1. Use HTTP methods (GET, POST, DELETE, etc) as defined.

2. Ensure URIs are RESTful
    - Short (as possible). This makes them easy to write down or spell or remember.
    - Hackable ‘up the tree’. The user should be able to remove the leaf path and get an expected page back. e.g. http://example.com/cars/alfa-romeos/gt you could remove the gt bit and expect to get back all the alfa-romeos.
    - Meaningful. Describes the resource. I should have a hint at the type of resource I am looking at (a blog post, or a conversation). Ideally I would have a clue about the actual content of the URI (e.g. a uri like uri-design-essay)
    - Predictable. Human-guessable. If your URLs are meaningful they may also be predictable. If your users understand them and can predict what a url for a given resource is then may be able to go ‘straight there’ without having to find a hyperlink on a page. If your URIs are predictable, then your developers will argue less over what should be used for new resource types.
    - Help visualize the site structure. This helps make them more ‘predictable’.
    - Readable.
    - Nouns, not verbs.
    - Query args (everything after the ?) are used on querying/searching resources (exclusively). They contain data the affects the query.
    - Consistent. If you use extensions, do not use .html in one location and .htm in another. Consistent patterns make URIs more predictable.
    - Stateless.
    - Return a representation (e.g. XML or json) based on the request headers, like Accept and Accept-Language rather than a change in the URI.
    - Tied to a resource. Permanent. The URI will continue to work while the resource exists, and despite the resource potentially changing over time.
    - Report canonical URIs. If you have two different URIs for the same resource, ensure you put the canonical URL in the response.
    - Follows the digging-deeper-path-and-backspace convention. URI path can be used like a backspace.
    - Uses name1=value1;name2=value2 (aka matrix parameters) when filtering collections of resources.

3. Stateless. The web server does not store any state about the client application/session state. The session is stored on the client. The server is stateless means that every server can service any client at any time, there is no session affinity or sticky sessions. The relevant session information is stored on the client and passed to the server as needed.

4. Responses defined as cacheable.