Assignment Title:

- **Service-Oriented Architecture (SOA)**
- **Exploring Distributed System Communication: RPC/RMI and REST Architecture**

Learning Outcomes

After completing this assignment, students will be able to:

1. Understand and implement two architectural styles — **Service-Oriented Architecture (SOA)**
2. Implement **Remote Procedure Call (RPC)** or **Remote Method Invocation (RMI)** communication between distributed components.
3. Design and implement a **RESTful API** using a *Book Management System* as the resource domain.
4. Compare **SOA, RPC/RMI** and **REST** architectures in terms of communication style, scalability, and coupling.

Task 1: Service-Oriented Architecture (SOA) – Client & Server Communication
Goal: Develop two independent services — a **Client Service** and a **Server Service** — that communicate through messages.
Instructions:

1. Use **Python** or **Java** to implement the services.
2. The **Client Service** should send a message:
   "I am Client".
3. The **Server Service** should respond:
   "I am Server".
4. Demonstrate a small two-way dialogue between the client and the server.
5. Use **socket programming** or **RESTful API communication** (e.g., Flask in Python or Spring Boot/HTTPServer in Java).

Expected Output Example:
Client: I am Client.
Server: I am Server.
Client: Nice to meet you!
Server: Nice to meet you too!

Hint:

- In **Python**, you can use the socket module or Flask.
- In **Java**, you can use ServerSocket and Socket classes.

Part 2: Remote Communication using RPC or RMI

Objective: To implement a simple **client-server communication** using **RPC (Python)** or **RMI (Java)**, where the client calls a remote method hosted by the server.

Task Description:

1. Create a **Server** that exposes a remote method, e.g.,
   greet(name) → returns "Hello <name>, this is the server!"
2. Create a **Client** that:
   - Connects to the remote server.
   - Calls the greet() method.
   - Displays the server's response.
3. Demonstrate the RPC or RMI interaction.

Expected Output Example:
Client Output:
Requesting remote method...
Response from Server: Hello Ali, this is the server!

Part 3: REST Architecture — Book Management System

Objective: To design a **RESTful API** using **Python (Flask)** or **Java (Spring Boot)** to manage a collection of *Book* resources.

Task Description:

1. Create a REST API server that manages **Book resources**.
2. Each book should have the following attributes:
   - id (integer)
   - title (string)
   - author (string)
   - price (float)

3. Implement the following REST operations:
    - **GET** /books → Retrieve all books
    - **GET** /books/<id> → Retrieve one book
    - **POST** /books → Add a new book
    - **PUT** /books/<id> → Update a book
    - **DELETE** /books/<id> → Delete a book
4. Test the API using Postman or a web browser.

Expected Output Example:

GET /books

{

"1": {"title": "Distributed Systems", "author": "Tanenbaum", "price": 50.0},

"2": {"title": "Clean Code", "author": "Robert C. Martin", "price": 45.0}

}

POST /books

{"message": "Book added", "id": 3}

Deliverables:

1. Provide a report document explaining the architectures of all the above three tasks, and the comparison of the technologies.
2. Provide source code the above three tasks.