
Unit-3: JSON

Answer the Following Questions (1 Mark Each)

1. The full form of **JSON** is **JavaScript Object Notation**.
2. JSON is mainly used for **storing and exchanging** data.
3. In JSON, data is written in **curly** {} and **square** [] brackets.
4. **False**. The provided documentation describing JSON syntax does not include any syntax for comments.
5. In JSON, a collection of key-value pairs is called an **Object**.
6. JSON is a text-based format derived from **JavaScript** object notation language.
7. JSON **string** values must always be enclosed in **double quotation** marks.
8. The data type used to represent true/false values in JSON is **Boolean**.
9. The provided document does not mention the specific file extension for JSON files.
10. In JSON, arrays are enclosed within **square** brackets.

Answer the Following Questions in Detail (3 Marks Each)

1. Write any three data types supported by JSON with examples.

JSON supports several data types. Here are three of them:

- **Number**: This type is a double-precision floating-point format. It does not use octal or hexadecimal formats.
  - *Example:* { "price": 500 }
- **String**: A string is a sequence of zero or more double-quoted Unicode characters. It can also include special characters using backslash escaping, such as \n for a new line.
  - *Example:* { "bookname": "VB BLACK BOOK" }
- **Boolean**: This type represents a value that can be either **true** or **false**.
  - *Example:* { "isPublished": true }

---

2. Explain JSON Objects with a simple example.

A JSON Object is an **unordered collection of name/value pairs**. The structure is enclosed in **curly braces {}**. Inside the object, each key must be a string, followed by a colon :, and then its corresponding value. If there are multiple key/value pairs, they are separated by a comma ,. Objects are ideal when the key names are arbitrary strings.

- **Example:**
  JSON
  ```
  {
    "bookname": "VB BLACK BOOK",
    "price": 500
  }
  ```

  In this example, "bookname" and "price" are the keys, and "VB BLACK BOOK" and 500 are their respective values.

---

3. Define JSON Schema and state its use.

Definition:
JSON Schema is a specification used to define the structure of JSON data. It is a format based on JSON itself.
Use:
The primary use of JSON Schema is to describe your existing data format. This provides clear documentation that is both human- and machine-readable. It allows you to set constraints on the JSON data, such as required keys, data types for values, and minimum or maximum values, which is useful for validating that the JSON data is structured correctly.

---

4. Write a short note on JSON Parsing.

JSON Parsing is the process of converting a JSON-formatted string into an in-memory object that a programming language can work with. To parse a JSON object, one would typically instantiate an object of a class like JSONObject and pass the string containing the JSON data to it.

The JSONObject class provides various methods to extract specific values from the parsed data. For instance, a JSON file consists of objects with different key/value pairs, and there are separate functions for parsing each component. Methods like getBoolean(String name) or getInt(String name) can be used to retrieve a boolean or integer value associated with a specific key.

Answer the Following Questions (5 Marks Each)

1. Explain JSON Syntax with proper rules and one example.

JSON (JavaScript Object Notation) has a straightforward syntax that is considered a subset of JavaScript's syntax.

Key Syntax Rules:

- **Data Structure:** Data is organized in two main ways:
  1. A collection of **name/value pairs**, known as an object.
  2. An **ordered list of values**, known as an array.
- **Objects:** Objects are enclosed in **curly braces {}**. They contain key/value pairs where the key must be a string.
- **Arrays:** Arrays are enclosed in **square brackets []** and hold an ordered list of values.
- **Separators:**
  - A **colon :** separates a name (or key) from its value within an object.
  - A **comma ,** separates key/value pairs within an object and values within an array.
- **Data Types:** Values can be a string (in double quotes), a number, a boolean (true or false), an array, or another object.

Example:
The following example shows a JSON object that contains various data types, including a nested object and an array.

JSON

```
{
  "name": "John",
```

```
  "age": 30,
  "isStudent": false,
  "courses": ["Web API", "Java"]
}
```

---

2. Differentiate between JSON Serialization and Parsing with examples.

JSON Serialization and Parsing are opposite processes that involve converting data between an in-memory object and a JSON string format.

JSON Serialization 📝
Serialization is the process of converting an object from a programming language into a JSON-formatted string. This is done to store the data or transmit it over a network. The JsonSerializer is a tool that can be used to write an object's data directly into a JSON text format. This process can be customized with various settings to control how the final JSON string is formatted.

- **Example Concept:** Imagine you have a student object in your code. Serializing it would turn it into the following string: {"name":"Aarav","age":25}.

JSON Parsing 🔍
Parsing is the process of converting a JSON-formatted string back into an in-memory object that the application can use and manipulate. A parser, like the JSONObject class, reads the string and rebuilds the data structure it represents. After parsing, you can use methods like getString() or getInt() to access the data by its key.

- **Example Concept:** Taking the string {"name":"Aarav","age":25} and parsing it would create a student object in your program. You could then access the data like student.name to get the value "Aarav".

---

3. Write advantages and disadvantages of JSON in detail.

Advantages of JSON 👍

- **Faster Execution**: JSON has a very simple, lightweight syntax, which makes it easy for machines to parse. This leads to faster data exchange and quicker application responses.
- **Strong Server Parsing**: Server-side parsing of JSON is very efficient. This speed is crucial for web applications, as it allows servers to process requests and send back data to users quickly.
- **Browser Compatibility**: It has wide support across different browsers and operating

systems. This reduces the effort needed to make an application work consistently for all users.
- **Excellent for Data Sharing**: JSON is considered a superior tool for sharing data of any size, including audio and video. This is because it stores data in arrays, which makes the transfer process easier and more organized.

Disadvantages of JSON 👎

- **No Error Handling**: A significant drawback is that the JSON specification itself has no built-in mechanism for handling errors.
- **Security Vulnerabilities**: JSON can be risky if used with untrusted services or browsers. A JSON response can be wrapped in a function call that a browser might execute, potentially opening the web application to attacks. It is very important to be aware of these threats.
- **Limited Development Tools**: Compared to other formats, there is a more limited set of tools available for developers working with JSON.

---

4. Explain JSON Objects and Arrays with examples.

JSON Objects
A JSON Object is an unordered set of name/value pairs used to represent a single entity.

- **Structure:**
  - They are enclosed in **curly braces {}**.
  - Each key must be a string and is followed by a **colon:**.
  - The key/value pairs are separated by **commas ,**.
- **Usage:** Objects are best used when the keys for the data are arbitrary, descriptive strings.
- **Example:**

```JSON
{
  "bookname": "VB BLACK BOOK",
  "price": 500
}
```

JSON Arrays
A JSON Array is an ordered collection of values.

- **Structure:**
  - They are enclosed in **square brackets []**.
  - The values inside the array are separated by **commas ,**.
  - Values can be of any valid JSON data type, including strings, numbers, booleans,

objects, or even other arrays.

- **Usage:** Arrays should be used when you have a list of items, especially when the keys would have been sequential integers (like 0, 1, 2, ...).
- **Example:**

```JSON
[
  "Apple",
  "Banana",
  "Orange"
]
```

Unit-4: Different types of Web Services

Answer the Following Questions (1 Mark Each)

1. The full form of **SOAP** is **Simple Object Access Protocol**.
2. The full form of **REST** is **REpresentational State Transfer**.
3. SOAP is based on **XML**.
4. REST commonly uses the **HTTP** protocol for communication.
5. In SOAP, the **Fault** block contains error messages.
6. RESTful architecture is based on **web standards** principles.
7. SOAP messages are written in **XML** format.
8. The main advantages of REST are its **simplicity** and its ability to work with **heterogeneous languages and environments**.
9. A SOAP message is divided into four main parts: the **Envelope**, **Header**, **Body**, and **Fault** elements.
10. REST methods like GET, POST, PUT, and DELETE are similar to **Read, Insert, Update, and Delete** operations.

Answer the Following Questions in Detail (3 Marks Each)

1. List and explain any three advantages of SOAP.

SOAP (Simple Object Access Protocol) offers several key advantages for data interchange between applications:

- **Platform and OS Independent** 💻: SOAP is designed to be completely independent of the platform and operating system. This means an application built with any programming language on a Windows platform can seamlessly communicate with another application on a Linux platform.
- **Works on Standard HTTP Protocol** 🌐: SOAP functions over HTTP, which is the default protocol for all web applications. This is a major benefit because it requires no special customizations to work on the World Wide Web.
- **Lightweight Protocol** 🪶: Because SOAP is based on XML, which is itself a lightweight data interchange language, SOAP is also considered a lightweight protocol.

---

2. Write a short note on SOAP Building Blocks.

A SOAP message is an XML document composed of several key building blocks that structure the message:

- **Envelope**: This is the mandatory root element that identifies the XML document as a SOAP message. It encapsulates all other parts of the message.
- **Header**: This is an optional element that contains header information. It can hold data like authentication credentials or definitions of complex data types that might be used in the message body.
- **Body**: This element is mandatory and contains the actual call and response information. This is where the data being sent between the web service and the client application resides.
- **Fault**: This optional element is used within the Body to provide information about errors and status that occurred while processing the message.

---

3. Explain any three RESTful methods with examples.

RESTful web services use standard HTTP methods to perform operations on resources. Here are three common methods:

- **GET**: This method is used to retrieve or read data from a resource. It is a read-only and safe operation.
    - **Example**: GET /UserManagement/rest/UserService/users/1
    - **Operation**: This URI would get the user with an ID of 1.
- **POST**: This method is used to create a new resource. It is non-idempotent, meaning calling it multiple times can cause different results.
    - **Example**: POST /UserManagement/rest/UserService/users/2
    - **Operation**: This would insert a new user with an ID of 2.
- **DELETE**: This method is used to remove a resource. It is idempotent, meaning that calling it multiple times has the same effect as calling it once.
    - **Example**: DELETE /UserManagement/rest/UserService/users/1
    - **Operation**: This would delete the user with an ID of 1.

---

4. Define Web Service Components with examples.

Web services are built upon several key components that work together to enable communication:

- **XML-RPC**: This is the simplest XML-based protocol for exchanging information between computers. It uses XML messages sent via HTTP POST to perform remote procedure calls (RPCs). **Example**: A Java client can use XML-RPC to communicate with a Perl server, demonstrating its platform-independent nature.
- **SOAP**: A communication protocol for exchanging information between applications. It uses an XML-based format for sending messages over the internet and can get around firewalls.
- **WSDL (Web Services Description Language)**: An XML-based language used to describe what a web service can do and how to access it. It defines the interface to the web service, acting as a standard format for describing a web service.
- **UDDI (Universal Description, Discovery, and Integration)**: An XML-based standard for describing, publishing, and finding web services. It acts as a distributed registry where businesses can discover each other and learn how to interact over the internet.

Answer the Following Questions (5 Marks Each)

1. Explain SOAP message structure with a neat diagram.

A SOAP message is an XML document that web services use to communicate. Its structure is well-defined and consists of a few key elements that wrap the core data.

Core Components of the Structure:

- **Envelope**: This is the outermost element of the message and is mandatory. It wraps the entire message and identifies it as a SOAP message.
- **Header (Optional)**: If present, the header is the first child of the Envelope. It can contain supplementary information not directly part of the main message, such as authentication details.
- **Body (Mandatory)**: This element contains the primary message data, which includes the specific procedure call and its parameters or the response data.
- **Fault (Optional)**: If an error occurs, a Fault element is placed inside the Body. It contains details about the error and status information.

Diagram and Example Breakdown:

The following code shows a simple SOAP message structure:

XML

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
   <Guru99WebService xmlns="http://tempuri.org/">
     <TutorialID>int</TutorialID>
   </Guru99WebService>
 </soap:Body>
</soap:Envelope>
```

Explanation of the example:

1. **soap:Envelope**: This is the root element that encapsulates the entire SOAP message.
2. **soap:Body**: This element contains the details of the actual message.
3. **Guru99WebService**: This represents the name of the web service method being called.
4. **TutorialID**: This is a parameter required by the web service, which is of the type 'int'.

---

2. Compare SOAP and REST (SOAP vs REST).

SOAP and REST are two different approaches to building web services, each with distinct characteristics.

Feature	SOAP (Simple Object Access Protocol)	REST (REpresentational State Transfer)
Type	SOAP is a **protocol**.	REST is an **architectural style**.
Data Format	Strictly permits the **XML** data format only.	Permits various data formats like Plain text, HTML, XML, and **JSON**.
Standards	Defines **strict standards** that must be followed.	Does not define as many strict standards; it is more flexible.
How it Works	Exposes application logic through **service interfaces**. The Java API for it is JAX-WS.	Exposes application logic using **URIs** (Uniform Resource Identifiers). The Java API is JAX-RS.
Bandwidth	Requires **more bandwidth and resources**.	Requires **less bandwidth** and resources.
Security	Defines its **own security** standards.	**Inherits security measures** from the underlying transport protocol.
Usage	SOAP cannot use REST because SOAP itself is a protocol.	REST can use SOAP as an underlying protocol because REST is a concept.

---

3. Write advantages of REST with its architecture.

Advantages of REST ✅

RESTful services have become popular due to several key advantages that make them suitable for modern web applications:

- **Supports Heterogeneous Environments**: REST allows applications built in different programming languages and running on different environments (e.g., Windows, Linux) to communicate with each other easily.
- **Works with Diverse Devices**: In today's world, applications need to work on everything from mobile phones to car systems. RESTful APIs simplify the process of making these devices talk to web applications without needing to know the underlying technology of the device.
- **Cloud Compatibility**: As applications increasingly move to cloud platforms like Azure and Amazon, REST has become essential. These platforms provide many APIs based on the RESTful architecture, so building services with REST makes it much easier to integrate with and leverage cloud-based services.

REST Architecture 🏗️

The REST (REpresentational State Transfer) architecture, first introduced by Roy Fielding, is a set of principles for designing networked applications.

- **Based on Web Standards**: It is an architecture based on existing web standards and primarily uses the **HTTP protocol** for communication.
- **Resource-Oriented**: In REST, every component is considered a **resource** (e.g., a user, a product). A REST server provides access to these resources, and a REST client accesses and modifies them.
- **URIs for Identification**: Each resource is uniquely identified by a **URI** (Uniform Resource Identifier) or a global ID.
- **Multiple Representations**: Resources can be represented in different formats, such as text, XML, or JSON. **JSON is the most popular format** used with REST today.

---

4. Explain when to use REST and when to use SOAP with examples.

Choosing between REST and SOAP depends heavily on the specific requirements of the

application.

When to Use REST 🚀

REST is generally preferred for its simplicity, speed, and ease of use, making it ideal for modern web and mobile applications.

- Case 1: Developing a Public API
  When creating a public API that will be used by many different developers and applications, REST is the better choice. Its focus on simple, resource-based operations (GET, POST, PUT, DELETE) inherited from HTTP makes it very easy for both developers and web browsers to consume. This simplicity is why major companies like Google and Amazon have moved their public APIs from SOAP to REST.
- Case 2: Mobile Apps and Extensive Back-and-Forth Communication
  APIs for applications that require a lot of messaging, like mobile apps, should use REST. Because REST is stateless (the server doesn't store client information between requests), it handles interruptions well.
  - **Example**: If a user is uploading a photo to **Instagram** on their phone and loses their internet connection, the RESTful service allows the upload process to be retried easily once the connection is back. A stateful SOAP operation would require more complex code to re-establish the connection and state. REST allows for quick calls to a URL for fast responses, and the independence of each request makes testing much simpler.

Unit-5: Web Service (WS) Security and Standards

Answer the Following Questions (1 Mark Each)

1. The full form of **SOA** is **Service Oriented Architecture**.
2. A **threat** or **attack** is an attempt to gain unauthorized access to a system.
3. The provided document does not contain the full form of **SQL**.
4. The provided document does not specify what **SSL** stands for, although it mentions it as a security countermeasure.

5. The provided document does not describe an attack that makes a service unavailable by overwhelming it with traffic.
6. In SOA, services are designed to be **loosely** coupled.
7. The provided document does not mention the **File System Editor**.
8. The provided document does not mention the **User Interface Editor**.
9. The provided document does not mention an editor used to check system requirements before installation.
10. **SSL** is a widely used protocol to secure data over the web.

Answer the Following Questions in Detail (3 Marks Each)

1. Write any three common web security threats.

Three of the top threats directed at Web services are:

- **Unauthorized Access** 🚫: This occurs when weak authentication or authorization is exploited to gain access to sensitive information and operations. Web services with restricted information should always authenticate and authorize their callers to prevent this.
- **Parameter Manipulation** 🔧: This refers to the unauthorized modification of data sent between the consumer and the web service. An attacker can intercept a message, modify its contents, and then send it to its intended endpoint.
- **Message Replay** 🔁: In this type of attack, an attacker captures a message, copies it, and replays it to the web service while impersonating the original client. The message may or may not be modified during this process.

---

2. Explain Web Service Security Standards briefly.

The **WS-Security** standard centers on including the security definition directly within the **SOAP Header** of a message. Credentials in the SOAP header are managed in two primary ways:

1. **UsernameToken**: This is a special element used to pass the username and password to the web service.
2. **BinarySecurityToken**: This method is used when more advanced encryption techniques

like **Kerberos** or **X.509** certificates are required for authentication.

The security model often involves a **Security Token Service**, which issues a security token to the client. The client then embeds this token in the SOAP message when calling the web service, which can then be authenticated.

---

3. What are the uses of the File System Editor, User Interface Editor, and Launch Conditions Editor in a setup project?

The provided document does not contain any information regarding the **File System Editor**, **User Interface Editor**, or **Launch Conditions Editor**. The guide explains how to create a web service project but does not cover setup projects or these specific editors.

---

4. State any three principles of SOA.

Service Oriented Architecture (SOA) is based on several key principles. Here are three of them:

- **Standardized Service Contract**: Services must adhere to a service description that describes what the service is about. This makes it easier for client applications to understand what the service does.
- **Loose Coupling**: This principle states that there should be as little dependency as possible between a web service and the client that invokes it. This ensures that if the service's functionality changes, it does not break the client application.
- **Service Abstraction**: Services should hide the logic they encapsulate from the outside world. A service should only expose what it does, not the complex internal details of how it does it.

Answer the Following Questions (5 Marks Each)

1. Explain countermeasures to prevent web security threats.

To build secure web services, several countermeasures can be implemented to address common threats:

- **To Prevent Unauthorized Access**:
  - Use strong authentication methods like **password digests**, **Kerberos tickets**, or **X.509 certificates** in SOAP headers.
  - Implement **role-based authorization** to restrict access to specific web service files or methods.
- **To Prevent Parameter Manipulation**:
  - **Digitally sign** the message so the recipient can verify that it has not been tampered with in transit.
  - **Encrypt the message payload** to provide both privacy and tamper-proofing.
- **To Prevent Network Eavesdropping**:
  - Use transport-level encryption like **SSL or IPSec** if you control both endpoints of the communication.
  - **Encrypt the message payload** itself to provide privacy, which is effective even when the message passes through intermediary nodes.
- **To Prevent Disclosure of Configuration Data**:
  - Use **NTFS permissions** to authorize access to WSDL files or remove them from the web server entirely.
  - **Disable the documentation protocols** to prevent the dynamic generation of WSDL.
  - Implement strong exception handling that returns only minimal, harmless information to the client.
- **To Prevent Message Replay**:
  - Use an encrypted communication channel like **SSL**.
  - **Encrypt the message payload** to prevent man-in-the-middle attacks where contents are modified.
  - Use a **unique message ID** with each request to detect duplicates and digitally sign the message to prevent tampering.

---

2. How can we build secure web services? Explain with steps.

You can build a secure web service by implementing custom authentication within the SOAP header. Here is a summary of the steps using ASP.NET and Visual Studio:

1. **Create the Project**: Start by creating a new **ASP.NET Web Application** in Visual Studio. Give the project a suitable name and location.
2. **Add a Web Service File**: Right-click the project and add a new **Web Service (ASMX)** file.

3. **Define a Custom SOAP Header Class**: Create a new class that inherits from SoapHeader. This class will be used to define the security credentials you want to pass in the SOAP header. Inside this class, define public string variables for UserName and Password.

```
C#
// Custom class for security credentials
public class AuthHeader : System.Web.Services.Protocols.SoapHeader
{
   public string UserName;
   public string Password;
}
```

4. **Implement the Web Method**:
   - Inside your web service class, create a public instance of your new AuthHeader class.
   - Use the [SoapHeader] attribute just above your web method to associate it with the credentials object.
   - In the web method, add code to validate the UserName and Password from the credentials object. If the credentials are valid, return the expected result. If not, throw a SoapException to deny access.

```
C#
public class TutorialService : System.Web.Services.WebService
{
   public AuthHeader Credentials; // Object for credentials

   [WebMethod]
   [SoapHeader("Credentials")] // Links the header to the object
   public string Guru99WebService()
   {
      // Validation logic
      if (Credentials.UserName.ToLower() != "guru99" ||
         Credentials.Password.ToLower() != "guru99password")
      {
         throw new SoapException("Unauthorized", SoapException.ClientFaultCode);
      }
      else
      {
         return "This is a Guru99 Web service";
      }
   }
}
```

5. **Verify**: Once the service is running, you can inspect its WSDL (Service Description) to see that the UserName and Password fields are now part of the definition, indicating they must be sent when the service is invoked.

---

3. Discuss best practices for web service security with examples.

Implementing robust security for web services involves several best practices beyond just authentication and encryption:

- **Auditing and Log Management**: Log all requests that come to the web service. This creates a detailed report of who invoked the service, which is crucial for impact analysis if a security breach occurs.
- **Monitor the Flow of Calls**: Note and log the flow of calls, especially when one application calls multiple web services and authentication tokens are passed between them.
- **Avoid Logging Sensitive Information**: Never include sensitive data like **passwords** or **credit card numbers** in your log files. If an event contains such information, it must be discarded before being logged.
- **Track Significant Business Operations**: Instrument your application to record access to sensitive business logic.
    - **Example**: In an online shopping application, the entire workflow—from choosing items, adding them to the cart, to the final purchase—should be tracked by the web service to monitor for unusual activity.
- **Use Proper Authentication**: Authentication establishes a client's identity using a set of credentials. A critical best practice is to **never store user credentials**. If WS-Security is used, the web service should receive the credentials from the SOAP header, use them for authentication, and then immediately discard them.

---

4. Explain SOA principles in detail with suitable examples.

Service-Oriented Architecture (SOA) is an architectural pattern that structures software components as independent services communicating over a network. It is based on the following key principles:

- **Standardized Service Contract**: Services must have a formal description of what they do, making it easier for client applications to understand and interact with them.
- **Loose Coupling**: There should be minimal dependency between services and the clients that use them. This allows a service's functionality to change without breaking the client application.
- **Service Abstraction**: Services hide their internal logic from the outside world. They only expose *what* they do, not *how* they do it, simplifying their use.
- **Service Reusability**: Logic is organized into services with the goal of maximizing reuse

across multiple applications. Once code for a service is written, it should have the ability to work with various application types.

- **Service Autonomy**: Services have complete control over the logic they encapsulate.
- **Service Statelessness**: Ideally, services should be stateless, meaning they don't hold information from one request to the next. The state should be managed by the client application.
    - **Example**: A web service might provide the price of an item. If that item is added to a shopping cart and the user proceeds to checkout, the responsibility of carrying the item's price to the payment page belongs to the web application, not the web service.
- **Service Discoverability**: Services can be discovered, typically in a service registry like **UDDI**.
- **Service Composability**: Large problems are broken down into smaller, manageable services. An application's functionality should be divided into modules, each with a separate business function.
- **Service Interoperability**: Services should use standards like **XML** and communicate over protocols like **HTTP** to allow diverse clients to use them.