PANDIT DEENDAYAL ENERGY UNIVERSITY SCHOOL OF TECHNOLOGY

Course: Cyber Security

Course Code: 23CP310P

LAB REPORT

B.Tech. (Computer Engineering)

Semester 6(G9)

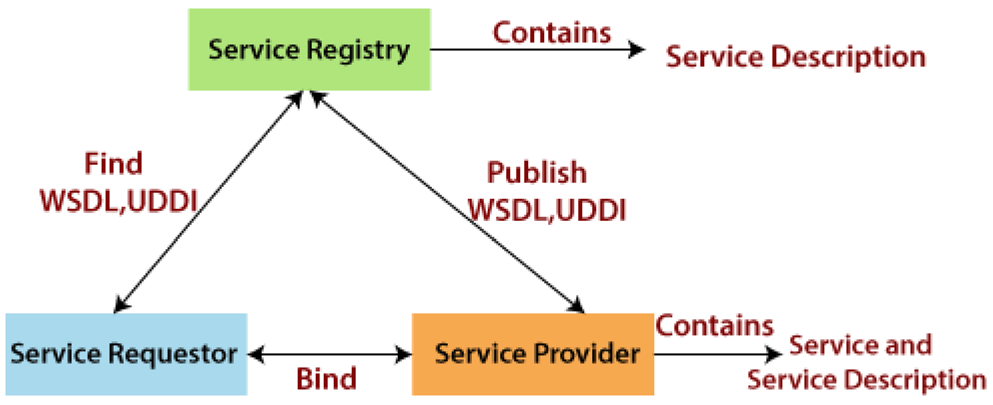
Submitted by:
JYOT PANDYA
22BCP350

1..Architecture of Web Services and the Role of Servers

Web services are software systems designed to support interoperable machine-to-machine interaction over a network. They typically use standardized protocols like HTTP, XML, SOAP, WSDL, and REST to facilitate communication between different systems.

Key Components of Web Services Architecture:

1. **Client**: The application or system that initiates a request to the web service.
2. **Server**: The system that hosts the web service and processes client requests.
3. **Protocols**: Standards like HTTP, HTTPS, SOAP, REST, etc., used for communication.
4. **Data Formats**: XML, JSON, or other formats used to structure the data being exchanged.
5. **Service Description**: WSDL (Web Services Description Language) for SOAP-based services or OpenAPI for RESTful services, which describe how to interact with the service.

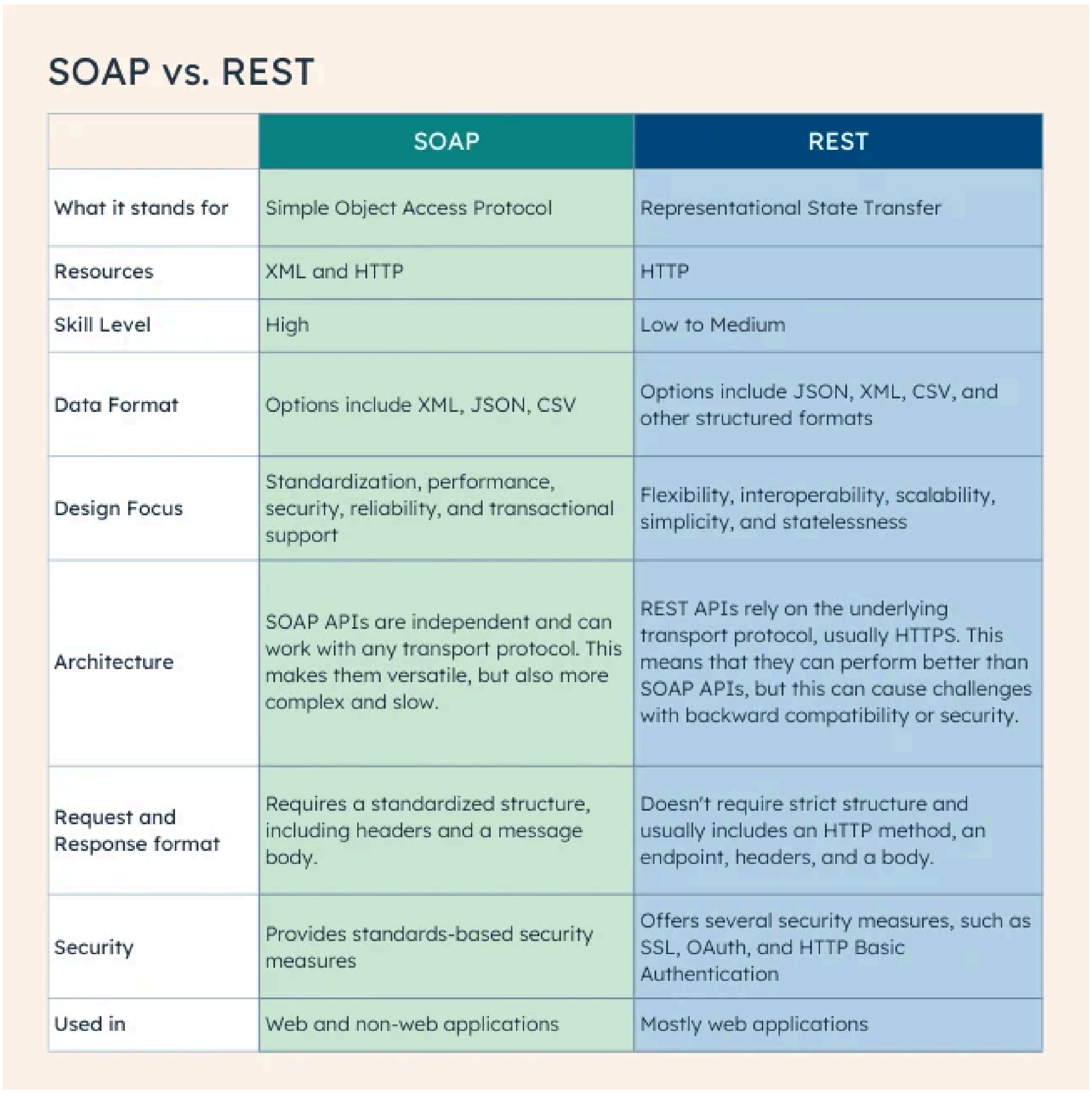


Web Service Roles, Operations and Artifacts

Role of Servers:

- **Hosting**: Servers host the web service, making it accessible over the internet or an intranet.
- **Request Handling**: Servers receive client requests, process them, and return appropriate responses.
- **Scalability**: Servers can be scaled horizontally (adding more servers) or vertically (increasing server resources) to handle increased load.
- **Security**: Servers implement security measures like SSL/TLS encryption, authentication, and authorization to protect the service.

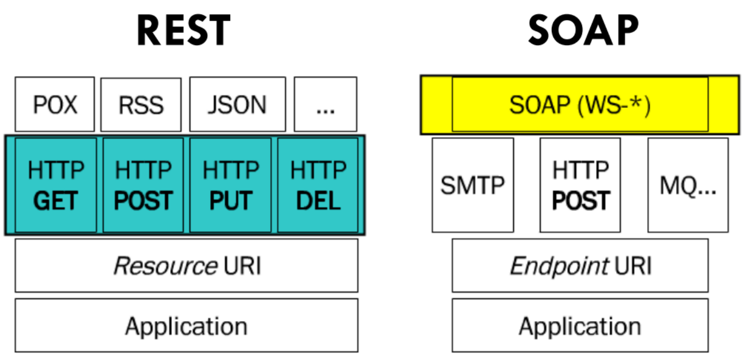
2..RESTful vs. SOAP-Based Services



SOAP vs. REST

	SOAP	REST
What it stands for	Simple Object Access Protocol	Representational State Transfer
Resources	XML and HTTP	HTTP
Skill Level	High	Low to Medium
Data Format	Options include XML, JSON, CSV	Options include JSON, XML, CSV, and other structured formats
Design Focus	Standardization, performance, security, reliability, and transactional support	Flexibility, interoperability, scalability, simplicity, and statelessness
Architecture	SOAP APIs are independent and can work with any transport protocol. This makes them versatile, but also more complex and slow.	REST APIs rely on the underlying transport protocol, usually HTTPS. This means that they can perform better than SOAP APIs, but this can cause challenges with backward compatibility or security.
Request and Response format	Requires a standardized structure, including headers and a message body.	Doesn't require strict structure and usually includes an HTTP method, an endpoint, headers, and a body.
Security	Provides standards-based security measures	Offers several security measures, such as SSL, OAuth, and HTTP Basic Authentication
Used in	Web and non-web applications	Mostly web applications

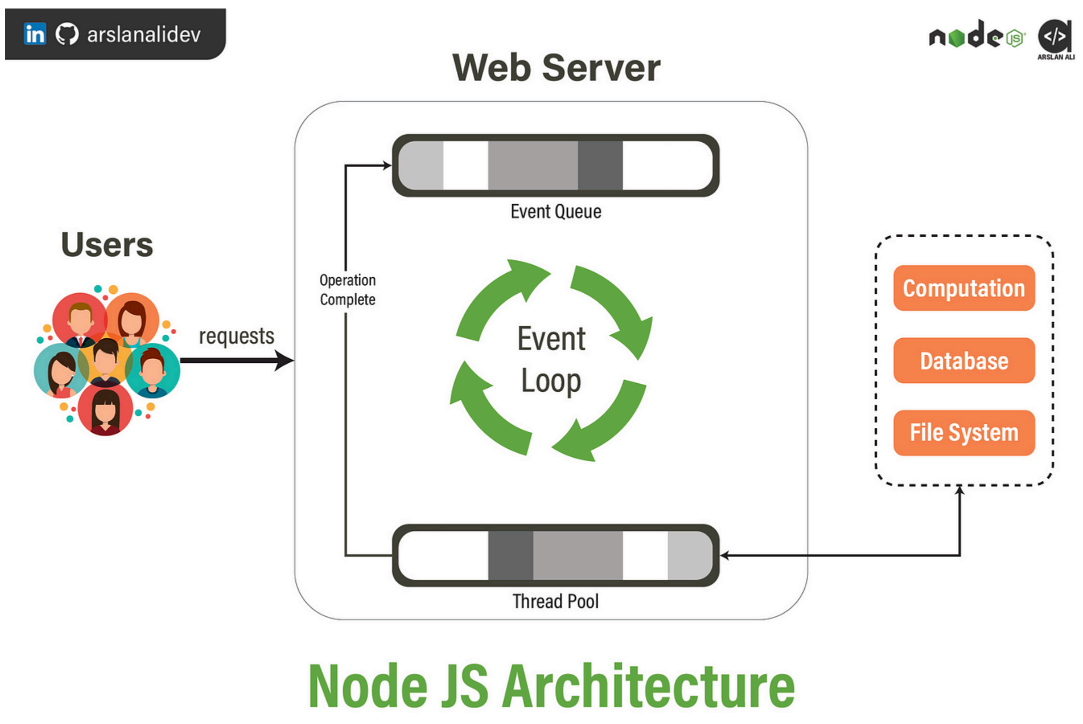
- **Protocol Layering**



3.Implementing a Simple HTTP-Based Web Service

simple web service using **Node.js**.

1.Using Node.js (Express)

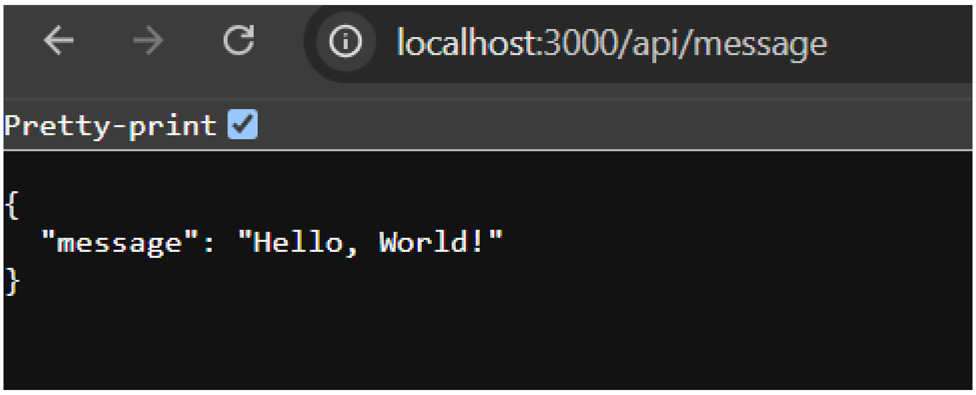


Code:

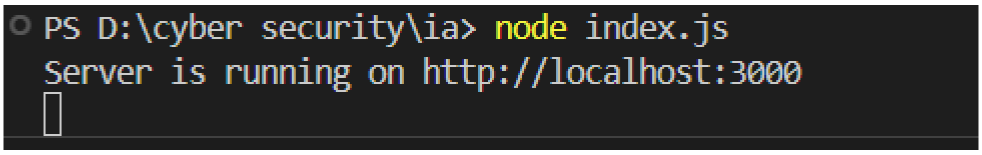
```
const express = require('express');

const app = express();

const port = 3000;


// Middleware to parse JSON bodies

app.use(express.json());


// Define a simple GET endpoint

app.get('/api/message', (req, res) => {

  res.json({ message: 'Hello, World!' });

});
```

```
// Start the server
app.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}`);
});
```

Output:



```
{
  "message": "Hello, World!"
}
```



```
PS D:\cyber security\ia> node index.js
Server is running on http://localhost:3000
```

Deployment on Localhost

To deploy the Node.js web service on localhost, the following steps were executed:

1. **Start the Server**:
   - Execute the following command in the terminal to run the service:
     `node index.js`
   - This command initiates the server and makes it listen on port 3000.
2. **Access the Web Service**:
   - Open a web browser and navigate to
     `http://localhost:3000/api/message`.
   - Alternatively, use a tool like `curl` to test the endpoint:
     curl http://localhost:3000/api/message
3. **Verify Output**:
   - The expected output is a JSON message indicating the service is running correctly:

```
{
  "message": "Hello, World!"
}
```