Distributed & Mobile Computing

IS384

Web Services

Web services are a way for applications to **communicate** with each other over the internet.

They are based on the idea of using standard protocols to exchange data between **different** systems.

This makes them very flexible and powerful way to build **distributed** applications.

Types of Web Services

There are three main types of web services: **XML-RPC**, **SOAP**, and **RESTful**.

- XML-RPC: is a simple but powerful protocol for exchanging data between applications. It is based on the XML format and uses a simple request-response pattern.

Types of Web Services Cont...

- **SOAP**: is a more complex protocol that provides more features than XML-RPC. It is based on the XML format and uses a web services description language (WSDL) to describe the services that are available.
- **RESTful**: RESTful web services are based on the Representational State Transfer (REST) architectural style. They are designed to be lightweight and easy to use.

XML-RPC

XML-RPC (XML Remote Procedure Call) is a protocol used for remote procedure calls using XML as a data format and HTTP as a transport mechanism.

It enables software applications running on different platforms and written in different programming languages to communicate and invoke procedures across a network.

Basic Structure of XML-RPC:

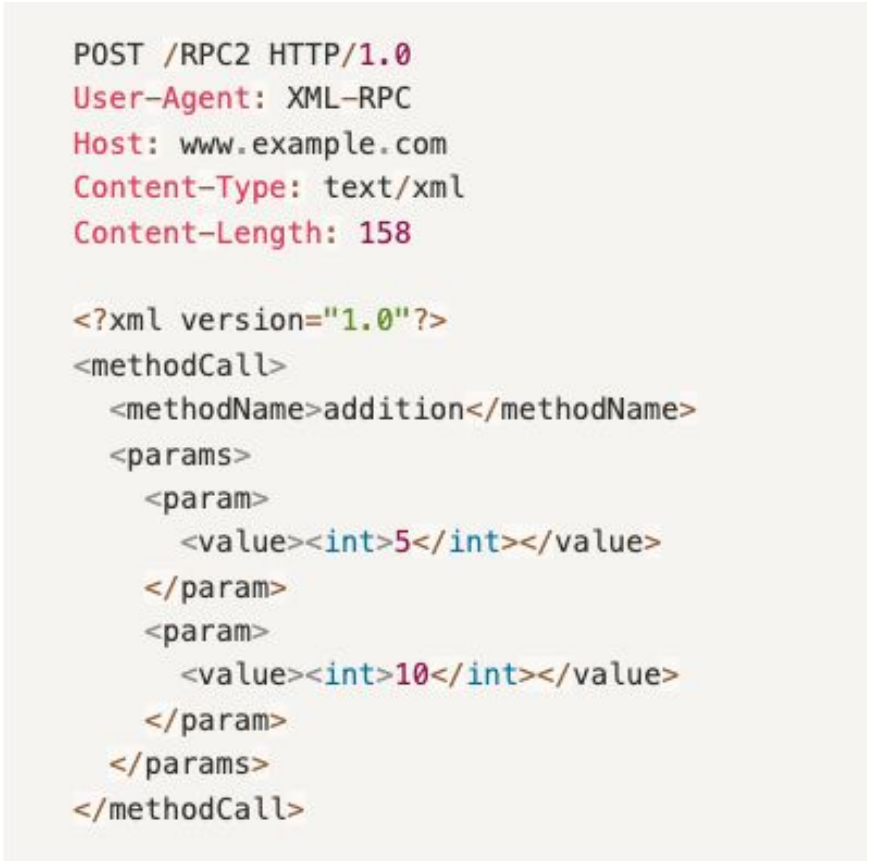
- XML-RPC messages are structured using XML to represent data being sent between a client and a server.

- The communication involves a client making requests to a server using HTTP POST requests with XML-RPC-encoded data.

XML-RPC Message Components:

- **Method Calls**: Represent the remote procedure calls initiated by the client.
- **Parameters**: Data sent along with the method calls, encoded as XML elements.
- **Responses**: Server's reply to the method calls, encoded as XML.

Example of XML-RPC Message:



```
POST /RPC2 HTTP/1.0
User-Agent: XML-RPC
Host: www.example.com
Content-Type: text/xml
Content-Length: 158

<?xml version="1.0"?>
<methodCall>
  <methodName>addition</methodName>
  <params>
    <param>
      <value><int>5</int></value>
    </param>
    <param>
      <value><int>10</int></value>
    </param>
  </params>
</methodCall>
```

Server Response (XML-RPC format):

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><int>15</int></value>
    </param>
  </params>
</methodResponse>
```

XML-RPC Message Explanation...

- The client sends an XML-RPC request via an HTTP POST request to the server specifying the method name 'addition' and passing two integer parameters (5 and 10) within <params>.

- The server receives the request, executes the 'addition' method, and responds with an XML-RPC message containing the sum (15) as the result.

RESTful Web Services

- REST is an architectural style that focuses on designing networked applications by utilizing the existing features of the HTTP protocol.
- It emphasizes a stateless client-server communication model with cacheability, uniform interfaces, and layered architecture.

Key Concepts of REST:

A. **Resources**: In REST, everything is considered a resource identified by a unique URI. Resources can be objects, data, or services.

B. **HTTP Methods (Verbs)**: RESTful services utilize standard HTTP methods for performing actions on resources:
   - *GET*: Retrieve resource representation.
   - *POST*: Create a new resource.
   - *PUT*: Update an existing resource.
   - *DELETE*: Delete a resource.

Key Concepts of REST Cont...

C. **Representations**: Resources are represented in different formats such as **JSON**, XML, or HTML.

Example of a RESTful Web Service:

Suppose we're developing a simple RESTful service to manage a collection of books.

- **Resource (Book):** A book resource identified by a unique URI.
- **HTTP Methods Used:** *GET, POST, PUT, DELETE.*

Example API Endpoints:

1. Retrieve All Books (GET Request):
    - `GET /books` - Fetches a list of all books.

Example of a RESTful Web Service:

2. Retrieve a Specific Book (GET Request):

- `GET /books/{id}` - Fetches details of a specific book identified by {id}.

3. Add a New Book (POST Request):

- **POST /books** - Creates a new book. Request body contains book details.

Example of a RESTful Web Service:

4. Update a Book (PUT Request):

- **PUT /books/{id}** - Updates details of the book identified by {id}. Request body contains updated book information.

5. Delete a Book (DELETE Request):

- **DELETE /books/{id}** - Deletes the book identified by {id}.

Data serialization and standard serialization formats e.g. JSON, XML, and YAML

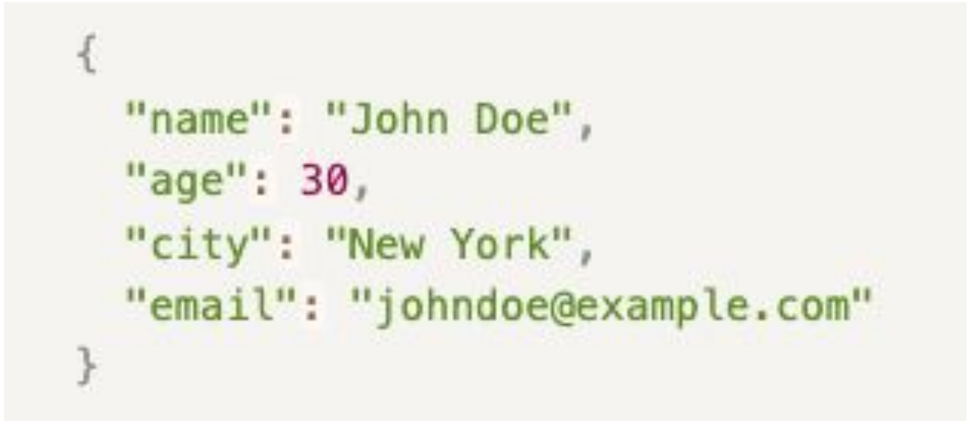
- Data serialization is the process of converting complex data structures or objects into a format suitable for storage or transmission.

- **Purpose**: Facilitates data interchange between different systems, languages, or platforms.

Serialization Formats:

a. JSON (JavaScript Object Notation):

- **Structure**: Lightweight, human-readable format based on key-value pairs, arrays, and nested structures.



```
{
  "name": "John Doe",
  "age": 30,
  "city": "New York",
  "email": "johndoe@example.com"
}
```

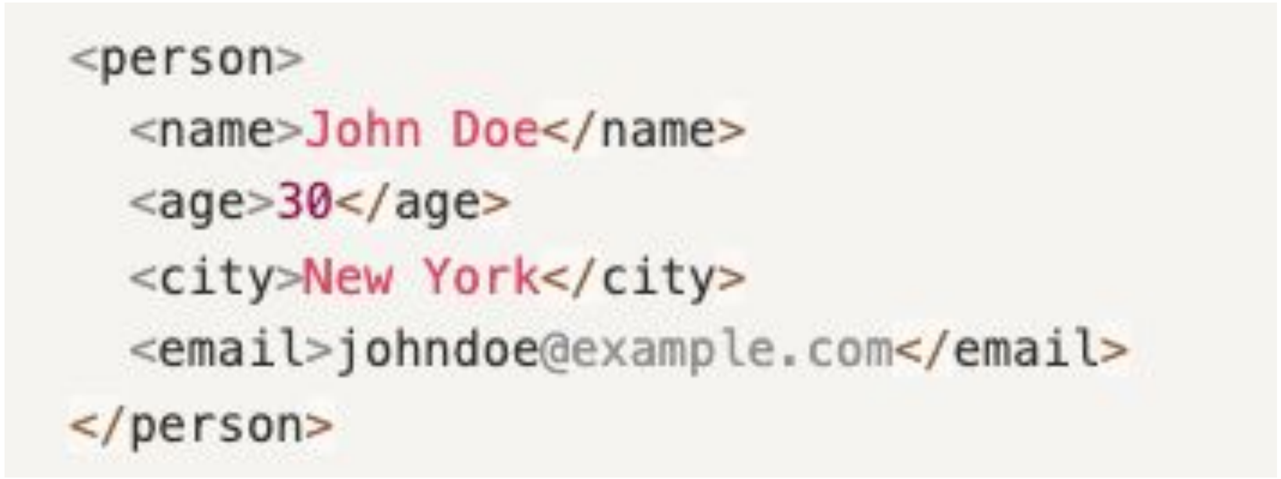
Serialization Formats Cont...

Usage: Widely used in web applications, APIs, configuration files due to simplicity and ease of parsing.

b. XML (eXtensible Markup Language):

- **Structure**: Markup language using tags to define data structure and hierarchy.

Serialization Formats Cont...



```xml
<person>
  <name>John Doe</name>
  <age>30</age>
  <city>New York</city>
  <email>johndoe@example.com</email>
</person>
```

Usage: Commonly used in web services, document storage, and configuration.

Use Cases:

- JSON: API data exchange, web services.
- XML: Document storage, data exchange between heterogeneous systems.
- YAML: Configuration files, data serialization in scripting languages.

Security

Transport and Application Level Security

Transport and application level security are two different strategies for securing data transmission between applications. Both play crucial roles in web services security.

Transport Level Security (TLS):

- **Purpose:** Protects data in transit between applications.
- **Method:** Encrypts the entire communication channel using symmetric key cryptography and a Message Authentication Code (MAC) for data integrity.
- **Example:** HTTPS uses TLS to secure communication between web browsers and servers.

Advantages:

- Provides strong confidentiality and integrity protection.
- Transparent to applications, requiring no code changes.
- Widely supported by most applications and platforms.

Disadvantages:

- May impact performance due to encryption overhead.
- Requires configuration of certificates and key management.

Application Level Security (ALS):

- **Purpose:** Protects specific application data and functionalities.
- **Method:** Implements security mechanisms within the application itself, such as encryption, digital signatures, and access control.
- **Examples:**
  - OAuth: Securely authorizes access to user data.
  - JSON Web Token (JWT): Securely transmits user information between applications.
  - SAML: Provides single sign-on functionality.

Application Level Security (ALS):

- **Advantages:**
    - Provides granular control over application-specific security needs.
    - More flexible and adaptable to specific requirements.
- **Disadvantages:**
    - Requires development effort to implement and maintain.
    - May not be as widely supported as TLS.

Authentication Protocols

Authentication protocols are responsible for verifying the identity of users or applications attempting to access a service. They are essential for preventing unauthorized access and ensuring data security.

Common Authentication Protocols:

- **HTTP Basic Authentication:** Simple username and password based authentication.
- **OAuth:** Securely grants access to user data on third-party applications without revealing passwords.
- **OpenID Connect:** Provides single sign-on functionality using OAuth and JWT.
- **Kerberos:** Secure network authentication protocol based on shared secret keys and trusted third-party authentication server (KDC).

Digital Certificates

Digital certificates are electronic documents that bind a public key to the identity of an individual or entity. They are used to establish trust and secure communication channels.

Components of a Digital Certificate:

- **Subject**: The identity of the entity owning the certificate.
- **Issuer**: The trusted authority that issued the certificate.
- **Public key**: Used for encrypting data sent to the certificate owner.
- **Validity period**: The time frame during which the certificate is valid.
- **Digital signature**: Ensures the integrity and authenticity of the certificate.

Uses of Digital Certificates:

- Secure communication channels (HTTPS, TLS)
- Secure email (S/MIME)
- Code signing and software distribution
- Digital signatures for documents and transactions

Transactions, concurrency and replication controls:

Transaction:

Transaction is a collection of related **Read** and **Write** operations used to perform some related task.

Example:

```
R(A)
A= A+500
W(A)
R(B)
B=B-500
W(B)
commit
```

Transaction Cont..

Where R(A) is a read operation to read a data item A, W(A) is a write operation to write a data item A and commit is a commit operation to commit the transaction.

ACID Properties of Transactions:

- **Atomicity**: All operations in a transaction either succeed or fail together.
- **Consistency**: The transaction transforms the database from one consistent state to another.
- **Isolation**: Concurrent transactions are isolated from each other's effects.
- **Durability**: The effects of a committed transaction are permanent.

Atomicity:

Definition: Atomicity ensures that a transaction is indivisible and is treated as a single unit of work.

Principle: Either all operations within a transaction are executed successfully, or none of them are executed at all.

Example: Consider a bank transfer involving debit from one account and credit to another. Atomicity ensures that if the debit succeeds and the credit fails (or vice versa), the entire transaction is rolled back to maintain consistency.

Consistency:

Definition: Consistency ensures that a database moves from one consistent state to another after a successful transaction.

Principle: Transactions bring the database from one valid state to another valid state, maintaining all integrity constraints, triggers, and business rules.

Example: If an operation violates a constraint (e.g., foreign key constraint), the entire transaction is rolled back, preventing any inconsistent state in the database.

Isolation:

Definition: Isolation ensures that concurrent execution of transactions does not lead to interference or inconsistency.

Principle: Transactions execute independently of each other, and their intermediate states are hidden from other transactions until they are committed.

Example: If multiple transactions are modifying the same data simultaneously, isolation ensures that each transaction sees a consistent snapshot of the data without interference from other transactions.

Durability

Definition: Durability ensures that the changes made by committed transactions persist even in the event of system failure or crash.

Principle: Once a transaction is committed, its changes are permanently stored in the database and remain intact even in the face of system failures.

Example: After a successful transaction, even if the system crashes, the changes made by that transaction are durable and will be reflected in the database when the system is recovered.