**Part 1 of IA 2**

**Questions:**

1. Explain the architecture of web services and the role of servers in hosting them.
2. Differentiate between RESTful and SOAP-based services.
3. Implement a simple HTTP-based web service using Flask or Node.js and deploy it on a server.

**Answer 1:**

**Web Services Architecture**

Web services follow a client-server model, enabling communication between different systems over a network. They allow applications to exchange data or perform tasks regardless of platform or programming language.

**Key Components**

1. **Client**
   * The client can be a web browser, mobile app, or another service making API calls.
   * It sends HTTP requests (GET, POST, PUT, DELETE) to the server.
2. **Server**
   * The server processes incoming requests, retrieves or modifies data, and sends appropriate responses.
   * Frameworks like **Flask (Python)** or **Express.js (Node.js)** are commonly used to manage server-side logic and handle requests.
3. **API (Application Programming Interface)**
   * Defines how clients interact with the web service.
   * Uses standards like **RESTful APIs** (most common) or **SOAP** (older but still in use).
   * Example endpoints:
     + GET /users → Fetch user data
     + POST /users → Add a new user
4. **Database**
   * Stores and manages data for the web service.
   * Common options include **MySQL**, **PostgreSQL**, and **MongoDB**.
   * The server interacts with the database to fetch or update data as needed.
5. **Middleware**
   * Middleware acts as a bridge between the client and server to handle tasks like authentication, logging, and request processing.
   * Example: **JWT (JSON Web Tokens)** for secure user authentication.

**Typical Workflow**

1. The **client** sends an HTTP request.
2. **Middleware** may validate the request or enforce security policies.
3. The **server** processes the request and may interact with the **database**.
4. The **server** sends a structured response (e.g., JSON or XML) back to the **client**.

**Answer 2:**

**Difference Between RESTful and SOAP-Based Services**

Both RESTful and SOAP-based services are used for web communication, but they differ significantly in their architecture, design, and usage. Here's a detailed comparison:

| **Aspect** | **RESTful Services** | **SOAP-Based Services** |
| --- | --- | --- |
| **Protocol** | Uses **HTTP** as the primary communication protocol. | Can use various protocols like **HTTP**, **SMTP**, **TCP**, etc. |
| **Message Format** | Commonly uses lightweight formats like **JSON** or **XML**. | Uses **XML** exclusively, making it more verbose. |
| **Flexibility** | Highly flexible, allowing data exchange in different formats (JSON, XML, etc.). | Rigid structure with strict XML-based messaging. |
| **Performance** | Faster due to less overhead and reduced payload size. | Slower due to its heavier XML format and strict standards. |
| **Ease of Use** | Easier to implement with simple CRUD operations via HTTP methods (GET, POST, PUT, DELETE). | More complex to implement with defined standards for messaging and error handling. |
| **State Management** | Stateless architecture: each request is independent. | Can be **stateful** or **stateless** depending on the application's design. |
| **Security** | Relies on HTTPS for secure communication; additional security layers like OAuth or JWT are often implemented. | Has built-in security features such as WS-Security for secure message transmission. |
| **Best Suited For** | Lightweight web services requiring fast communication and scalability. | Applications requiring strict security, ACID compliance, or complex transactions. |

**Example Use Cases**

* **RESTful Services:** Social media APIs, mobile applications, web-based applications.
* **SOAP Services:** Banking systems, financial transactions, or services requiring strict security standards.

**Answer 3:**

First create repository on GitHub and then push the index.js and required dependencies on the newly created repository.

**Step 1: Initialize Git**

git init

**Step 2: Add Your Files**

git add .

**Step 3: Commit the Changes**

git commit -m "Initial commit"

**Step 4: Link to GitHub Repository**

git remote add origin https://github.com/rachitshah-1/CyberSec-IA.git

Verify the remote URL:

git remote -v

**Step 5: Push to GitHub**

git branch -M main

git push -u origin main

**GitHub Link:** <https://github.com/rachitshah-1/CyberSec-IA>

A screenshot of a computer

AI-generated content may be incorrect.

http://localhost:3000/users → Returns the user list in JSON format.

A screenshot of a computer

AI-generated content may be incorrect.