A brief history of REST

- REST (Representational State Transfer) was developed in 1999 with the publication of the "Hypertext Transfer Protocol - HTTP/1.1" request for comments (RFC) document.
- The author of the RFC, Roy Fielding, defined a set of principles based on the HTTP and URI standards in his dissertation called "Architectural Styles and the Design of Network-based Software Architectures."

RESTful API

- A RESTful API follows the REST architecture and uses HTTP methods to expose resources and perform operations on those resources.
- Resources are the entities that the API exposes. Examples include user accounts, articles, and images.
- Endpoints are the URLs that the API uses to expose resources.

RESTful API

- RESTful APIs use standard HTTP methods such as GET, POST, PUT, DELETE to perform operations on resources.
- RESTful APIs are stateless, meaning that each request contains all the necessary information to complete the request.
- RESTful APIs return responses in a standard format such as JSON or XML, that can be parsed and used by the client.

RESTful vs SOAP

Feature	SOAP	REST
Protocol or architectural style	Protocol	Architectural style
Format	XML	HTTP requests and responses
Transportation	HTTP and other protocols	HTTP
Error handling	Built-in error handling	HTTP status codes
Advanced Features	Encryption, Authentication	Resource-based interface
Popularity	Less popular than REST	More popular than SOAP

REST Principles

To create a RESTful application using HTTP and URI standards, you should follow these principles:

1. All items are resources that can be identified by a unique identifier (URI).
2. Use standard HTTP methods for resource manipulation.
3. Resources can have multiple representations, such as text or image.
4. Communication should be stateless, meaning the server does not need to retain information about previous client requests.

Principle 1 – Everything Is A Resource

- Everything on the Internet can be considered a resource.
- Resources can be identified by a unique identifier (URI).
- Resources can be in various formats and have corresponding content types, such as image/jpeg or text/html.
- Data is not tied to a physical file, but can be represented in different ways through the use of content types.

Principle 2 – Each Resource Is Identifiable By A Unique Identifier

- Resources on the Internet should be uniquely identifiable by a URI.
- URIs can be in a human-readable format, which can help reduce the risk of logical errors in programs.
- The use of human-readable URIs can make data self-descriptive and facilitate further development.

Principle 2 – Each Resource Is Identifiable By A Unique Identifier

Here are a few sample examples of such URIs:

- http://www.mydatastore.com/images/vacation/2014/summer
- http://www.mydatastore.com/videos/vacation/2014/winter

These human-readable URIs expose different types of resources in a straightforward manner. In the example, it is quite clear that the resource types are as follows:

• Images, • Videos, • XML/JSON documents, • Some kinds of binary archive documents

Principle 3 – Use The Standard Http Methods

The native HTTP protocol (RFC 2616) defines eight actions, also known as verbs: • GET • POST • PUT • DELETE • HEAD • OPTIONS • TRACE • CONNECT

- The HTTP verbs GET, POST, PUT, and DELETE are commonly used for resource manipulation.
- These verbs correspond to the CRUD actions (Create, Read, Update, and Delete) in SQL.

Principle 3 – Use The Standard Http Methods

- If REST principles are followed, these HTTP verbs should be used appropriately.

HTTP verb	Action	Response status code
GET	Request a resource	"200 OK" if the resource exists, "404 Not Found" if it does not exist, and "500 Internal Server Error" for other errors
PUT	Create or update a resource	"201 CREATED" if a new resource is created, "200 OK" if updated, and "500 Internal Server Error" for other errors
POST	Update a resource	"200 OK" if the resource has been updated successfully, "404 Not Found" if the resource to be updated does not exist, and "500 Internal Server Error" for other errors
DELETE	Delete a resource	"200 OK" if the resource has been deleted successfully, "404 Not Found" if the resource to be deleted does not exist, and "500 Internal Server Error" for other errors

Principle 4 – Resources Can Have Multiple Representations

- Resources can be represented in different forms.
- REST-enabled endpoints should be able to use different formats as long as they are supported.
- The example given shows that a resource can be posted or requested in different representations, such as XML or JSON.

Principle 5 – communicate statelessly

- HTTP request resource manipulation operations should be atomic and modify resources in isolation.
- Complete resource updates should be sent, rather than partial updates.
- RESTful applications should be stateless, meaning that the server does not need to retain information about previous client requests.
- Statelessness allows for changes to be made to the server infrastructure without affecting the caller.
- It is possible to keep some state in a RESTful way, such as by including it in the URI.

Principle 5 – communicate statelessly

- REST has become popular in recent years.
- SOAP web services often have complex specifications and rules for interoperability, which can make them cumbersome to use.
- REST simplifies data transfer by introducing the concept of resources and standard methods for manipulating them.

The REST Goals

- Separation of the representation and the resource
- Visibility
- Reliability
- Scalability
- Performance

Separation Of The Representation And The Resource

- A resource is a set of information that can have multiple representations.
- The state of a resource is atomic, meaning it cannot be partially updated.
- The caller specifies the content-type header of the HTTP request.
- The server application should handle the representation and return the appropriate HTTP status code.

Separation Of The Representation And The Resource

- HTTP 200 OK in the case of success
- HTTP 400 Bad request if a unsupported content type is requested, or for any
other invalid request
- HTTP 500 Internal Server error when something unexpected happens during the request processing

Separation Of The Representation And The Resource Example 1

```
GET /data/balance/22082014 HTTP/1.1 Host: my-computer-hostname
Accept: text/xml

HTTP/1.1 200 OK
Content-Type: text/xml Content-Length: 140

<?xml version="1.0" encoding="utf-8"?>
<balance date="22082014">
<Item>Sample item</Item>
<price currency="EUR">100</price>
</balance>
```

Separation Of The Representation And The Resource Example 2

```
GET /data/balance/22082014 HTTP/1.1 Host: my-computer-hostname
Accept: application/json

HTTP/1.1 200 OK
Content-Type: application/json Content-Length: 120

{
  "balance": {
    "date": "22082014",
    "Item": "Sample item",
    "price": {
      "-currency": "EUR",
      "#text": "100"
      }
    }
}
```

Visibility

- REST is designed to be visible and simple.
- Visibility means that all aspects of the service should be self-descriptive and follow the natural HTTP language.
- Monitoring applications can observe the HTTP communication between the REST service and the caller to understand the behavior of the application.
- The statelessness and atomicity of requests and responses make it easy to understand if anything has gone wrong.

Reliability

- HTTP methods can be classified as safe or idempotent.
- A safe method is one that does not modify the state of a resource or cause any side effects when requested.
- An idempotent method produces the same response no matter how many times it is requested.
- The HTTP methods GET, PUT, and DELETE are idempotent, while POST is not.
- GET is the only safe method for the above.

Scalability and performance

- Statelessness is important for applications that need to scale easily.
- Scaling an application with state can be difficult, especially when zero downtime is required.
- Statelessness allows for hardware to be added to a load balancer without the need for nodes to sync with each other.

Scalability and performance

- Scalability is about being able to serve all clients in a reasonable amount of time and preventing Denial of Service attacks.
- Scalability should not be confused with performance, which is the time required to process a single request.
- Asynchronous, non-blocking architectures and event-driven design can improve the scalability and performance of an application.

Summary

- The infrastructure for developing and distributing RESTful applications already exists.
- RESTful applications rely heavily on existing web infrastructure.
- Libraries are available for various platforms to ease the development of RESTful applications.