Lesson 9
REST Architectural Style and RESTful APIs

REST (Representational State Transfer)

- CoAP, HTTP and XMPP use REST architectural style
- REST style developed by W3C Technical Architecture Group (TAG)
- Simpler than SOAP and WSDL (Web Services Description Language)

Representation of Each Application State

- Contains links that may be used the next time when the client selects those links and initiates a new state-transition
- For example, a state represented by URI of a resource of a resource type at a resource repository at a server
- For example, get ( ) method for operations using the state

REST architectural style

- A coordinated set of practices and constraints
- These practices and constraints used during design of software components in a distributed hypermedia

Stateless, Client-Server, Cacheable communication

- REST design depends on the characteristics of stateless, client-server, cacheable communication
- Client or server or intermediate systems can cache the responses

Definable Cacheability

- Clients or intermediate layers can cache the responses
- Server Responses must therefore be implicitly or explicitly, define themselves as cacheable, or not

Scalability

- Ability to support greater number of interactions among components and greater number of components

Resources

- Resources themselves are conceptually separate from the representations that are returned to the client
- For example, the server may send data HTML, XML, TLV or JSON from its database which can have other distinct internal representation at a connected database serve

REST Features

- use of specific practices,
- Client server interactions
- Services have characteristics of performance, and creation of scalable web objects and services

REST Feature of usages of Layered System

- A client can connect through intermediate (proxy or firewall or Gateway intermediary server) layers
- REST enables intermediate layer processing by constraining messages to be self-descriptive interaction.

REST Feature of enabling usage of Intermediate system(s)

- Client may not ordinarily be made aware whether it is connected directly to the end server
- enables assistance in transcoding and different protocols at two ends.
- Improves performance when using bigger scales and shared caches.

REST Feature of Simplicity of Interfaces

1. Modifiability of components enables the changes according to the needs, even while an App is running,
2. Visibility of communication between components by service agents,

REST Feature of Simplicity of Interfaces

3. Portability of components by moving the objects, Object means program code with the data
4. Reliability is the resistance to failure at the system level in the presence of failures within data elements, components or connectors,

Formal **REST constraint of Uniform Interfaces**

- **Identification** of individual resources, identified in requests, for example, using URIs in web-based REST systems,
- **Separation of concerns** which means Client–server interactions are such that a uniform interface separates clients from servers.

Separation of Concern architectural property

- That means– how does data store at server is not a concern for client and
- Client components can port on other objects
- user interface and user state are of no concern of Server.

REST's client–server separation of concerns

- Simplifies the component implementation
- Increases the scalability of pure server components
- Reduces the complexity of connector semantics, and
- improves the effectiveness of performance tuning

Formal **REST constraint of Uniform Interfaces**

- **Client holding representation** of a resource including any metadata attached then it means it has enough information to modify or delete the resource.
- **Manipulation of resources** through **representations** sent to the client,

Formal **REST constraint of Uniform Interfaces**

- **Stateless client–server communication** which means that constraints are such that client context does not store at the server, each request from client is considered independent of previous one

Stateless Communication

- Stateless means a transfer is not connected with the previous transfers (Meta data in header can define the state (for example sequence number) to link two independent data transfers

Client Data

- Itself free to include the context in the data, a session state holds at the client itself
- Client has information, using that the server serves the request.

Client Session State

- Can transfer to the server or another server
- It can be used for authentication
- Can also be used for data persistency for a period or at the database server when required

Option of transfer of executable code

- Code such as JavaScript on demand (optional)– (Code may also include JavaApplets and client-side scripts)

Self-descriptive messaging

- Each message is and includes enough information to describe how to process the message, for example, responses may indicate internal media type (MIME Type, for example, jpg)

State Transitions at Client

- Through actions that are dynamically identified within server by hyperlinks within the hypertext

Entry points in an Application

- Does not assume that any particular action available for any particular resources beyond those described in representations previously received from the server

RESTful

- RESTful When all interactions used in the Applications conform fully to the REST constraints then that are called RESTful.

RESTful APIs

- Comply with these constraints, and thus conforming to the REST architectural style
- Web services with RESTful APIs adhere to the REST architectural constraints

Summary

We learnt

- REST design depends on the characteristics of stateless, client-server, cacheable communication
- Uniform Interfaces: Identification, Separation of concerns, Client–server interactions,

Summary

We learnt

- Client holding representation of a resource including any metadata
- Manipulation of resources through representations sent to the client,
- Stateless client–server communication
- RESTful APIs complies with REST practices and constraints

Summary

We learnt

- CoAP, HTTP, XMPP REST constraints and Communication management functions
- HTTP enables access by GET, POST, PUT and DELETE methods for the resources and building web-services

End of Lesson 9 on REST Architectural Style and RESTful APIs