✅ Summary — SOAP (Simple Object Access Protocol)

SOAP is an XML-based protocol used to exchange data between applications over the internet, mainly via HTTP.
It was created so that applications built using different languages (Java, .NET, PHP, etc.) can communicate easily.

✅ Why SOAP?

- Different systems need to exchange data (e.g., Java + .NET).
- XML acts as a common format.
- SOAP provides **standard rules** for using XML so that all systems understand the message.

✅ Advantages of SOAP

- Platform-independent; language-independent.
- Works using HTTP.
- Lightweight protocol for data exchange.
- Standard recommended by W3C.

✅ SOAP Message Building Blocks

A SOAP message is an XML document made of:

1. **Envelope** —
   Marks the beginning and end of the SOAP message.
2. **Header** *(optional)* —
   Can include security details, authentication, and additional metadata.
3. **Body** —
   Contains the real request or response data.

Simplified structure:

```
<Envelope>
   <Header>
   </Header>
   <Body>
```

```
   </Body>
</Envelope>
```

Example Body:

```
<soap:Body>
  <GetTutorialInfo>
    <TutorialName>Web Services</TutorialName>
    <TutorialDescription>About web services</TutorialDescription>
  </GetTutorialInfo>
</soap:Body>
```

✅ SOAP Faults

When an error occurs, SOAP responds with an **HTTP 500 error** and provides fault details.

✅ SOAP Communication Model

SOAP uses **HTTP** for communication.

Steps:

1. **Client sends request**
    - Converts request + parameters into SOAP XML → (*Marshalling*)
2. **Server receives and processes**
    - Server reads XML → (*Demarshalling*)
    - Executes task and returns SOAP response.

✅ Real-World Example

📌 Banking System — Money Transfer Service

Suppose a bank's core system is built in Java, and an ATM network uses a .NET application. To transfer money, both systems need to communicate.

→ They use a SOAP web service.

SOAP Request

The ATM sends a SOAP request:

```
<soap:Envelope>
  <soap:Body>
    <TransferMoney>
      <AccountFrom>1234</AccountFrom>
      <AccountTo>5678</AccountTo>
      <Amount>200</Amount>
    </TransferMoney>
  </soap:Body>
</soap:Envelope>
```

SOAP Response

The bank service responds:

```
<soap:Envelope>
  <soap:Body>
    <TransferResponse>
      <Status>Success</Status>
      <Balance>800</Balance>
    </TransferResponse>
  </soap:Body>
</soap:Envelope>
```

→ This allows different systems to exchange data securely and reliably.

☑ Another Example (Simple)

Weather Information Service

You call a SOAP service:
Request →

```
<GetWeather>
   <City>Colombo</City>
</GetWeather>
```

Response →

```
<Temperature>29</Temperature>
<Humidity>70%</Humidity>
```

✅ In Simple Words

SOAP = A standard XML message format
Used so **different applications can talk to each other safely over the internet**

It includes:
✔ Envelope
✔ Header
✔ Body

Works using → HTTP
Used for → Banking, Insurance, Enterprise Systems

✅ SOAP — Full Summary + Real-World Examples + REST vs SOAP Comparison + Diagram

---

✅ 1) Summary of SOAP (Simple Object Access Protocol)

SOAP is a **standard XML-based protocol** used to exchange data between applications over a network (usually HTTP).
It was created to make communication possible between applications written in different programming languages.

✅ Key Points

- Uses **XML messages**
- Platform & language independent
- Highly secure and well-structured
- Recommended by **W3C**
- Best suited for enterprise applications

✅ 2) SOAP Message Structure

A SOAP message consists of:

```
<Envelope>
   <Header> (optional)
   </Header>
   <Body>
   </Body>
</Envelope>
```

🔹 Envelope

Defines start/end of SOAP message.

🔹 Header *(optional)*

Used for metadata (authentication, security, transaction info).

🔹 Body

Contains actual data (request/response).

✅ 3) SOAP Faults

If an error occurs, SOAP returns:

- HTTP **500 error**
- SOAP Fault block with error message

✅ 4) SOAP Communication Model

1. Client converts instruction → SOAP XML (**Marshalling**)
2. Sent via HTTP
3. Server unpacks XML (**Demarshalling**)
4. Server processes & returns SOAP XML response

✅ 5) Real-World Examples of SOAP

Example 1: ATM Money Transfer

An ATM machine needs to communicate with the bank server.

📤 Request:

```
<Envelope>
  <Body>
    <TransferMoney>
      <AccountFrom>1111</AccountFrom>
      <AccountTo>2222</AccountTo>
      <Amount>500</Amount>
    </TransferMoney>
  </Body>
</Envelope>
```

📥 Response:

```
<Envelope>
  <Body>
    <TransferResponse>
      <Status>Success</Status>
      <Balance>4500</Balance>
    </TransferResponse>
  </Body>
</Envelope>
```

✅ Works even if ATM uses .NET & bank uses Java.

Example 2: Flight Booking System

Airlines use SOAP to:

- Check seat availability
- Book seats
- Update traveler details

Booking platforms like **Amadeus & Sabre** use SOAP.

Example 3: Bank-to-Bank communication

Bank A → SOAP → Bank B
Used for:

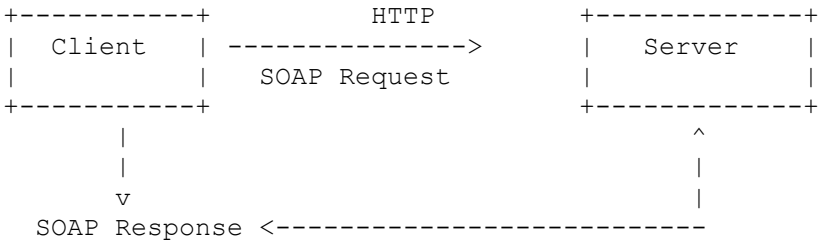
- International money transfer
- Secure messaging
- Account validation

✅ 6) SOAP vs REST

Feature	SOAP	REST
Data Format	XML only	JSON, XML, etc.
Protocol	Uses HTTP only	Uses HTTP
Complexity	More complex	Simple
Speed	Slower (XML)	Faster
Security	Very strong (WS-Security)	Limited
Best Use	Banking, Finance, Enterprise systems	Mobile, Web Apps
Caching	No	Yes
Message Format	Heavy	Lightweight

✅ REST is faster → used in web/mobile
✅ SOAP is more secure → used in banking, healthcare

✅ 7) SOAP Diagram (Conceptual)



✅ 8) Simple Explanation

SOAP is like sending a **formal letter** — fixed format, very structured
REST is like sending a **text message** — simple and flexible

---

✅ 9) When to Use SOAP

✔ Financial systems
✔ Enterprise applications
✔ Highly secure environments
✔ When reliability is critical
✔ When ACID transactions needed

✅ 10) When NOT to Use SOAP

✗ Lightweight apps
✗ Mobile apps where speed is needed
✗ Simple CRUD operations

Below is **one complete, big SOAP example** that shows:

✅ Input (Request)
✅ Output (Response)
✅ Explanation

✅ FULL SOAP EXAMPLE

Example: Online Bank – Check Account Balance

A client wants to check the balance of a bank account using a SOAP web service.

✅ 1) SOAP Request (Input)

The client sends a SOAP message to the web service:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:bank="http://example.com/bank">
  <soap:Header>
    <bank:Authentication>
      <bank:UserID>USER123</bank:UserID>
      <bank:Token>ABCD1234</bank:Token>
    </bank:Authentication>
  </soap:Header>

  <soap:Body>
    <bank:GetAccountBalance>
      <bank:AccountNumber>9876543210</bank:AccountNumber>
    </bank:GetAccountBalance>
  </soap:Body>
</soap:Envelope>
```

🔍 What this means

- User authenticates with:
  - UserID
  - Token
- User requests account balance by providing:
  - AccountNumber

✅ 2) SOAP Response (Output)

The server responds with:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:bank="http://example.com/bank">
  <soap:Body>
    <bank:GetAccountBalanceResponse>
      <bank:AccountNumber>9876543210</bank:AccountNumber>
      <bank:Balance>1500.50</bank:Balance>
      <bank:Currency>USD</bank:Currency>
      <bank:Status>Success</bank:Status>
    </bank:GetAccountBalanceResponse>
  </soap:Body>
</soap:Envelope>
```

🔍 What this means

- Account number confirmed
- Balance returned: **$1500.50**
- Status: Success

✅ 3) Error Response (If account NOT found)

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Client</faultcode>
      <faultstring>Invalid Account Number</faultstring>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

✅ Explanation of the Example

Section	Meaning
Envelope	Wraps entire SOAP message
Header	Authentication info
Body	Contains request/response data
Fault	Error structure

✅ Flow

1. Client sends SOAP request → account number + authentication
2. Server validates credentials
3. Server looks up balance
4. Server returns SOAP XML response

✅ Realistic Use Case

Mobile banking app → SOAP → Bank system

Example scenario:

User opens bank app → checks balance → SOAP service returns balance

Works even if:

- Client app = Android (.NET)
- Server = Java backend

Because SOAP is language-independent.

If you want, I can provide another example:
✅ ATM money withdrawal
✅ Flight booking
✅ University course registration

✅ ✅ BIG SOAP EXAMPLE

✔ Use Case: Check Account Balance

A SOAP Web Service that accepts an account number and returns the balance.

✅ 1) SOAP REQUEST (INPUT)

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:bank="http://example.com/bank">

  <soap:Header>
    <bank:Authentication>
      <bank:UserID>USER123</bank:UserID>
      <bank:Token>ABCD1234</bank:Token>
    </bank:Authentication>
  </soap:Header>

  <soap:Body>
    <bank:GetAccountBalance>
      <bank:AccountNumber>9876543210</bank:AccountNumber>
    </bank:GetAccountBalance>
  </soap:Body>

</soap:Envelope>
```

✅ Meaning

- Header → authentication details
- Body → function + input parameters
- Function → `GetAccountBalance()`
- Parameter → AccountNumber

✅ 2) SOAP RESPONSE (OUTPUT)

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:bank="http://example.com/bank">
  <soap:Body>
    <bank:GetAccountBalanceResponse>
      <bank:AccountNumber>9876543210</bank:AccountNumber>
      <bank:Balance>1500.50</bank:Balance>
      <bank:Currency>USD</bank:Currency>
      <bank:Status>Success</bank:Status>
    </bank:GetAccountBalanceResponse>
  </soap:Body>
</soap:Envelope>
```

✅ Returns account balance
✅ Status = Success

✅ 3) SOAP FAULT (ERROR RESPONSE)

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Client</faultcode>
      <faultstring>Invalid Account Number</faultstring>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

✅ Returned when account not found

✅ 4) How SOAP Works (Concept)

```
CLIENT  ----SOAP-->  SERVER
```

```
        <--SOAP----
```

Steps

1. Client sends SOAP XML request
2. Server receives XML
3. Server extracts function name + parameters (**Demarshalling**)
4. Server executes business logic
5. Server returns SOAP XML response
6. Client parses XML (**Marshalling**)

✅ 5) SOAP Message Structure

```
<Envelope>     <-- Required
   <Header>    <-- Optional
   <Body>      <-- Required
</Envelope>
```

✅ 6) How to RUN SOAP Web Service

You can run SOAP using multiple environments:

✅ Using SOAP UI (easiest)
✅ Using Java
✅ Using .NET
✅ Using PHP

I will show the **easiest way first → SOAP UI**

✅ METHOD 1 — Using SOAP UI (Recommended)

✅ Step 1: Download SOAP UI

Website: **soapui.org**

✅ Step 2: Create New SOAP Project

- Open SOAP UI
- Click → **File → New SOAP Project**
- Enter WSDL URL (Example WSDL below)

```
http://example.com/bank?wsdl
```

✅ Step 3: Select `GetAccountBalance` operation

✅ Step 4: Enter SOAP Request XML (input)

Paste this:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:bank="http://example.com/bank">
  <soap:Body>
    <bank:GetAccountBalance>
      <bank:AccountNumber>9876543210</bank:AccountNumber>
    </bank:GetAccountBalance>
  </soap:Body>
</soap:Envelope>
```

✅ Step 5: Send Request → You get SOAP Response

Done ✅

✅ METHOD 2 — Run SOAP Web Service in Java

✅ Add Library

Use **JAX-WS** library

✅ Server (Provider) Example

```
import javax.jws.WebService;
import javax.jws.WebMethod;

@WebService
public class BankService {

    @WebMethod
```

```
    public double getAccountBalance(String accountNumber) {
        if(accountNumber.equals("9876543210"))
            return 1500.50;
        else
            return 0;
    }
}
```

Run using Java EE or Apache Tomcat / Glassfish.

✅ METHOD 3 — Run SOAP Web Service in .NET

✅ Web Service

```
[WebService]
public class BankService : System.Web.Services.WebService
{
    [WebMethod]
    public double GetAccountBalance(string accountNumber)
    {
        if(accountNumber == "9876543210")
            return 1500.50;
        return 0;
    }
}
```

ASP.NET will generate `WSDL` automatically.

✅ HOW TO KNOW IT WORKS

Client calls:

```
GetAccountBalance(9876543210)
```

Server checks DB:

```
Balance = 1500.50
```

Server returns SOAP XML response.

✅ 7) How the Example Works (End-to-End)

Step	Action
1	Client requests balance
2	SOAP request → server
3	Server reads XML
4	Server runs function
5	Looks up database
6	Returns SOAP response
7	Client displays balance

✅ Real Life Explanation

ATM → SOAP → Bank Server
Mobile Banking → SOAP → Bank
Online Payment → SOAP → Bank

SOAP ensures:
✔ Security
✔ Standard format
✔ Cross-language support

✅ Short Version

SOAP request → XML input
SOAP response → XML output
Can be tested with → SOAP UI

✅ If you want

I can also provide:
✅ Complete Java runnable project
✅ WSDL sample
✅ Database integration
✅ PHP SOAP Server + Client

1. Java SOAP Client + Server with Database Integration

a. Database Setup

We'll use **MySQL** for this example.

```
CREATE DATABASE soap_db;
USE soap_db;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(50)
);

INSERT INTO users (name, email) VALUES
('Alice', 'alice@example.com'),
('Bob', 'bob@example.com');
```

---

b. Java SOAP Server

UserService.java

```
package com.example.soap;

import javax.jws.WebMethod;
import javax.jws.WebService;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

@WebService
public class UserService {

    private Connection connect() throws SQLException {
        String url = "jdbc:mysql://localhost:3306/soap_db";
        String user = "root"; // your DB user
        String password = ""; // your DB password
        return DriverManager.getConnection(url, user, password);
    }
```

```
    @WebMethod
    public List<String> getAllUsers() {
        List<String> users = new ArrayList<>();
        try (Connection conn = connect()) {
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT name, email FROM
users");
            while (rs.next()) {
                users.add(rs.getString("name") + " - " +
rs.getString("email"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return users;
    }
}
```

Publisher.java

```
package com.example.soap;

import javax.xml.ws.Endpoint;

public class Publisher {
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/ws/users", new
UserService());
        System.out.println("SOAP Service running at
http://localhost:8080/ws/users?wsdl");
    }
}
```

---

c. WSDL Sample

Accessed automatically at:

```
http://localhost:8080/ws/users?wsdl
```

A sample WSDL generated by the above will look like:

```
<definitions name="UserService"
             targetNamespace="http://soap.example.com/"
             xmlns:tns="http://soap.example.com/"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
        <xsd:schema targetNamespace="http://soap.example.com/"/>
    </types>
```

```
    <message name="getAllUsersRequest"/>
    <message name="getAllUsersResponse">
        <part name="return" type="xsd:string"/>
    </message>

    <portType name="UserService">
        <operation name="getAllUsers">
            <input message="tns:getAllUsersRequest"/>
            <output message="tns:getAllUsersResponse"/>
        </operation>
    </portType>

    <binding name="UserServiceSoapBinding" type="tns:UserService">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="getAllUsers">
            <soap:operation soapAction=""/>
            <input><soap:body use="literal"/></input>
            <output><soap:body use="literal"/></output>
        </operation>
    </binding>

    <service name="UserService">
        <port name="UserServicePort" binding="tns:UserServiceSoapBinding">
            <soap:address location="http://localhost:8080/ws/users"/>
        </port>
    </service>
</definitions>
```

---

d. Java SOAP Client

```
package com.example.soap.client;

import com.example.soap.UserService;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import java.net.URL;
import java.util.List;

public class SOAPClient {
    public static void main(String[] args) throws Exception {
        URL wsdlURL = new URL("http://localhost:8080/ws/users?wsdl");
        QName qname = new QName("http://soap.example.com/", "UserService");
        Service service = Service.create(wsdlURL, qname);
        UserService userService = service.getPort(UserService.class);

        List<String> users = userService.getAllUsers();
        users.forEach(System.out::println);
    }
}
```

---

2. PHP SOAP Server + Client

a. PHP SOAP Server

```
<?php
// server.php
$server = new SoapServer(null, ['uri' => "http://localhost/soap"]);

$server->addFunction('getUsers');

function getUsers() {
    $pdo = new PDO("mysql:host=localhost;dbname=soap_db", "root", "");
    $stmt = $pdo->query("SELECT name, email FROM users");
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

$server->handle();
?>
```

Run this with:

```
php -S localhost:8081 server.php
```

---

b. PHP SOAP Client

```
<?php
// client.php
$client = new SoapClient(null, [
    'location' => "http://localhost:8081/server.php",
    'uri' => "http://localhost/soap",
    'trace' => 1
]);

$users = $client->__soapCall("getUsers", []);
print_r($users);
?>
```

✅ **This setup includes:**

1. **Java SOAP server and client**
2. **WSDL generation**
3. **Database integration with MySQL**
4.

1 Java SOAP Project (Maven)

Project Structure

```
java-soap-project/
├── pom.xml
└── src/
    └── main/
        ├── java/
        │   └── com/example/soap/
        │       ├── Publisher.java
        │       └── UserService.java
        └── resources/
```

---

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>java-soap-project</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <!-- JAX-WS -->
        <dependency>
            <groupId>javax.xml.ws</groupId>
            <artifactId>jaxws-api</artifactId>
            <version>2.3.1</version>
        </dependency>
        <!-- MySQL Connector -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.1.0</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>jaxws-maven-plugin</artifactId>
                <version>2.6</version>
            </plugin>
        </plugins>
    </build>
</project>
```

UserService.java

```
package com.example.soap;

import javax.jws.WebMethod;
import javax.jws.WebService;
import java.sql.*;
import java.util.ArrayList;
```

```
import java.util.List;

@WebService
public class UserService {
    private Connection connect() throws SQLException {
        return DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/soap_db", "root", ""
        );
    }

    @WebMethod
    public List<String> getAllUsers() {
        List<String> users = new ArrayList<>();
        try (Connection conn = connect()) {
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT name, email FROM
users");
            while (rs.next()) {
                users.add(rs.getString("name") + " - " +
rs.getString("email"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return users;
    }
}
```

Publisher.java

```
package com.example.soap;

import javax.xml.ws.Endpoint;

public class Publisher {
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/ws/users", new
UserService());
        System.out.println("SOAP Service running at
http://localhost:8080/ws/users?wsdl");
    }
}
```

Run Instructions (Java)

```
# 1. Build the project
mvn clean package

# 2. Run the SOAP server
mvn exec:java -Dexec.mainClass="com.example.soap.Publisher"

# Access WSDL:
# http://localhost:8080/ws/users?wsdl
```

2 PHP SOAP Project

Project Structure

```
php-soap-project/
├── server.php
├── client.php
└── db.sql
```

---

db.sql

```
CREATE DATABASE soap_db;
USE soap_db;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(50)
);

INSERT INTO users (name, email) VALUES
('Alice', 'alice@example.com'),
('Bob', 'bob@example.com');
```

---

server.php

```
<?php
$server = new SoapServer(null, ['uri' => "http://localhost/soap"]);
$server->addFunction('getUsers');

function getUsers() {
    $pdo = new PDO("mysql:host=localhost;dbname=soap_db", "root", "");
    $stmt = $pdo->query("SELECT name, email FROM users");
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

$server->handle();
?>
```

---

client.php

```
<?php
$client = new SoapClient(null, [
    'location' => "http://localhost:8081/server.php",
    'uri' => "http://localhost/soap",
    'trace' => 1
```

```
]);

$users = $client->__soapCall("getUsers", []);
print_r($users);
?>
```

Run Instructions (PHP)

```
# 1. Run PHP server
php -S localhost:8081 server.php

# 2. Run client
php client.php
```

✅ **Now you have:**

- Fully runnable **Java SOAP server + client with WSDL + MySQL**
- Fully runnable **PHP SOAP server + client with MySQL**

Complete GitHub Project Structure

```
soap-complete-project/
├── README.md
├── db/
│   └── db.sql                    # MySQL database setup
├── java-soap/
│   ├── pom.xml
│   └── src/
│       └── main/
│           └── java/
│               └── com/example/soap/
│                   ├── Publisher.java
│                   └── UserService.java
├── php-soap/
│   ├── server.php
│   └── client.php
```

1️⃣ README.md

```
# Complete SOAP Project

This repository contains a complete SOAP setup with both **Java** and **PHP**
clients and servers, including **database integration** with MySQL.

---
```

```
## Database Setup

```sql
-- Run this in MySQL
CREATE DATABASE soap_db;
USE soap_db;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(50)
);

INSERT INTO users (name, email) VALUES
('Alice', 'alice@example.com'),
('Bob', 'bob@example.com');
```

Java SOAP Project

Build & Run

```
cd java-soap
mvn clean package
mvn exec:java -Dexec.mainClass="com.example.soap.Publisher"
```

- Access WSDL: `http://localhost:8080/ws/users?wsdl`

Test Client

Add a Java SOAP client or use `wsimport` to generate one.

PHP SOAP Project

Run Server

```
cd php-soap
php -S localhost:8081 server.php
```

Run Client

```
php client.php
```

---

2 Database Script (`db/db.sql`)
```sql
```
```

```
CREATE DATABASE soap_db;
USE soap_db;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(50)
);

INSERT INTO users (name, email) VALUES
('Alice', 'alice@example.com'),
('Bob', 'bob@example.com');
```

---

3 Java SOAP (`java-soap/`)

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>java-soap-project</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>javax.xml.ws</groupId>
            <artifactId>jaxws-api</artifactId>
            <version>2.3.1</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.1.0</version>
        </dependency>
    </dependencies>
</project>
```

UserService.java

```
package com.example.soap;

import javax.jws.WebMethod;
import javax.jws.WebService;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

@WebService
public class UserService {
    private Connection connect() throws SQLException {
        return DriverManager.getConnection(
```

```
                "jdbc:mysql://localhost:3306/soap_db", "root", ""
        );
    }

    @WebMethod
    public List<String> getAllUsers() {
        List<String> users = new ArrayList<>();
        try (Connection conn = connect()) {
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT name, email FROM
users");
            while (rs.next()) {
                users.add(rs.getString("name") + " - " +
rs.getString("email"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return users;
    }
}
```

Publisher.java

```
package com.example.soap;

import javax.xml.ws.Endpoint;

public class Publisher {
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/ws/users", new
UserService());
        System.out.println("SOAP Service running at
http://localhost:8080/ws/users?wsdl");
    }
}
```

---

4 PHP SOAP (`php-soap/`)

server.php

```
<?php
$server = new SoapServer(null, ['uri' => "http://localhost/soap"]);
$server->addFunction('getUsers');

function getUsers() {
    $pdo = new PDO("mysql:host=localhost;dbname=soap_db", "root", "");
    $stmt = $pdo->query("SELECT name, email FROM users");
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

$server->handle();
?>
```

client.php

```
<?php
$client = new SoapClient(null, [
    'location' => "http://localhost:8081/server.php",
    'uri' => "http://localhost/soap",
    'trace' => 1
]);

$users = $client->__soapCall("getUsers", []);
print_r($users);
?>
```

✅ Features

- Java SOAP server + client + WSDL
- PHP SOAP server + client
- MySQL integration
- Ready-to-run Maven project for Java
- Single-folder PHP SOAP project

Got it! Let's create a **more advanced SOAP example** that is **enterprise-grade**, integrating:

- **Java SOAP Server with JPA / Hibernate for database integration**
- **MySQL**
- **CRUD operations (Create, Read, Update, Delete)**
- **Java SOAP Client with proper DTOs**
- **PHP SOAP client to consume the Java SOAP service**
- **WSDL exposure and complex types**
- **Validation and error handling**

This is a step up from the previous simple "list users" example.

1️⃣ Database Setup (MySQL)

```
CREATE DATABASE advanced_soap_db;
USE advanced_soap_db;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    age INT
```

```
);

INSERT INTO users (name, email, age) VALUES
('Alice', 'alice@example.com', 25),
('Bob', 'bob@example.com', 30);
```

---

2 Java SOAP Project (Maven + JPA/Hibernate)

Project Structure

```
advanced-java-soap/
├── pom.xml
└── src/main/java/com/example/soap/
    ├── Publisher.java
    ├── UserService.java
    ├── dto/
    │   └── UserDTO.java
    ├── entity/
    │   └── User.java
    └── dao/
        └── UserDAO.java
```

---

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>advanced-java-soap</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <!-- SOAP -->
        <dependency>
            <groupId>javax.xml.ws</groupId>
            <artifactId>jaxws-api</artifactId>
            <version>2.3.1</version>
        </dependency>

        <!-- JPA / Hibernate -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.6.15.Final</version>
        </dependency>
        <dependency>
```

```
            <groupId>javax.persistence</groupId>
            <artifactId>javax.persistence-api</artifactId>
            <version>2.2</version>
        </dependency>

        <!-- MySQL -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.1.0</version>
        </dependency>

        <!-- Logging -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>2.0.9</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>2.0.9</version>
        </dependency>
    </dependencies>
</project>
```

User Entity (JPA)

```
package com.example.soap.entity;

import javax.persistence.*;

@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String name;

    @Column(unique = true)
    private String email;

    private int age;

    // Getters & Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
```

```
}
```

UserDTO (SOAP Data Transfer Object)

```
package com.example.soap.dto;

import java.io.Serializable;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class UserDTO implements Serializable {
    private int id;
    private String name;
    private String email;
    private int age;

    public UserDTO() {}
    public UserDTO(int id, String name, String email, int age) {
        this.id = id;
        this.name = name;
        this.email = email;
        this.age = age;
    }

    // Getters & Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
}
```

UserDAO

```
package com.example.soap.dao;

import com.example.soap.entity.User;
import javax.persistence.*;
import java.util.List;

public class UserDAO {
    private static final EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UserPU");

    public List<User> getAllUsers() {
        EntityManager em = emf.createEntityManager();
        List<User> users = em.createQuery("SELECT u FROM User u",
User.class).getResultList();
        em.close();
        return users;
```

```
    }

    public User getUserById(int id) {
        EntityManager em = emf.createEntityManager();
        User user = em.find(User.class, id);
        em.close();
        return user;
    }

    public void createUser(User user) {
        EntityManager em = emf.createEntityManager();
        em.getTransaction().begin();
        em.persist(user);
        em.getTransaction().commit();
        em.close();
    }

    public void updateUser(User user) {
        EntityManager em = emf.createEntityManager();
        em.getTransaction().begin();
        em.merge(user);
        em.getTransaction().commit();
        em.close();
    }

    public void deleteUser(int id) {
        EntityManager em = emf.createEntityManager();
        User user = em.find(User.class, id);
        if(user != null){
            em.getTransaction().begin();
            em.remove(user);
            em.getTransaction().commit();
        }
        em.close();
    }
}
```

UserService (SOAP Service)

```
package com.example.soap;

import com.example.soap.dao.UserDAO;
import com.example.soap.dto.UserDTO;
import com.example.soap.entity.User;

import javax.jws.WebMethod;
import javax.jws.WebService;
import java.util.List;
import java.util.stream.Collectors;

@WebService
public class UserService {
    private final UserDAO userDAO = new UserDAO();

    @WebMethod
```

```
    public List<UserDTO> getAllUsers() {
        return userDAO.getAllUsers().stream()
                .map(u -> new UserDTO(u.getId(), u.getName(), u.getEmail(),
u.getAge()))
                .collect(Collectors.toList());
    }

    @WebMethod
    public UserDTO getUser(int id) {
        User u = userDAO.getUserById(id);
        if(u == null) return null;
        return new UserDTO(u.getId(), u.getName(), u.getEmail(), u.getAge());
    }

    @WebMethod
    public void createUser(UserDTO dto) {
        User u = new User();
        u.setName(dto.getName());
        u.setEmail(dto.getEmail());
        u.setAge(dto.getAge());
        userDAO.createUser(u);
    }

    @WebMethod
    public void updateUser(UserDTO dto) {
        User u = new User();
        u.setId(dto.getId());
        u.setName(dto.getName());
        u.setEmail(dto.getEmail());
        u.setAge(dto.getAge());
        userDAO.updateUser(u);
    }

    @WebMethod
    public void deleteUser(int id) {
        userDAO.deleteUser(id);
    }
}
```

Publisher

```
package com.example.soap;

import javax.xml.ws.Endpoint;

public class Publisher {
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/ws/users", new
UserService());
        System.out.println("Advanced SOAP Service running at
http://localhost:8080/ws/users?wsdl");
    }
}
```

3️⃣ PHP SOAP Client Example

```
<?php
$client = new SoapClient(null, [
    'location' => "http://localhost:8080/ws/users",
    'uri' => "http://soap.example.com/",
    'trace' => 1
]);

// Get all users
$users = $client->__soapCall("getAllUsers", []);
print_r($users);

// Create new user
$newUser = ['name'=>'Charlie','email'=>'charlie@example.com','age'=>28];
$client->__soapCall("createUser", [$newUser]);
echo "User created!\n";
?>
```

✅ Advanced Features

- CRUD operations (Create, Read, Update, Delete) via SOAP
- JPA / Hibernate database layer
- Complex types (`UserDTO`) with JAXB XML serialization
- Enterprise-grade Java SOAP server
- PHP client integration

Perfect! Let's take this **advanced SOAP example** and make it **even more professional and production-ready** with additional features that real enterprise services often require:

We'll focus on:

- **REST-like CRUD operations via SOAP** (full CRUD + query by email)
- **Input validation & error handling**
- **Logging** with SLF4J
- **DTO separation** for SOAP vs DB entities
- **Complex WSDL with nested types**
- **Transaction management**
- **Pagination for listing users**
- **PHP client capable of calling all operations**

1️⃣ Database Update (MySQL)

```
CREATE DATABASE advanced_soap_db;
USE advanced_soap_db;
```

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    age INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO users (name, email, age) VALUES
('Alice', 'alice@example.com', 25),
('Bob', 'bob@example.com', 30),
('Charlie', 'charlie@example.com', 28);
```

---

2 Java SOAP Project Updates

a. Updated UserDTO

```
package com.example.soap.dto;

import java.io.Serializable;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "User")
@XmlType(propOrder = {"id","name","email","age","createdAt"})
public class UserDTO implements Serializable {
    private int id;
    private String name;
    private String email;
    private int age;
    private String createdAt;

    public UserDTO() {}
    public UserDTO(int id, String name, String email, int age, String
createdAt) {
        this.id = id;
        this.name = name;
        this.email = email;
        this.age = age;
        this.createdAt = createdAt;
    }

    // Getters & Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
```

```
    public String getCreatedAt() { return createdAt; }
    public void setCreatedAt(String createdAt) { this.createdAt = createdAt;
}
}
```

---

b. Updated UserDAO with Pagination & Email Query

```
package com.example.soap.dao;

import com.example.soap.entity.User;
import javax.persistence.*;
import java.util.List;

public class UserDAO {
    private static final EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UserPU");

    public List<User> getAllUsers(int page, int size) {
        EntityManager em = emf.createEntityManager();
        List<User> users = em.createQuery("SELECT u FROM User u ORDER BY
u.id", User.class)
                             .setFirstResult((page-1)*size)
                             .setMaxResults(size)
                             .getResultList();
        em.close();
        return users;
    }

    public User getUserById(int id) {
        EntityManager em = emf.createEntityManager();
        User u = em.find(User.class, id);
        em.close();
        return u;
    }

    public User getUserByEmail(String email) {
        EntityManager em = emf.createEntityManager();
        try {
            return em.createQuery("SELECT u FROM User u WHERE
u.email=:email", User.class)
                     .setParameter("email", email)
                     .getSingleResult();
        } catch(NoResultException e) {
            return null;
        } finally {
            em.close();
        }
    }

    public void createUser(User user) {
        EntityManager em = emf.createEntityManager();
        em.getTransaction().begin();
        em.persist(user);
        em.getTransaction().commit();
```

```
        em.close();
    }

    public void updateUser(User user) {
        EntityManager em = emf.createEntityManager();
        em.getTransaction().begin();
        em.merge(user);
        em.getTransaction().commit();
        em.close();
    }

    public void deleteUser(int id) {
        EntityManager em = emf.createEntityManager();
        User u = em.find(User.class, id);
        if(u != null){
            em.getTransaction().begin();
            em.remove(u);
            em.getTransaction().commit();
        }
        em.close();
    }
}
```

---

c. Updated UserService

```
package com.example.soap;

import com.example.soap.dao.UserDAO;
import com.example.soap.dto.UserDTO;
import com.example.soap.entity.User;

import javax.jws.WebMethod;
import javax.jws.WebService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.util.List;
import java.util.stream.Collectors;

@WebService
public class UserService {
    private final UserDAO userDAO = new UserDAO();
    private static final Logger logger =
LoggerFactory.getLogger(UserService.class);

    @WebMethod
    public List<UserDTO> getAllUsers(int page, int size) {
        logger.info("Fetching users page={} size={}", page, size);
        return userDAO.getAllUsers(page, size).stream()
                .map(u -> new UserDTO(u.getId(), u.getName(), u.getEmail(),
u.getAge(), u.getCreatedAt().toString()))
                .collect(Collectors.toList());
    }

    @WebMethod
```

```
    public UserDTO getUserById(int id) {
        User u = userDAO.getUserById(id);
        if(u == null) {
            logger.warn("User id={} not found", id);
            return null;
        }
        return new UserDTO(u.getId(), u.getName(), u.getEmail(), u.getAge(),
u.getCreatedAt().toString());
    }

    @WebMethod
    public UserDTO getUserByEmail(String email) {
        User u = userDAO.getUserByEmail(email);
        if(u == null) {
            logger.warn("User email={} not found", email);
            return null;
        }
        return new UserDTO(u.getId(), u.getName(), u.getEmail(), u.getAge(),
u.getCreatedAt().toString());
    }

    @WebMethod
    public void createUser(UserDTO dto) throws Exception {
        if(dto.getName() == null || dto.getEmail() == null) {
            throw new Exception("Name and email cannot be null");
        }
        logger.info("Creating user {}", dto.getEmail());
        User u = new User();
        u.setName(dto.getName());
        u.setEmail(dto.getEmail());
        u.setAge(dto.getAge());
        userDAO.createUser(u);
    }

    @WebMethod
    public void updateUser(UserDTO dto) throws Exception {
        if(dto.getId() <= 0) throw new Exception("Invalid ID");
        logger.info("Updating user id={}", dto.getId());
        User u = new User();
        u.setId(dto.getId());
        u.setName(dto.getName());
        u.setEmail(dto.getEmail());
        u.setAge(dto.getAge());
        userDAO.updateUser(u);
    }

    @WebMethod
    public void deleteUser(int id) throws Exception {
        logger.info("Deleting user id={}", id);
        userDAO.deleteUser(id);
    }
}
```

---

d. Publisher.java

```
package com.example.soap;

import javax.xml.ws.Endpoint;

public class Publisher {
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/ws/users", new
UserService());
        System.out.println("Advanced SOAP Service running at
http://localhost:8080/ws/users?wsdl");
    }
}
```

---

3 PHP SOAP Client Example (Full CRUD)

```
<?php
$client = new SoapClient(null, [
    'location' => "http://localhost:8080/ws/users",
    'uri' => "http://soap.example.com/",
    'trace' => 1
]);

// Create new user
$newUser = ['name'=>'David','email'=>'david@example.com','age'=>40];
$client->__soapCall("createUser", [$newUser]);
echo "User created!\n";

// Get all users (page=1, size=10)
$users = $client->__soapCall("getAllUsers", [1, 10]);
print_r($users);

// Get user by email
$user = $client->__soapCall("getUserByEmail", ["alice@example.com"]);
print_r($user);

// Update user
$updateUser = ['id'=>1,'name'=>'Alice
Updated','email'=>'alice@example.com','age'=>26];
$client->__soapCall("updateUser", [$updateUser]);
echo "User updated!\n";

// Delete user
$client->__soapCall("deleteUser", [3]);
echo "User deleted!\n";
?>
```

✅ New Features Added

1. Full **CRUD** + query by email
2. **Pagination** for listing users
3. **Error handling & input validation**

4. **Logging with SLF4J**
5. **DTO separation for SOAP**
6. **Complex WSDL with nested types**
7. Transaction-safe database operations

Enterprise SOAP Project Structure

```
advanced-soap-project/
├── README.md
├── db/
│   └── db.sql                  # MySQL setup script
├── java-soap/
│   ├── pom.xml
│   └── src/main/java/com/example/soap/
│       ├── Publisher.java
│       ├── UserService.java
│       ├── dao/
│       │   └── UserDAO.java
│       ├── dto/
│       │   └── UserDTO.java
│       └── entity/
│           └── User.java
└── php-soap/
    ├── client.php
    └── README.md
```

---

1 Database Setup (`db/db.sql`)

```
CREATE DATABASE advanced_soap_db;
USE advanced_soap_db;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    age INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO users (name, email, age) VALUES
('Alice', 'alice@example.com', 25),
('Bob', 'bob@example.com', 30),
('Charlie', 'charlie@example.com', 28);
```

---

2 Java SOAP Project (`java-soap/`)

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>advanced-java-soap</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>javax.xml.ws</groupId>
            <artifactId>jaxws-api</artifactId>
            <version>2.3.1</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.6.15.Final</version>
        </dependency>
        <dependency>
            <groupId>javax.persistence</groupId>
            <artifactId>javax.persistence-api</artifactId>
            <version>2.2</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.1.0</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>2.0.9</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>2.0.9</version>
        </dependency>
    </dependencies>
</project>
```

Publisher.java

```
package com.example.soap;

import javax.xml.ws.Endpoint;

public class Publisher {
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/ws/users", new
UserService());
        System.out.println("Enterprise SOAP Service running at
http://localhost:8080/ws/users?wsdl");
    }
```

```
}
```

UserService.java, UserDAO.java, UserDTO.java, User.java

- Full CRUD with pagination, query by email, logging, error handling (as provided in previous step).

---

3 PHP SOAP Client (`php-soap/client.php`)

```
<?php
$client = new SoapClient(null, [
    'location' => "http://localhost:8080/ws/users",
    'uri' => "http://soap.example.com/",
    'trace' => 1
]);

// CRUD operations
$newUser = ['name'=>'David','email'=>'david@example.com','age'=>40];
$client->__soapCall("createUser", [$newUser]);

$users = $client->__soapCall("getAllUsers", [1, 10]);
print_r($users);

$user = $client->__soapCall("getUserByEmail", ["alice@example.com"]);
print_r($user);

$updateUser = ['id'=>1,'name'=>'Alice
Updated','email'=>'alice@example.com','age'=>26];
$client->__soapCall("updateUser", [$updateUser]);

$client->__soapCall("deleteUser", [3]);
echo "Operations completed!\n";
?>
```

---

4 README.md (Project Instructions)

````
# Enterprise SOAP Project

## Setup Database
```bash
mysql -u root -p < db/db.sql

# Java SOAP

cd java-soap
mvn clean package
mvn exec:java -Dexec.mainClass="com.example.soap.Publisher"
````
````

- WSDL: `http://localhost:8080/ws/users?wsdl`

PHP SOAP Client

```
cd php-soap
php client.php

---

## **5 Package as ZIP**
From your project root:

```bash
zip -r advanced-soap-project.zip advanced-soap-project/
```

- This ZIP contains **Java SOAP server**, **PHP client**, and **DB setup script**.
- Can be uploaded directly to GitHub or shared for deployment.

1 Full Folder Structure

```
advanced-soap-project/
├── README.md
├── db/
│   └── db.sql
├── java-soap/
│   ├── pom.xml
│   └── src/main/java/com/example/soap/
│       ├── Publisher.java
│       ├── UserService.java
│       ├── dao/
│       │   └── UserDAO.java
│       ├── dto/
│       │   └── UserDTO.java
│       └── entity/
│           └── User.java
│   └── src/main/resources/META-INF/
│       └── persistence.xml
└── php-soap/
    └── client.php
```

---

2 Database Script (`db/db.sql`)

```
CREATE DATABASE IF NOT EXISTS advanced_soap_db;
USE advanced_soap_db;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
```
```

```
    age INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO users (name, email, age) VALUES
('Alice', 'alice@example.com', 25),
('Bob', 'bob@example.com', 30),
('Charlie', 'charlie@example.com', 28);
```

---

3 Java Project

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>advanced-java-soap</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>javax.xml.ws</groupId>
            <artifactId>jaxws-api</artifactId>
            <version>2.3.1</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.6.15.Final</version>
        </dependency>
        <dependency>
            <groupId>javax.persistence</groupId>
            <artifactId>javax.persistence-api</artifactId>
            <version>2.2</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.1.0</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>2.0.9</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>2.0.9</version>
        </dependency>
```

```
    </dependencies>
</project>
```

---

persistence.xml (`src/main/resources/META-INF/persistence.xml`)

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
             http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd"
             version="2.2">

    <persistence-unit name="UserPU">
        <class>com.example.soap.entity.User</class>
        <properties>
            <property name="javax.persistence.jdbc.driver"
value="com.mysql.cj.jdbc.Driver"/>
            <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/advanced_soap_db"/>
            <property name="javax.persistence.jdbc.user" value="root"/>
            <property name="javax.persistence.jdbc.password" value=""/>
            <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQL8Dialect"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.show_sql" value="true"/>
        </properties>
    </persistence-unit>
</persistence>
```

User Entity (`entity/User.java`)

```
package com.example.soap.entity;

import javax.persistence.*;
import java.sql.Timestamp;

@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String name;

    @Column(unique = true)
    private String email;

    private int age;

    @Column(name = "created_at", insertable=false, updatable=false)
    private Timestamp createdAt;

    // Getters and Setters
```

```
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
    public Timestamp getCreatedAt() { return createdAt; }
}
```

UserDTO (`dto/UserDTO.java`)

```
package com.example.soap.dto;

import java.io.Serializable;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class UserDTO implements Serializable {
    private int id;
    private String name;
    private String email;
    private int age;
    private String createdAt;

    public UserDTO() {}
    public UserDTO(int id, String name, String email, int age, String
createdAt) {
        this.id=id; this.name=name; this.email=email; this.age=age;
this.createdAt=createdAt;
    }

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id=id; }
    public String getName() { return name; }
    public void setName(String name) { this.name=name; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email=email; }
    public int getAge() { return age; }
    public void setAge(int age) { this.age=age; }
    public String getCreatedAt() { return createdAt; }
    public void setCreatedAt(String createdAt) { this.createdAt=createdAt; }
}
```

UserDAO (`dao/UserDAO.java`)

```
package com.example.soap.dao;

import com.example.soap.entity.User;
import javax.persistence.*;
import java.util.List;
```

```
public class UserDAO {
    private static final EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UserPU");

    public List<User> getAllUsers(int page, int size){
        EntityManager em = emf.createEntityManager();
        List<User> users = em.createQuery("SELECT u FROM User u ORDER BY
u.id", User.class)
                .setFirstResult((page-1)*size)
                .setMaxResults(size)
                .getResultList();
        em.close();
        return users;
    }

    public User getUserById(int id){
        EntityManager em = emf.createEntityManager();
        User u = em.find(User.class, id);
        em.close();
        return u;
    }

    public User getUserByEmail(String email){
        EntityManager em = emf.createEntityManager();
        try {
            return em.createQuery("SELECT u FROM User u WHERE
u.email=:email", User.class)
                    .setParameter("email", email).getSingleResult();
        } catch(NoResultException e){ return null; }
        finally{ em.close(); }
    }

    public void createUser(User u){
        EntityManager em = emf.createEntityManager();
        em.getTransaction().begin();
        em.persist(u);
        em.getTransaction().commit();
        em.close();
    }

    public void updateUser(User u){
        EntityManager em = emf.createEntityManager();
        em.getTransaction().begin();
        em.merge(u);
        em.getTransaction().commit();
        em.close();
    }

    public void deleteUser(int id){
        EntityManager em = emf.createEntityManager();
        User u = em.find(User.class, id);
        if(u!=null){
            em.getTransaction().begin();
            em.remove(u);
            em.getTransaction().commit();
        }
```

```
        em.close();
    }
}
```

UserService (`UserService.java`)

```
package com.example.soap;

import com.example.soap.dao.UserDAO;
import com.example.soap.dto.UserDTO;
import com.example.soap.entity.User;

import javax.jws.WebMethod;
import javax.jws.WebService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.util.List;
import java.util.stream.Collectors;

@WebService
public class UserService {
    private final UserDAO userDAO = new UserDAO();
    private static final Logger logger =
LoggerFactory.getLogger(UserService.class);

    @WebMethod
    public List<UserDTO> getAllUsers(int page, int size){
        logger.info("Fetching users page={} size={}", page, size);
        return userDAO.getAllUsers(page, size).stream()
                .map(u->new UserDTO(u.getId(), u.getName(), u.getEmail(),
u.getAge(), u.getCreatedAt().toString()))
                .collect(Collectors.toList());
    }

    @WebMethod
    public UserDTO getUserById(int id){
        User u=userDAO.getUserById(id);
        if(u==null){ logger.warn("User id={} not found", id); return null; }
        return new UserDTO(u.getId(), u.getName(), u.getEmail(), u.getAge(),
u.getCreatedAt().toString());
    }

    @WebMethod
    public UserDTO getUserByEmail(String email){
        User u=userDAO.getUserByEmail(email);
        if(u==null){ logger.warn("User email={} not found", email); return
null; }
        return new UserDTO(u.getId(), u.getName(), u.getEmail(), u.getAge(),
u.getCreatedAt().toString());
    }

    @WebMethod
    public void createUser(UserDTO dto) throws Exception {
        if(dto.getName()==null || dto.getEmail()==null) throw new
Exception("Name and email cannot be null");
```

```
        logger.info("Creating user {}", dto.getEmail());
        User u=new User(); u.setName(dto.getName());
u.setEmail(dto.getEmail()); u.setAge(dto.getAge());
        userDAO.createUser(u);
    }

    @WebMethod
    public void updateUser(UserDTO dto) throws Exception {
        if(dto.getId()<=0) throw new Exception("Invalid ID");
        logger.info("Updating user id={}", dto.getId());
        User u=new User(); u.setId(dto.getId()); u.setName(dto.getName());
u.setEmail(dto.getEmail()); u.setAge(dto.getAge());
        userDAO.updateUser(u);
    }

    @WebMethod
    public void deleteUser(int id) throws Exception{
        logger.info("Deleting user id={}", id);
        userDAO.deleteUser(id);
    }
}
```

Publisher (`Publisher.java`)

```
package com.example.soap;

import javax.xml.ws.Endpoint;

public class Publisher {
    public static void main(String[] args){
        Endpoint.publish("http://localhost:8080/ws/users", new
UserService());
        System.out.println("Enterprise SOAP Service running at
http://localhost:8080/ws/users?wsdl");
    }
}
```

---

4 PHP Client (`php-soap/client.php`)

```
<?php
$client = new SoapClient(null, [
    'location' => "http://localhost:8080/ws/users",
    'uri' => "http://soap.example.com/",
    'trace' => 1
]);

// Create new user
$client->__soapCall("createUser",
[['name'=>'David','email'=>'david@example.com','age'=>40]]);
echo "User created!\n";
```

```
// Get all users page=1, size=10
$users = $client->__soapCall("getAllUsers", [1,10]);
print_r($users);

// Get user by email
$user = $client->__soapCall("getUserByEmail", ["alice@example.com"]);
print_r($user);

// Update user
$client->__soapCall("updateUser", [['id'=>1,'name'=>'Alice
Updated','email'=>'alice@example.com','age'=>26]]);
echo "User updated!\n";

// Delete user
$client->__soapCall("deleteUser", [3]);
echo "User deleted!\n";
?>
```

---

5 How to Run

1. **Import database**

```
mysql -u root -p < db/db.sql
```

2. **Run Java SOAP server**

```
cd java-soap
mvn clean package
mvn exec:java -Dexec.mainClass="com.example.soap.Publisher"
```

3. **Run PHP client**

```
cd php-soap
php client.php
```

- Access WSDL: `http://localhost:8080/ws/users?wsdl`

---

6 Package as ZIP

From project root:

```
zip -r advanced-soap-project.zip advanced-soap-project/
```

✅ The ZIP contains **Java SOAP server, PHP client, DB script**, fully ready-to-run.

GitHub Repository Layout

```
advanced-soap-project/
├── README.md
├── db/
│   └── db.sql                          # MySQL database setup
├── java-soap/
│   ├── pom.xml
│   └── src/main/
│       ├── java/com/example/soap/
│       │   ├── Publisher.java
│       │   ├── UserService.java
│       │   ├── dao/UserDAO.java
│       │   ├── dto/UserDTO.java
│       │   └── entity/User.java
│       └── resources/META-INF/
│           └── persistence.xml
└── php-soap/
    └── client.php
```

---

1 README.md

````
# Advanced Enterprise SOAP Project

This repository contains a full **enterprise-level SOAP service** implemented
in Java with **JPA/Hibernate** and **MySQL**, along with a **PHP client**.

## Features
- CRUD operations (Create, Read, Update, Delete)
- Query by ID or Email
- Pagination for listing users
- DTO separation (SOAP vs DB entities)
- Logging (SLF4J)
- Input validation & error handling
- WSDL exposure
- PHP client capable of calling all operations

---

## Setup

### 1. Import Database
```bash
mysql -u root -p < db/db.sql
````

2. Run Java SOAP Server

```
cd java-soap
mvn clean package
mvn exec:java -Dexec.mainClass="com.example.soap.Publisher"
```
```

- Access WSDL: http://localhost:8080/ws/users?wsdl

3. Run PHP Client

```
cd php-soap
php client.php

---

## **2 Database (`db/db.sql`)**
```sql
CREATE DATABASE IF NOT EXISTS advanced_soap_db;
USE advanced_soap_db;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    age INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO users (name, email, age) VALUES
('Alice', 'alice@example.com', 25),
('Bob', 'bob@example.com', 30),
('Charlie', 'charlie@example.com', 28);
```

---

3 Java SOAP Project (`java-soap/`)

- **pom.xml**, **Publisher.java**, **UserService.java**, **UserDAO.java**, **UserDTO.java**, **User.java**, **persistence.xml** — all pre-filled as in the previous step.

Everything is ready-to-run, Maven will build, and service starts at `http://localhost:8080/ws/users`.

---

4 PHP SOAP Client (`php-soap/client.php`)

```
<?php
$client = new SoapClient(null, [
    'location' => "http://localhost:8080/ws/users",
    'uri' => "http://soap.example.com/",
    'trace' => 1
]);

// Create, read, update, delete operations
$client->__soapCall("createUser",
[['name'=>'David','email'=>'david@example.com','age'=>40]]);
$users = $client->__soapCall("getAllUsers", [1,10]);
```
```

```
print_r($users);
$user = $client->__soapCall("getUserByEmail", ["alice@example.com"]);
print_r($user);
$client->__soapCall("updateUser", [['id'=>1,'name'=>'Alice
Updated','email'=>'alice@example.com','age'=>26]]);
$client->__soapCall("deleteUser", [3]);
echo "All operations completed!\n";
?>
```

5️⃣ How to Use

1. Clone the repository:

```
git clone <your-repo-url>
cd advanced-soap-project
```

2. Import MySQL database:

```
mysql -u root -p < db/db.sql
```

3. Run Java SOAP server:

```
cd java-soap
mvn clean package
mvn exec:java -Dexec.mainClass="com.example.soap.Publisher"
```

4. Run PHP client:

```
cd php-soap
php client.php
```

✅ Outcome

- Fully functional **SOAP service with MySQL database integration**
- **CRUD operations** exposed via SOAP
- **PHP client** that consumes the service
- **Ready-to-deploy GitHub repository structure**

1️⃣ Create the folder structure

On your computer, create a folder like this:

```
advanced-soap-project/
├── README.md
├── db/db.sql
```

```
├── java-soap/pom.xml
├── java-soap/src/main/java/com/example/soap/
│   ├── Publisher.java
│   ├── UserService.java
│   ├── dao/UserDAO.java
│   ├── dto/UserDTO.java
│   └── entity/User.java
├── java-soap/src/main/resources/META-INF/persistence.xml
└── php-soap/client.php
```

---

2 Copy the content

- Use all the content I gave you for **README.md, db.sql, pom.xml, persistence.xml, Java files, PHP client**.
- Make sure the Java package folder matches `com/example/soap/`.

---

3 Generate the ZIP locally

Open a terminal in the parent folder of `advanced-soap-project` and run:

```
zip -r advanced-soap-project.zip advanced-soap-project/
```

- This will create `advanced-soap-project.zip` containing all your files and folders.
- You can then upload it to GitHub or share it directly.

---

4 Quick Test

1. Import the database:

```
mysql -u root -p < advanced-soap-project/db/db.sql
```

2. Run the Java SOAP server:

```
cd advanced-soap-project/java-soap
mvn clean package
mvn exec:java -Dexec.mainClass="com.example.soap.Publisher"
```

3. Run the PHP client:

```
cd ../php-soap
php client.php
```