests.
* Clients can interact with stateless services without needing to maintain any specific state information.

4. How can caching be organized in RESTful services?

Caching in RESTful services can be organized using various mechanisms and HTTP headers. (**HTTP Cache-Control Header, HTTP ETag Header, Last-Modified Header**)

5. How can versioning be organized in RESTful services?

1. Versioning through URI Path.

**https://api.example.com/v1/resource**

**https://api.example.com/v2/resource**

1. Versioning through query parameters.

**https://api.example.com/resource?version=v1**

**https://api.example.com/resource?version=v2**

1. Versioning through custom headers.

**X-API-Version: v1**

**X-API-Version: v2**

1. Versioning through content negotiation.

6. What are the best practices of resource naming?

6.1 Use nouns to represent resources

6.2 Use hyphens (-) to improve the readability of URIs

6.3 Avoid using file extensions

6.4 Version APIs

6.5 Use query component to filter URI collection

6.6 Never use CRUD function names in URIs and do not use Verbs

7. What are OpenAPI and Swagger? What implementations/libraries for .NET exist? When would you prefer to generate API docs automatically and when manually?

**OpenAPI** is a specification for describing and documenting RESTful APIs. It was formerly known as Swagger Specification and is often still referred to as "Swagger." OpenAPI provides a standardized way to define API endpoints, their parameters, response formats, and more, making it easier for developers to understand, use, and interact with APIs. **Swagger**, on the other hand, is a set of open-source tools and a framework for building and documenting RESTful APIs using the OpenAPI Specification.

.Net lib example => Nwag ,Swashbuckle  
  
Generate API Docs Automatically: When API is frequently changing

Generate API Docs Manually:Legacy or complex API documentation

8. What is OData? When will you choose to follow it and when not?

OData APIs are typically used for exposing and interacting with data in a structured and consistent manner.It is a standardized protocol for building and consuming RESTful APIs. It's a good choice when you need standardized, queryable, and interoperable access to data. However, it might not be the best fit for every API, especially when simplicity, customization, or non-data-centric operations are key considerations. The decision to use OData should be based on the specific requirements and constraints of the project.

9. What is Richardson Maturity Model? Is it always a good idea to reach the 3rd level of maturity?

The Richardson Maturity Model is a framework proposed by Leonard Richardson to assess the level of maturity of a RESTful API based on its design and use of HTTP principles.

Level 3 is the highest level of REST maturity. At this level, the API includes hypermedia controls in responses, allowing clients to discover and navigate the API's capabilities dynamically.

But it isn’t always good to use, like:

Some clients may not support HATEOAS or may find it challenging to work with. In such cases, Level 3 might not be the best choice.

10. What does pros and cons REST have in comparison with other web API types?

**REST vs. SOAP**:

*Pros of REST*:

1. **Simplicity**: REST APIs are relatively simple and easy to understand, making them accessible to a wide range of developers.
2. **Statelessness**: REST is stateless, which simplifies client-server interactions and supports scalability.
3. **HTTP Standard**: REST leverages the existing HTTP standard, which is widely supported and understood, and it's firewall-friendly.
4. **Flexibility**: REST allows clients to request only the data they need, reducing over-fetching of data.

*Cons of REST*:

1. **Limited Contract**: REST APIs don't provide a formal contract or strong typing, which can make it harder to ensure that clients and servers understand each other.
2. **Lack of Real-Time Data**: REST isn't well-suited for real-time data queries and subscriptions.
3. **Over- or Under-Fetching**: Clients may need to make multiple requests to fetch related data.