# Understanding REST and SOAP APIs

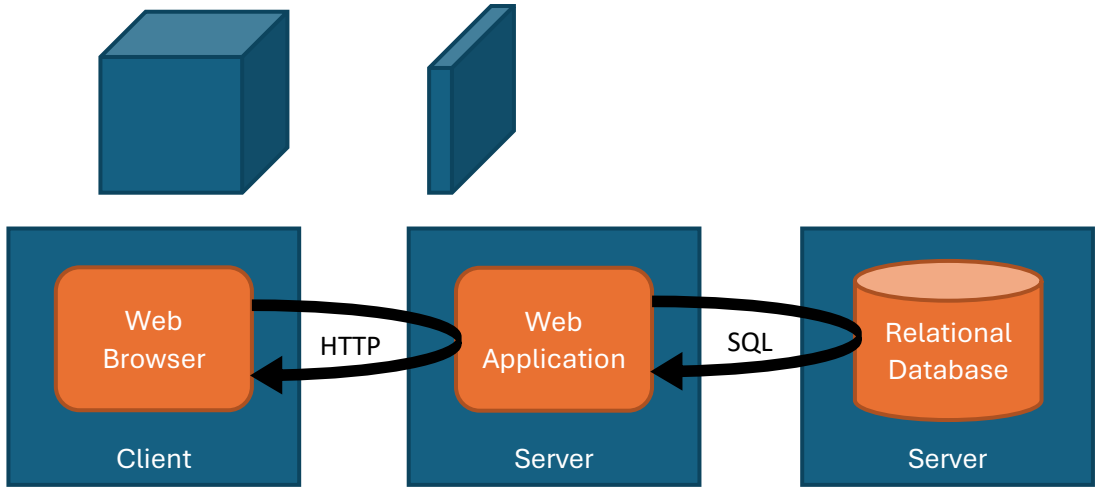
A Comparative Overview of Two Web Service Architectures

# Understanding REST and SOAP APIs

- **Recap of Chapter 2:**
  - We previously explored the various types of APIs and the data formats we use to facilitate communication between systems.
- **Chapter Focus:**
  - This chapter offers a concise overview of REST and SOAP APIs. We will explore the fundamental principles of each, their respective HTTP methods and protocols, URL structures, response codes, and key design concepts. Additionally, we will compare their strengths, use cases, and when to choose one over the other.

# What is REST?

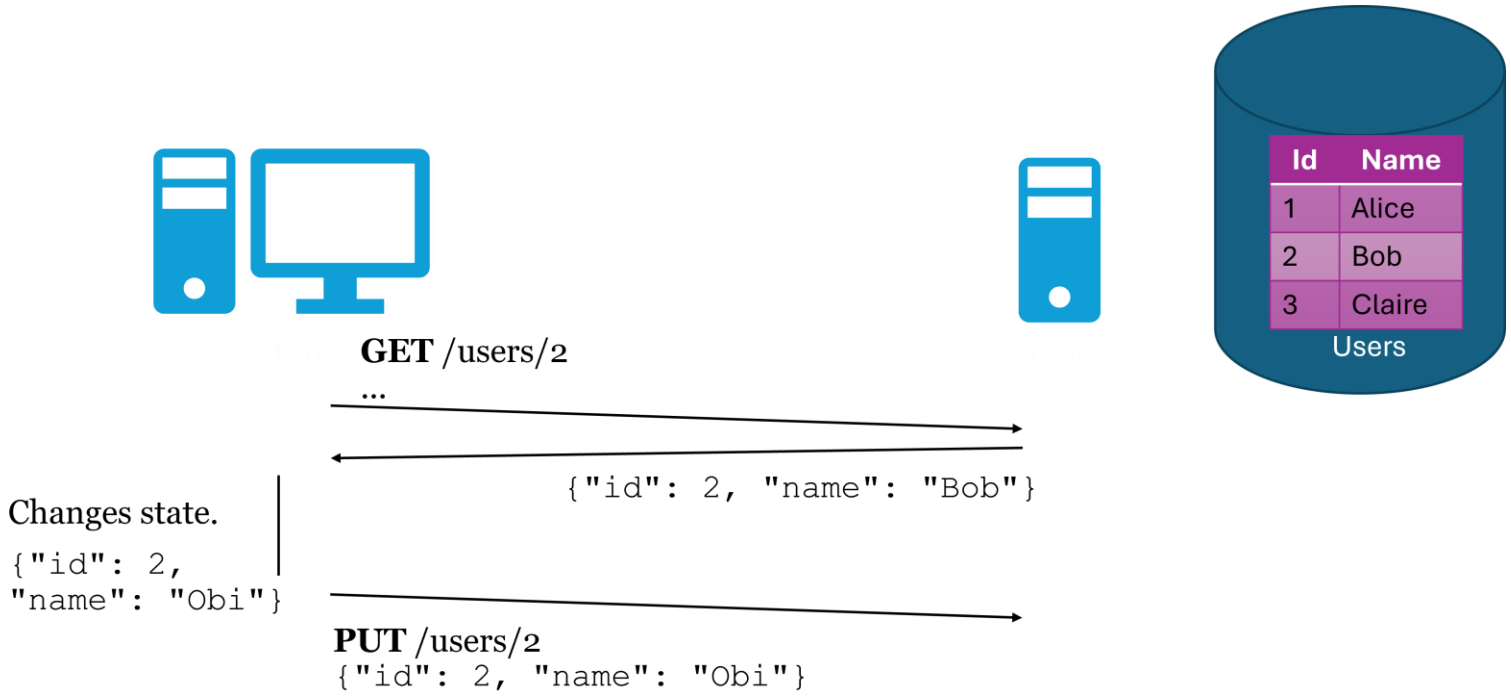
- **REST stands for Representational State Transfer.**
- **It is an architectural style for designing networked applications.**
- **RESTful systems communicate via HTTP requests.**
- **The key principles of REST include statelessness, resource-based operations, and a uniform interface.**



# RESTful HTTP Methods

- **GET:** Retrieve data from a server.
- **POST:** Submit data to be processed to a server.
- **PUT:** Update existing data on a server.
- **DELETE:** Remove data from a server.
- **PATCH:** Apply partial modifications to a resource.

# RESTful HTTP Methods - Example



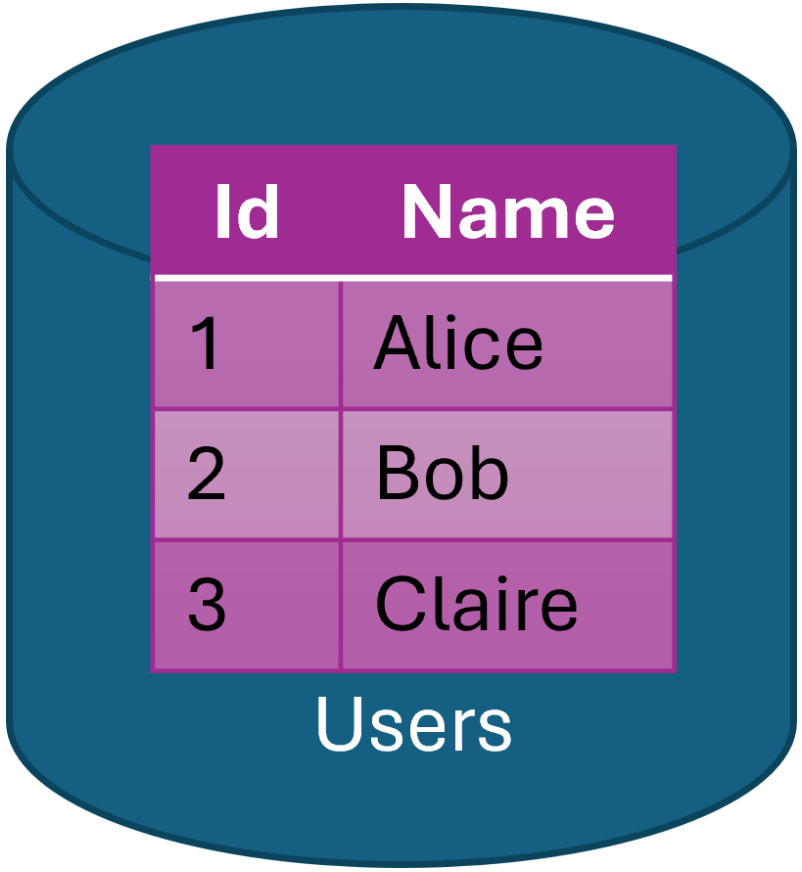
# RESTful URL Structure

- **Resources in REST are identified by URLs.**
- **Example URL:** https://api.example.com/users/123
- **In this example:**
  - https://api.example.com is the base URL.
  - /users refers to the resource (users).
  - /123 identifies a specific user (user with ID 123).

# REST example

A server with information about users.

- The GET method is used to retrieve resources.
  - `GET /users`
  - `GET /users/2`
  - `GET /users/pages/1`
  - `GET /users/gender/female`
  - `GET /users/age/18`
  - `GET /users/???`
  - `GET /users/2/name`
  - `GET /users/2/pets`



# REST example

A server with information about users.

- The GET method is used to retrieve resources.
  - Which data format? Specified by the `Accept` header!

```
GET /users HTTP/1.1
Host: the-website.com
Accept: application/json
```


```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 66

[
  {"id": 1, "name": "Alice"},
  {"id": 2, "name": "Bob"}
]
```

# REST example

A server with information about users.

- The POST method is used to create resources.
  - Which data format? Specified by the `Accept` and `Content-Type` header!

```
POST /users HTTP/1.1
Host: the-website.com
Accept: application/json
Content-Type: application/xml
Content-Length: 49

<user>
  <name>Claire</name>
</user>
```

```
HTTP/1.1 201 Created
Location: /users/3
Content-Type: application/json
Content-Length: 28

{"id": 3, "name": "Claire"}
```

# REST example

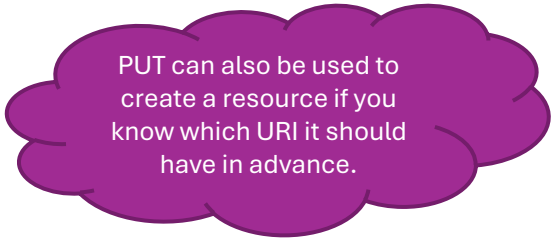
A server with information about users.

- The PUT method is used to update an entire resource.

```
PUT /users/3 HTTP/1.1
Host: the-website.com
Content-Type: application/xml
Content-Length: 52

<user>
  <id>3</id>
  <name>Cecilia</name>
</user>
```

```
HTTP/1.1 204 No Content
```



# REST example

A server with information about users.

- The DELETE method is used to delete a resource.

```
DELETE /users/2 HTTP/1.1
Host: the-website.com
```

```
HTTP/1.1 204 No Content
```

# REST example

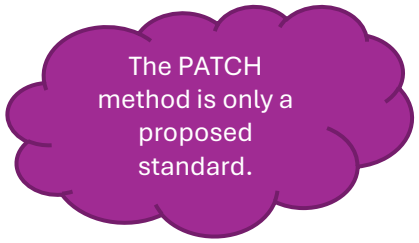
A server with information about users.

- The PATCH method is used to update parts of a resource.

```
PATCH /users/1 HTTP/1.1
Host: the-website.com
Content-Type: application/xml
Content-Length: 37

<user>
  <name>Amanda</human>
</user>
```

```
HTTP/1.1 204 No Content
```



# REST example

A server with information about users.

- What if something goes wrong?
  - Use the HTTP status codes to indicate success/failure.

```
GET /users/999 HTTP/1.1
Host: the-website.com
Accept: application/json
```

```
HTTP/1.1 404 Not Found
```

# REST API Response Codes

- **1xx:** Informational responses (e.g., 100 Continue).
- **2xx:** Success (e.g., 200 OK, 201 Created).
- **3xx:** Redirection (e.g., 301 Moved Permanently).
- **4xx:** Client errors (e.g., 404 Not Found, 400 Bad Request).
- **5xx:** Server errors (e.g., 500 Internal Server Error).

# Introduction to SOAP

- **SOAP stands for Simple Object Access Protocol.**
- **It is a protocol for exchanging structured information in web services.**
- **SOAP uses XML for message formatting and relies on other protocols like HTTP, SMTP for message negotiation and transmission.**

# SOAP Request Example

•SOAP requests are XML-based and typically include an envelope, header, and body.

- **Explanation:**
  - **Envelope:** Encapsulates the entire SOAP message.
  - **Header:** Optional section for metadata (e.g., authentication).
  - **Body:** Contains the actual request or response data.

```
POST /Service.asmx HTTP/1.1
Host: www.example.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.example.com/GetUserDetails"

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <!-- Optional: Headers can include authentication and other metadata -->
  </soap:Header>
  <soap:Body>
    <GetUserDetails xmlns="http://www.example.com/">
      <UserID>123</UserID>
    </GetUserDetails>
  </soap:Body>
</soap:Envelope>
```

# SOAP Response Example

- **Text:**
- **SOAP Response Structure:**
  - Like requests, SOAP responses are also XML-based, with a similar structure.
- **Explanation:**
  - **Body:** The response body contains the requested data, formatted as XML.
  - **GetUserDetailsResponse:** Contains the actual data retrieved from the service.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetUserDetailsResponse xmlns="http://www.example.com/">
      <GetUserDetailsResult>
        <User>
          <UserID>123</UserID>
          <Name>John Doe</Name>
          <Email>johndoe@example.com</Email>
        </User>
      </GetUserDetailsResult>
    </GetUserDetailsResponse>
  </soap:Body>
</soap:Envelope>
```

# Key Characteristics of SOAP

- **Protocol-based:** SOAP is a strict protocol with rules for encoding, binding, and processing messages.
- **XML-based:** All SOAP messages are written in XML, making it platform-independent.
- **WS-Security:** Provides built-in security, allowing encryption and authentication.
- **Stateful Operations:** SOAP can support both stateful and stateless operations, though it is commonly used for stateful ones.

# REST vs. SOAP - Key Differences

- **Architectural Style vs. Protocol:**
  - **REST:** Architectural style using standard HTTP.
  - **SOAP:** Protocol with specific standards and rules.
- **Data Format:**
  - **REST:** Supports multiple formats (JSON, XML, HTML).
  - **SOAP:** Strictly uses XML.
- **Complexity:**
  - **REST:** Simpler and easier to use.
  - **SOAP:** More complex due to its strict rules and XML-based messaging.

# REST vs. SOAP - Communication

- **REST:** Communication is stateless, meaning each request from a client contains all the information the server needs.
- **SOAP:** Can be stateless or stateful. State information can be maintained across multiple requests using WS-Security or other means.
- **REST:** Uses standard HTTP methods (GET, POST, PUT, DELETE).
- **SOAP:** Relies on a combination of WSDL (Web Services Description Language) for service description and SOAP protocol for message exchange.

# REST vs. SOAP - Performance

- **REST:** Generally faster due to its stateless nature and support for caching.
- **SOAP:** Slower, as it involves processing XML and often requires additional overhead in the form of WS-Security.
- **REST:** More suitable for high-performance, scalable web services.
- **SOAP:** Often used in enterprise environments where security and transaction compliance are critical.

# REST vs. SOAP - Use Cases

- **REST:**
  - Ideal for web and mobile applications that require fast, lightweight data exchange.
  - Commonly used in modern web services like social media, e-commerce, and cloud services.
- **SOAP:**
  - Preferred for enterprise-level applications where security, ACID compliance (Atomicity, Consistency, Isolation, Durability), and transaction management are crucial.
  - Used in banking, payment gateways, telecommunications.

# When to Use REST vs. SOAP

- **Choose REST if:**
  - You need a simple, scalable, and flexible service.
  - The application requires support for multiple formats (e.g., JSON, XML).
  - Speed and ease of development are priorities.
- **Choose SOAP if:**
  - The application requires advanced security features.
  - You need to maintain state across multiple requests.
  - The service must adhere to strict standards for transactions and messaging.

# Conclusion

- **Both REST and SOAP are powerful tools for building APIs, each with its own advantages and ideal scenarios.**
- **Understanding the differences helps in choosing the right approach for your application's needs.**