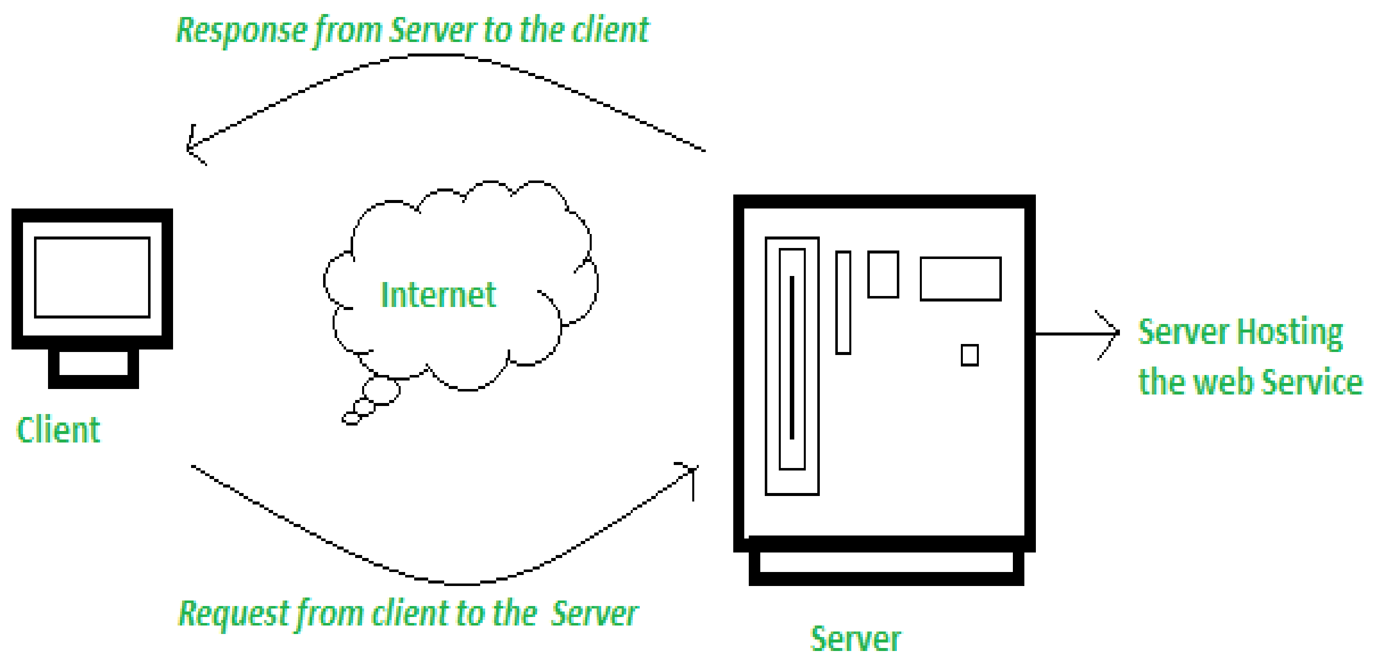
Introduction to Web Services

What are Web Services ?

- A **Web Service** is a standardized way of integrating web-based applications using open standards like
  - APIs
  - JSON
  - SOAP
  - REST
- They enable communication between different software applications running on various platforms and technologies over the internet.


Response from Server to the client
Internet
Client
Server Hosting the web Service
Request from client to the Server
Server

Characteristics of Web Services

- Interoperability : Allows applications written in different programming languages to communicate seamlessly.
- Standardized Protocols: Use protocols like HTTP, XML, JSON, and SOAP for communication.
- Platform Independence: Applications on different platforms (Windows, Linux, macOS) can interact with each other.
- Stateless Communication: Each interaction between client and server is independent.
- Reusability : Services are modular and can be reused across multiple applications.

Purpose of Web Services

Application Integration: Connect different systems and applications within or across organizations.

Data Sharing: Facilitate data exchange between different applications, such as a mobile app fetching weather data.

Automation: Enable machine-to-machine interactions for automated processes, such as IoT devices communicating with a central server.

Example of Web Services

Weather API : Fetch current and forecasted weather data

Payment Gateway API : Process online transactions

Social Media APIs : Post updates or fetch data from platforms like Twitter and Facebook

Key Components of Web Services

Service Provider – The system of offering the service

Service Consumer – The system consuming the service

Service Registry - A directory where services are published for discovery

APIs

- What is an API ?
  - An **API (Application Programming Interface)** is a set of protocols, tools, and definitions for building application software. APIs serve as an intermediary between applications, enabling them to communicate.
- **How APIs Work**
  - **Endpoint**: A specific URL where the API can be accessed.
  - **Request**: The consumer sends a request to the API endpoint.
  - **Response**: The API returns the requested data or action result in a defined format (e.g., JSON, XML).

JSON (**JavaScript Object Notation)**

- **Purpose**: A lightweight data format used for data interchange between systems.
- **Characteristics**:
    - Easy for humans to read and write.
    - Easy for machines to parse and generate.
    - Language-independent, though inspired by JavaScript.
- **Where is it Used?**
    - Web APIs.
    - Configuration files.
    - Data storage in NoSQL databases.

JSON Syntax

- **Objects**: Represented by curly braces { }, containing key-value pairs.
- **Arrays**: Represented by square brackets [ ], containing ordered data.
- **Data Types**: Strings, numbers, booleans, arrays, objects, and null.

- Keys are strings enclosed in double quotes.Values can be strings, numbers, arrays, booleans, objects, or null.

Advantages of JSON

- Human-readable and simple structure.
- Lightweight and compact.
- Widely supported across programming languages.
- Ideal for real-time communication between web services.

Tools for JSON

- **JSON Editors and Viewers**
  - **JSONLint**: Online tool for validation and formatting.
  - **VS Code**: JSON extensions for formatting and validation.
  - **Notepad++**: Use the JSON Viewer plugin for editing.
- **2. JSON Formatters**
  - **jsonformatter.org**: Beautify and validate JSON.
  - **Postman**: View formatted JSON responses from APIs.
- **3. JSON Data Generators**
  - **Mockaroo**: Create realistic JSON test data.
  - **JSON Generator**: Generate structured JSON programmatically.
- **4. Parsing and Manipulation**
  - **Python**: json module for parsing and creating JSON.
  - **JavaScript**: JSON.parse() and JSON.stringify().
  - **Java**: Jackson and Gson libraries.
- **5. Validation and Conversion**
  - **JSON Schema**: Validate JSON structure against schemas.
  - **JSON2CSV**: Convert JSON to CSV or vice versa.
  - **Quicktype**: Generate code models from JSON.
- **6. Debugging Tools**
  - **Postman**: Test and debug JSON in API responses.

XML - eXtensible Markup Language.

- Designed to store and transport data in a structured, human-readable, and machine-readable format.
- **Characteristics**:
  - Tags define the structure and meaning of data.
  - Self-descriptive, extensible, and hierarchical.
  - Platform and language-independent.
- **Where is it Used?**
  - Data exchange between systems.
  - Configuration files.
  - Web services (e.g., SOAP).

XML Syntax - XML Structure and Syntax

- **Key Features**:**Elements**: Defined by opening <tag> and closing </tag> (e.g., <name>Test</name>).
- **Attributes**: Key-value pairs within tags (e.g., <employee id="101">).
- **Hierarchy**: XML is structured like a tree.
- **Well-Formed Rules**:
  - Every opening tag must have a closing tag.
  - Tags are case-sensitive.
  - Properly nested tags are required.

```
<employee id="101">
        <name>Alice</name>
        <age>30</age>
        <skills>
           <skill>Java</skill>
           <skill>Python</skill>
        </skills>
</employee>
```

Advantages and Use Cases of XML

- **Advantages**:
  - Human-readable and flexible.
  - Self-descriptive data structure.
  - Extensible and supports namespaces.
  - Ideal for data validation with DTD or XSD.
- **Common Use Cases**:**Web Services**:
  - SOAP, RSS feeds.
- **Configuration**: Build files in Maven, Ant.
- **Documents**: XHTML, SVG.

XML Tools

- **Editors**: Notepad++, VS Code, XMLSpy.
- **Parsing Libraries**:
  - Python: xml.etree.ElementTree, lxml.
  - Java: DOM, SAX parsers.
  - JavaScript: DOMParser.

Comparison between XML and JSON

Feature	XML	JSON
Definition	eXtensible Markup Language	JavaScript Object Notation
Purpose	Markup language for structured data storage	Data-interchange format
Syntax	Uses tags (<tag> and </tag>)	Uses key-value pairs ("key": "value")
Verbosity	More verbose due to tags	Less verbose and compact
Data Types	Treats all data as text	Supports native types (e.g., string, number, boolean)
Schema Validation	Supports DTD and XSD	No built-in schema validation
Readability	Human-readable but more complex	Human-readable and simpler
Use Cases	Documents, configurations, SOAP APIs	Web APIs, configurations, real-time data
Parsing Speed	Slower due to XML's complexity	Faster due to simpler syntax
Attributes	Supports attributes for metadata	No attributes; uses nested objects instead
Namespaces	Fully supports namespaces	Does not support namespaces
Supported Formats	Only XML	JSON, XML, YAML, etc.

XML Example , JSON Example

```xml
<employee id="101">
  <name>Alice</name>
  <age>30</age>
  <skills>
    <skill>Java</skill>
    <skill>Python</skill>
  </skills>
</employee>
```

```json
{
  "employee": {
    "id": 101,
    "name": "Alice",
    "age": 30,
    "skills": ["Java", "Python"]
  }
}
```

REST (Representational State Transfer)

- REST is an architectural style for designing networked applications. It uses standard HTTP methods and treats all resources as nouns accessible via URLs.
- **Key Features**
  - **Statelessness**: Each request contains all the information needed to process it.
  - **Cacheable**: Responses can be cached to improve performance.
  - **Layered System**: Components are loosely coupled, improving scalability.
- **Common HTTP Methods in REST**
  1. **GET**: Retrieve data (e.g., fetch user details).
  2. **POST**: Create new data (e.g., register a new user).
  3. **PUT**: Update existing data (e.g., update user information).
  4. **DELETE**: Remove data (e.g., delete a user).

Characteristics of REST

Statelessness: No client context is stored on the server.

Resource-Based: Resources are identified by URIs.

Uniform Interface: Simplifies the interaction between client and server.

Cacheable: Responses can be cached to improve performance.

Layered System: Supports scalability through multiple layers.

Principles of REST

- **Client-Server**: Separation of concerns.
- **Stateless**: Each request is independent.
- **Cacheable**: Responses must indicate whether they are cacheable.
- **Uniform Interface**: Consistent structure for requests and responses.
- **Layered System**: Intermediary servers can improve scalability.

HTTP Methods in REST

HTTP Method	**Action**	**Example**
GET	Read data	/users
POST	Create	/users
PUT	Update	/users/1
DELETE	Delete	/users/1

REST API Structure

- **Content:Base URL**: https://api.example.com
- **Endpoints**: /users, /products
- **Parameters**: Query (?id=1) and Path (/users/1)
- **Headers**: Metadata like Authorization, Content-Type.

REST Request and Response Example

Request –

- GET https://api.example.com/users/1

Response –

```
{
"id": 1,
"name": “test”,
"email": “test@example.com”
}
```

Advantages of REST

- **Content**: Simplicity: Easy to implement and use.
- **Performance**: Lightweight and faster with JSON.
- **Scalability**: Stateless communication allows horizontal scaling.
- **Flexibility**: Works with multiple data formats (JSON, XML, etc.).

Limitations of REST

- Statelessness can increase overhead.
- Not ideal for complex operations requiring multiple calls.
- Lack of built-in security (requires HTTPS, tokens, etc.).

SOAP (Simple Object Access Protocol)

- SOAP stands for Simple Object Access Protocol.
- A protocol for exchanging structured information in web services.
- Uses XML for message formatting.
- Relies on application layer protocols like HTTP and SMTP.

Key Features of SOAP

- Platform and language-independent.
- Based on XML, ensuring interoperability.
- Built-in error handling.
- Extensible through standards like WS-Security.
- Supports ACID transactions.

SOAP Message Structure

- **Envelope**: Defines the start and end of the message.
- **Header**: Optional, contains metadata.
- **Body**: Contains the actual message content.
- **Fault**: Optional, used for error handling.

SOAP Message

```
<soap:Envelope>
    <soap:Header>...</soap:Header>
    <soap:Body>
    <Response>...</Response>
    </soap:Body>
</soap:Envelope>
```

SOAP Use Cases

- **Banking and Finance**: Secure transactions.
- **Enterprise Applications**: Robust error handling.
- **Healthcare**: Data exchange (HL7 standard).
- **Government Services**: Reliable messaging.

SOAP Advantages and Disadvantages

Advantages	Disadvantages
High security with WS-Security.	Verbose due to XML format.
Reliable messaging and ACID compliance.	Slower compared to REST.
Extensible and highly standardized.	Complex setup and overhead.
Works well in distributed environments.	Works well in distributed environments.

SOAP Example

Request

```
<soap:Envelope>
   <soap:Body>
     <Add xmlns="http://example.com/calculator">
         <intA>10</intA>
         <intB>20</intB>
     </Add>
   </soap:Body>
</soap:Envelope>
```

Response

```
<soap:Envelope>
    <soap:Body>
        <AddResponse xmlns="http://example.com/calculator">
             <Result>30</Result>
        </AddResponse>
   </soap:Body>
</soap:Envelope>
```

SOAP vs REST

Feature	SOAP	REST
Protocol	SOAP Protocol	HTTP
Format	XML	JSON, XML
Complexity	High	Simple
Security	WS-Security	HTTPS
Performance	Slower due to XML overhead	Faster due to lightweight JSON

Tools for Working with SOAP

- **SOAP Clients**: Postman, SoapUI.
- **SOAP Libraries**: Zeep (Python), Apache Axis2 (Java).
- **Frameworks**: Spyne (Python), Spring Boot (Java).