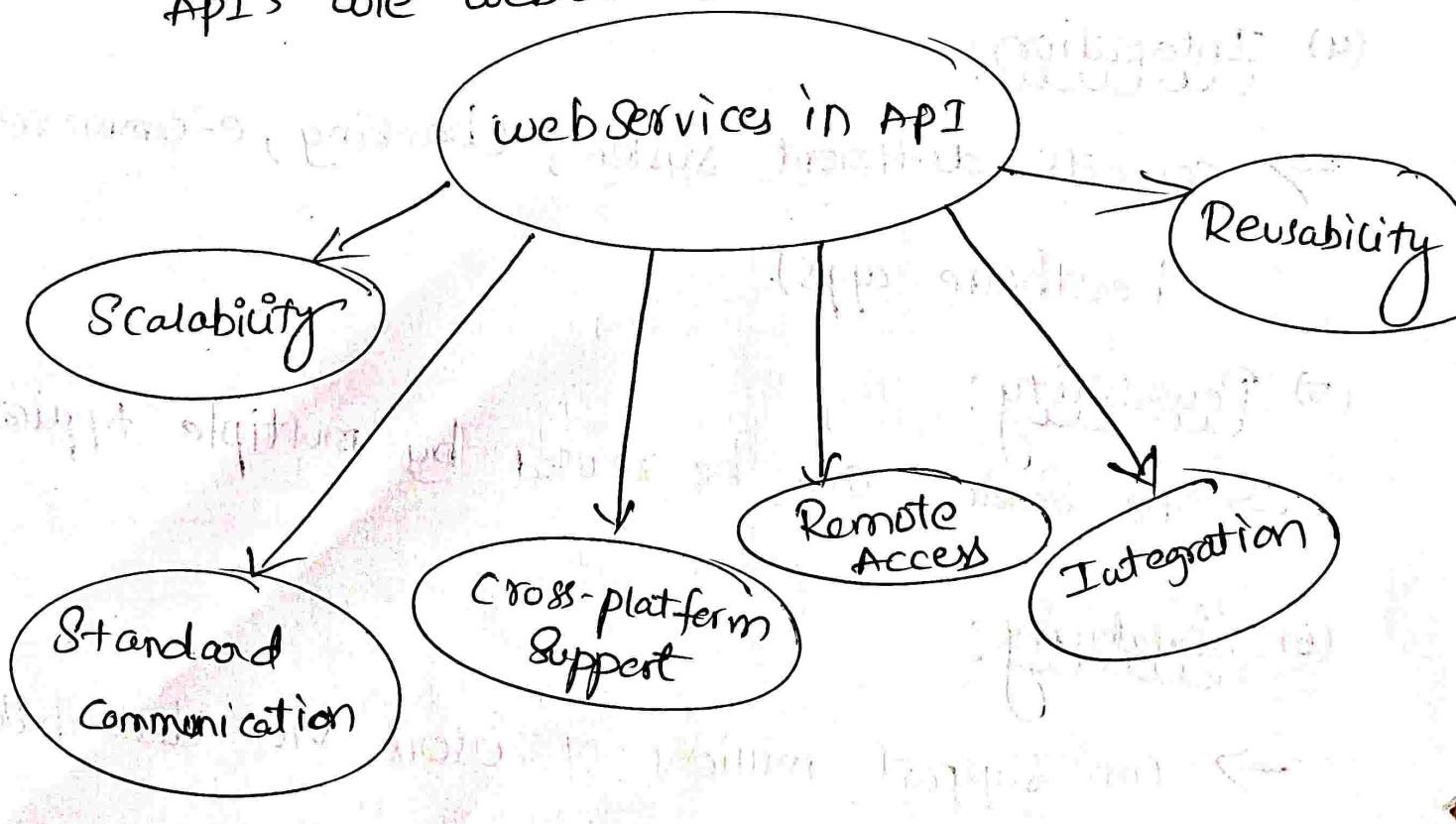
✳ Why WebServices: (API)

API: (Application programming Interface):

→ A Set of rules that allows one Software to interact with another.

→ Web Services are a type of API that Specifically use web protocols (HTTP/HTTPS) for communication

→ So, all web Services are API's, but not all API's are web Services.



Reasons why we use webService in API:

(1) Standard communication:

→ Uses HTTP, XML, JSON, SOAP, REST for consistent interaction.

(2) Cross-platform Support:

→ Works between Java, .NET, Python etc.

(3) Remote Access:

→ Can call Services over the internet from anywhere.

(4) Integration:

→ connects different Systems (banking, e-commerce, healthcare apps).

(5) Reusability:

→ One Service can be reused by multiple Applications.

(6) Scalability:

→ Can Support millions of users via distributed Systems.

Example:

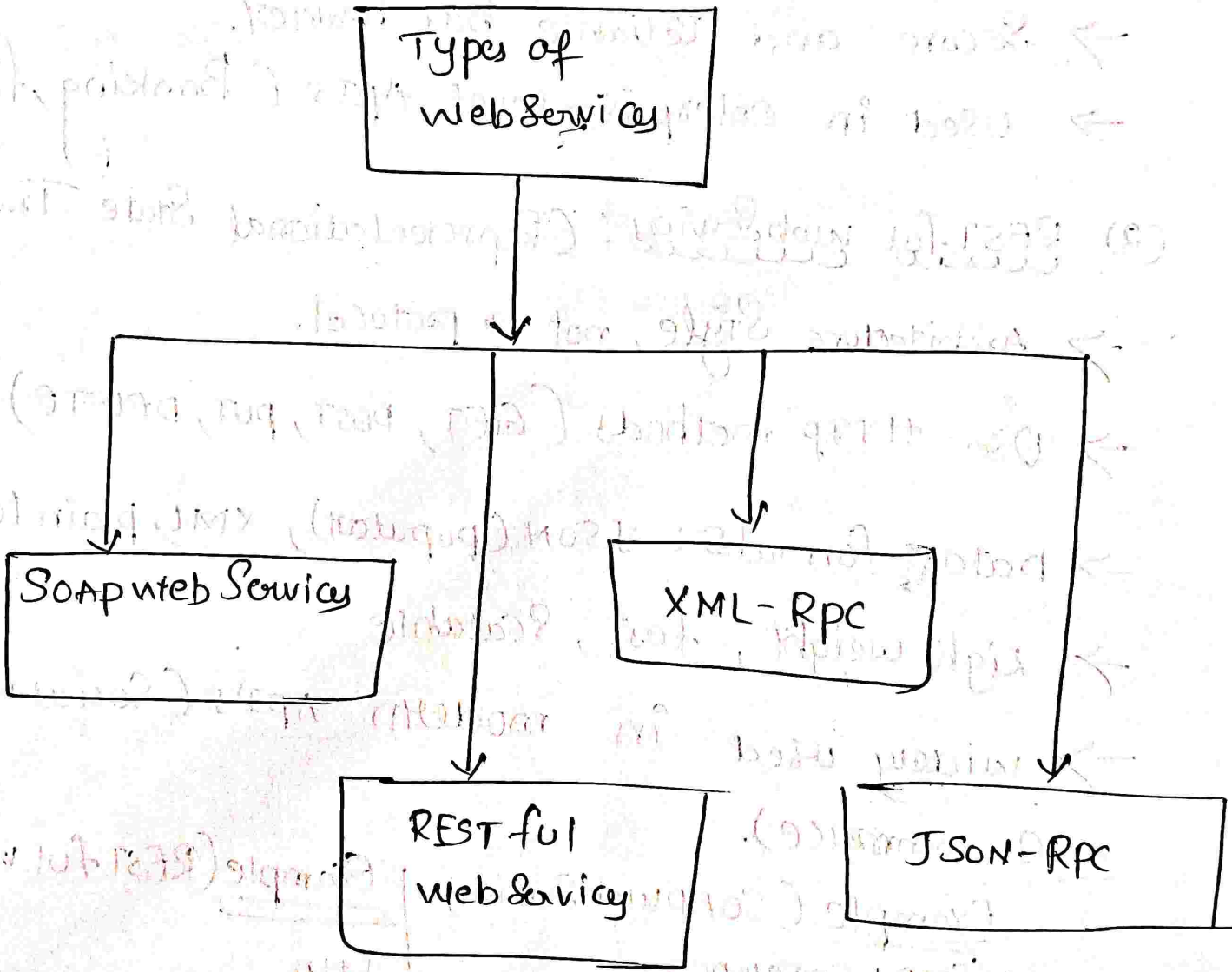
→ when you use a mobile UPI payment app, it calls bank web Services APIs to check your balance, transfer money and confirm the transaction.

1

* Types of Web Services in API :

→ Web Services are type of API (Application programming Interface) that allow communication between two applications over a network.

The main types are



Types of webservices in API are

(1) SOAP Web Service (Simple object Access protocol)

→ protocol-based
→ Uses XML for data exchange.
→ Requires WSDL (Web Services Description Language)
→ Secure and Reliable but heavier.
→ Used in enterprise-level APIs (Banking, finance)

(2) RESTful webservices: (Representational State Transfer)

→ Architecture Style, not a protocol.
→ Uses HTTP methods (GET, POST, PUT, DELETE)
→ Data formats: JSON (popular), XML, plain text.
→ Light weight, fast, Scalable
→ widely used in modern APIs (Social Media, e-commerce).

Example (SOAP webservice)

```
<Soap:Envelope>
  <Soap:Body>
    <Get Weather>
      <city> kalcinada </city>
    </Get Weather>
  </Soap:Body>
</Soap:Envelope>
```

Example (RESTful webservice

```
http
GET/api/Weather?city=kalcinada
                    HTTP/1.1

Host: api.example.com
Response: json
{ "city": "kalcinada",
  temperature": "15°C" }
```

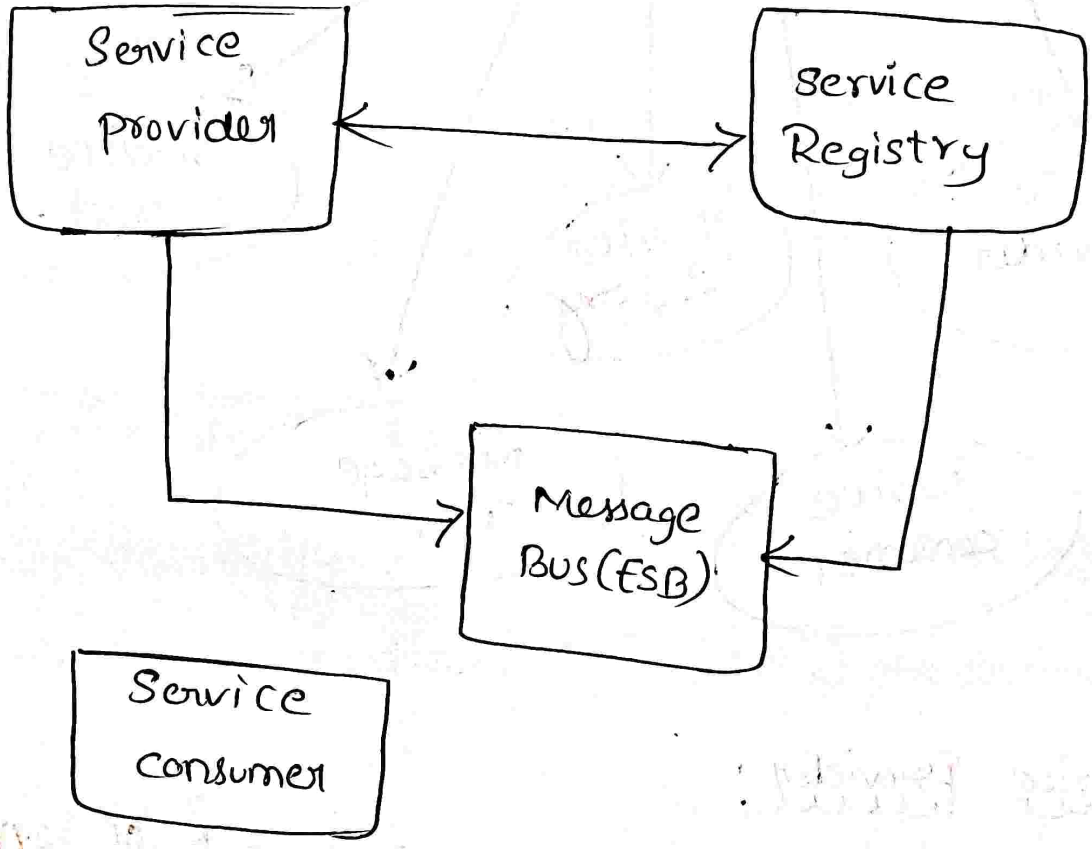
3) XML-RPC: (Remote procedure call)

→ Remote procedure call using XML
→ Encodes requests / responses in XML
→ Works over HTTP
→ Older style, less common now.

(4) JSON-RPC (Javascript object Notation)

→ Remote Procedure call using JSON.
→ Similar to XML-RPC but uses JSON.
→ Simpler, faster, light weight.

* SOA Architecture: (Service-oriented Architecture)

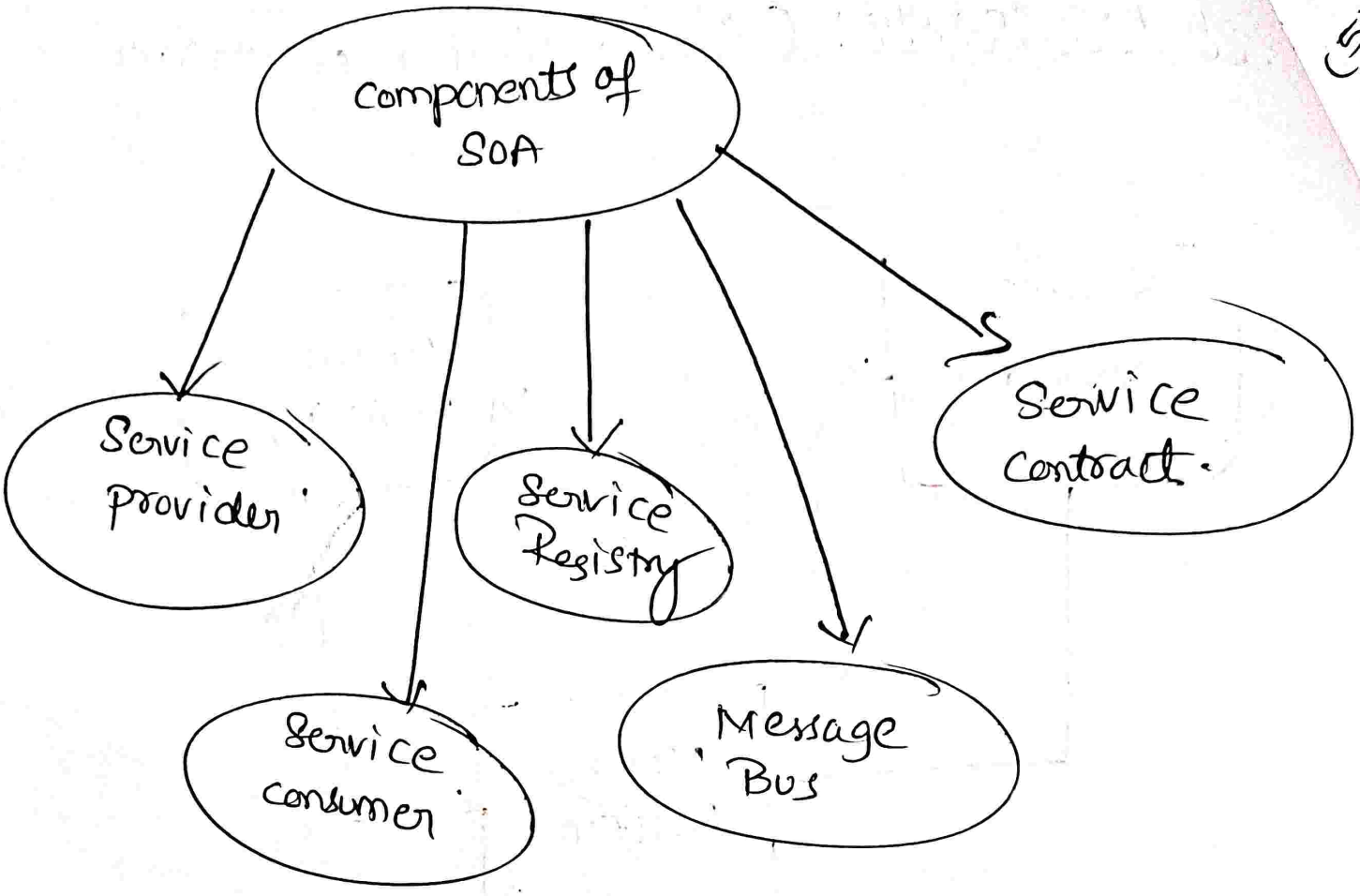


SOA :-

→ It is a design style where software components are built as independent services that can communicate with each other.

→ Each service is self-contained, reusable and provides a specific bussiness function.

→ Communication happen through standard protocols like → HTTP
→ SOAP
→ REST
→ XML
→ JSON



(1) Service Provider:

→ Service provider is a main component of SOA, creates and publishes services.

(2) Service Consumer

→ uses / invokes the service.

(3) Service Registry:

→ A directory where services are published and discovered.

(4) Message Bus (Enterprise Service Bus - ESB)

→ Message Communication between services.

(5) Service Contract:

→ Defines rules (WSDL, API contract).

Advantages:

→ Loose Coupling (Services are independent)

→ Reusability (Same Service can be reused in many apps).

→ platform independent (works on any OS/language)

→ Scalability & flexibility.

→ Easy integration of new and old Systems.

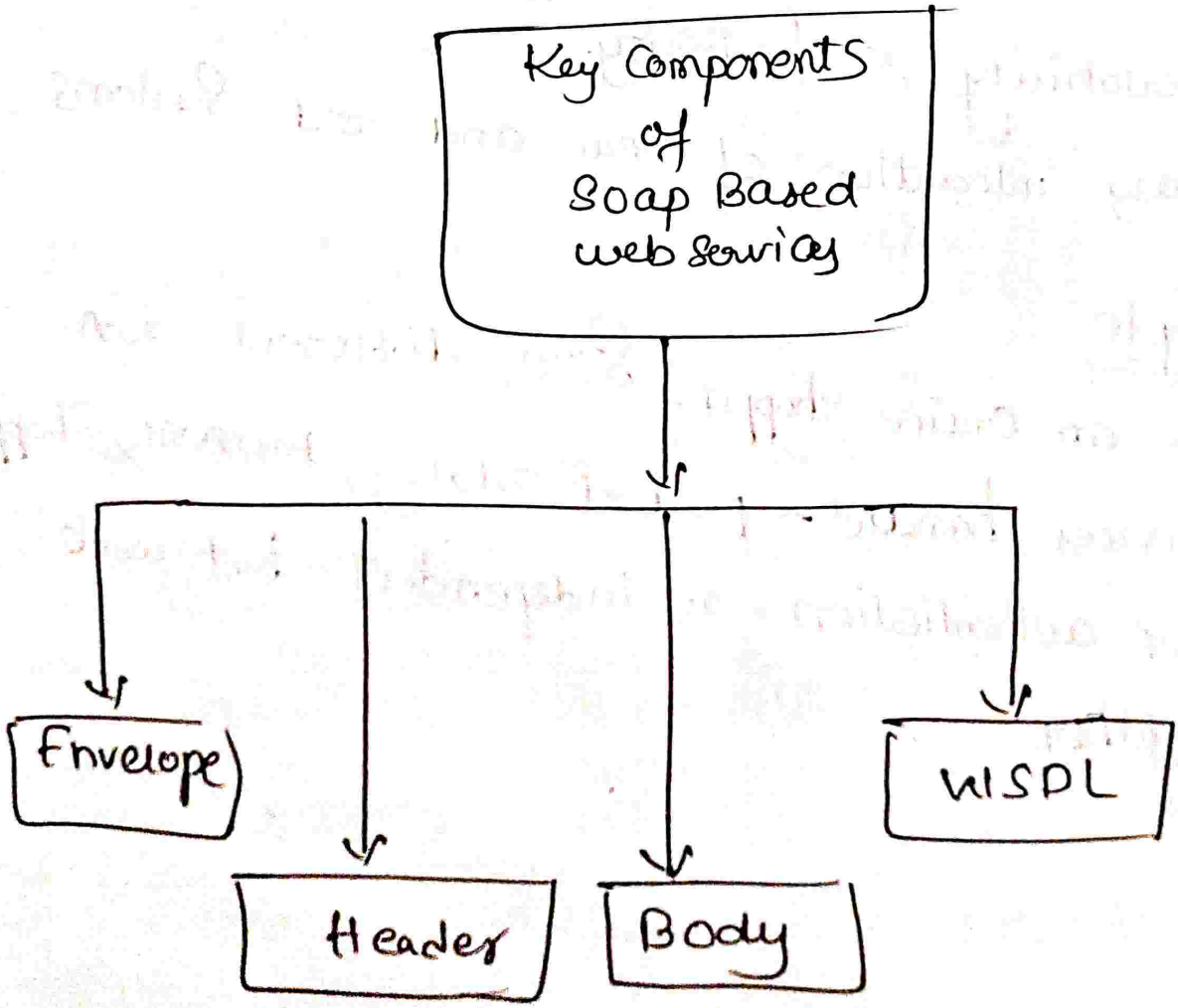
Example

→ In an Online shopping Site, different SoA Services handle - product catalog, payment, Shopping, User authentication - all independent but work together.

* SOAP Based web Services:

→ SOAP (Simple Object Access protocol) is a protocol used for exchanging Structured information in web Services.

→ It releases on XML for message format can operate over protocols like HTTP, SMTP and TCP.



Key components

(1) Envelope – Root element of a SOAP message containing Header and Body.

(2) Header – optional element containing metadata.

(3) Body – Contains the actual message Content.

(4) WSDL (Web Services Description Language) :- XML file describing the SOAP Service, operation and Datatypes.

Advantages

→ platform Independent

→ protocol flexibility (HTTP, SMTP, TCP)

→ Built in error Handling

→ Support WS-Security

Dis-Advantage

→ Slower than REST

→ More complex to implement.

- Example SOAP Request :

```
<GetEmployeeDetailsRequest xmnlg="http://example.com/
                                  employee">

  <EmployeeID>1243</EmployeeID>

</GetEmployeeDetailsRequest>
```

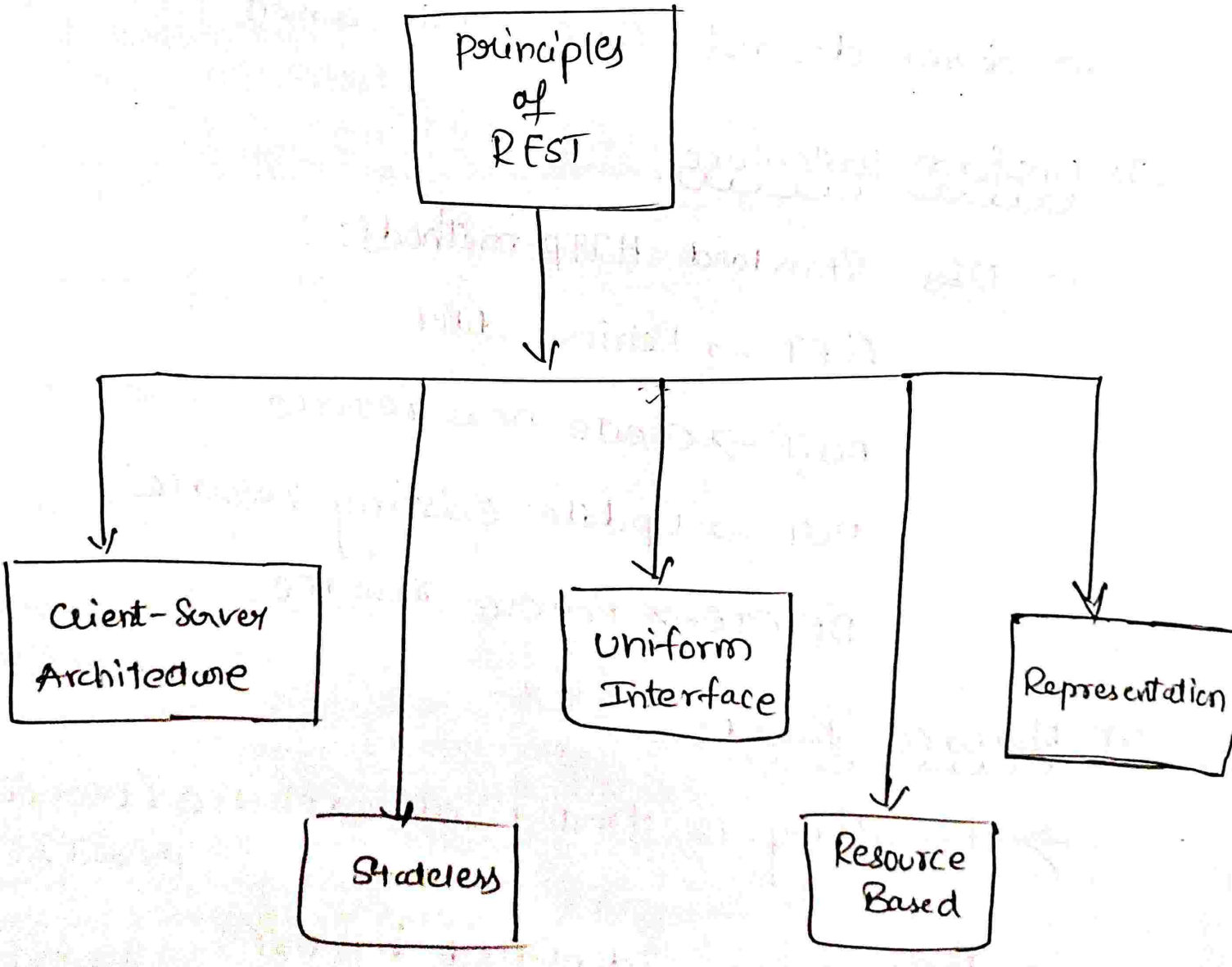
Example SOAP Response :

```
<GetEmployeeDetailsResponse xmnlg="http://example.com/
                                  employee">
    <Employee>
      <ID>1243</ID>
      <Name>vaibhav</Name>
    </Employee>
</GetEmployeeDetailsResponse>
```

* RESTful web Services:

→ A RESTful web Service is a way for applications to communicate with each other over the internet using REST (Representational State Transfer) principles.

It is based on HTTP protocol and uses Stateless communication between client & Server.



Principles of REST:

(1) Client-Server Architecture:

→ Client (like browser, mobile app) request data.

→ Server processes and sends response.

(2) Stateless:

→ Each request from the client to the server must contain all the information needed.

→ Server does not store client session state.

(3) Uniform Interface:

→ Uses Standard HTTP methods:

GET → Retrieve data

POST → Create new resource

PUT → Update existing resource

DELETE → Remove resource

(4) Resource-Based:

→ Everything is treated as a resource (user, order, product).

→ Resources are identified by URLs

Eg: /api/users/1

(5) Representation:

→ Resources are usually returned in JSON (or) XML

Advantages of RESTful web Services:

→ Simple and lightweight.

→ platform independent (works with any language)

→ Scalable and flexible.

→ uses Standard HTTP methods.

* How to create RESTful web Services:

Steps to create RESTful Services

1. Choose a Technology / Framework

↳ Java → Spring Boot, JAX-RS

↳ phyton → Flask, FastAPI,

↳ Node.js → Express.js

↳ .NET → ASP.NET web API

2. Design Resources (API endpoints)

→ Decide which resources you need (users, orders, products etc).

→ Assign meaningful URL's (eg., /api/users, api/orders)

3. Use HTTP Methods:

↳ GET - Retrieve

↳ POST - create

↳ PUT - update

↳ DELETE - Remove

~~Use HTTP~~

4. Set Data Format:

→ Use JSON (default & lightweight) or XML if needed.

5. Implement Controller / Handler

→ Create methods mapped to endpoints.

6. Test your API:

→ Use tools like postman, curl (or) browser for GET requests.

Example:

```
import java.util.*;
import org.springframework.web.bind.annotation.*;
@RestController
@RequestMapping("/api/users")
    public class UserController
        {
        private Map<Integer, String> users = new HashMap();
        //GET - Get all users
        @GetMapping
        public Collection<String> getUsers()
            {
            return users.value();
            }
        //GET - Get user by ID
        @GetMapping("/{id}")
    public String getuser(@pathVariable int id)
        {
        return users.getordefault(id, "user not found");
        }
```

```
// POST - create new user
@PostMapping
public String createUser(@RequestParam String name)
{
    int id = users.size() + 1;
    users.put(id, name);
    return "User created with ID: " + id;
}
// DELETE - Delete User
@DeleteMapping("/{id}")
public String deleteUser(@PathVariable int id)
{
    users.remove(id);
    return "User deleted with ID: " + id;
}
}
```