A computer server and information

Description automatically generated with medium confidence

A diagram of a server architecture

Description automatically generated

A diagram of a application

Description automatically generated

A screenshot of a computer program

Description automatically generated

SOAP (Simple Object Access Protocol), REST (Representational State Transfer), and Web Services are three different approaches for designing and implementing web-based communication systems. Here's a detailed comparison of these three approaches:

**SOAP (Simple Object Access Protocol):**

1. **Protocol**:
   * SOAP is a protocol, meaning it defines a set of rules and conventions for structuring messages and performing communication.
2. **Message Format**:
   * SOAP messages are typically XML-based. They have a well-defined structure with headers for metadata and a body for the actual data.
3. **Statefulness**:
   * SOAP can be designed to be stateful, meaning it can maintain session information between requests.
4. **Transport Protocol**:
   * SOAP can work over multiple transport protocols, including HTTP, SMTP, and more. It is not tied to a specific transport.
5. **Security**:
   * SOAP has built-in security features, such as WS-Security, for handling authentication, encryption, and digital signatures.
6. **Complex Operations**:
   * SOAP is well-suited for complex operations and transactions. It supports features like ACID transactions.
7. **Tool Support**:
   * There are many tools and libraries available for working with SOAP-based web services in various programming languages.

**REST (Representational State Transfer):**

1. **Architectural Style**:
   * REST is an architectural style or design pattern, not a protocol. It relies on a set of principles for creating web services.
2. **Message Format**:
   * REST messages are not restricted to XML; they can use various formats, including JSON, XML, HTML, and others.
3. **Statelessness**:
   * REST is inherently stateless. Each request from a client to a server must contain all the information needed to understand and process the request.
4. **Transport Protocol**:
   * REST primarily uses HTTP as its transport protocol. It leverages HTTP methods (GET, POST, PUT, DELETE, etc.) for performing operations on resources.
5. **Security**:
   * REST relies on underlying security mechanisms provided by HTTP, such as HTTPS for encryption and HTTP headers for authentication.
6. **Simplicity and Scalability**:
   * REST is designed to be simple and easy to understand. It is well-suited for scalable and lightweight applications.
7. **State Management**:
   * Since REST is stateless, session management is typically handled on the client side, often using tokens or cookies.

**Web Services:**

1. **General Term**:
   * Web services is a broad term that encompasses various technologies and protocols for enabling communication between different software applications over the web.
2. **Includes Both SOAP and REST**:
   * Web services can include both SOAP-based and RESTful services, as well as other technologies like XML-RPC and JSON-RPC.
3. **Technology Agnostic**:
   * Web services are not tied to a specific technology or protocol; they represent a concept of enabling interoperability between distributed systems.
4. **Standards and Specifications**:
   * Web services often adhere to industry standards and specifications, such as WSDL (Web Services Description Language) for SOAP-based services.

In summary, SOAP is a protocol with a rigid structure, REST is an architectural style with flexibility, and web services encompass a broad range of technologies for enabling communication between software applications. The choice between SOAP and REST often depends on the specific requirements and constraints of a given project, with REST being favored for simplicity and scalability in many cases.

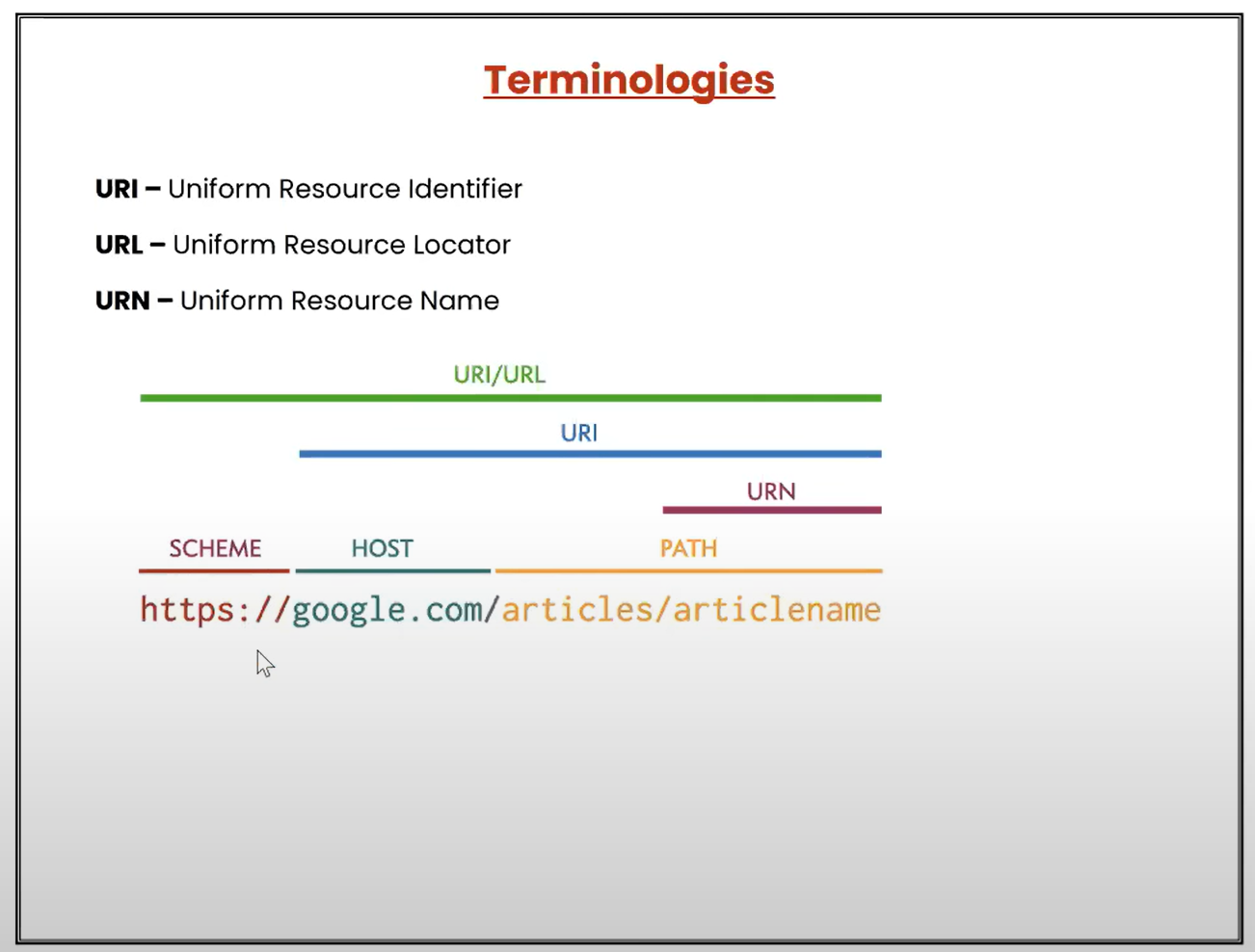
Top of Form

A diagram of a process flow

Description automatically generated

A screenshot of a computer

Description automatically generated



URI (Uniform Resource Identifier), URL (Uniform Resource Locator), and URN (Uniform Resource Name), and provide examples for each.

**1. URI (Uniform Resource Identifier):**

A Uniform Resource Identifier (URI) is a generic term used to identify a resource on the internet. It can be further classified into two specific types: URLs and URNs.

**2. URL (Uniform Resource Locator):**

A Uniform Resource Locator (URL) is a specific type of URI that not only identifies a resource but also provides the means to locate it using its address. A URL typically consists of the following components:

* **Scheme**: Specifies the protocol or method used to access the resource (e.g., "http," "https," "ftp").
* **Host**: Indicates the domain name or IP address of the server where the resource is hosted.
* **Port (Optional)**: Specifies the port number on the server for the resource, if different from the default for the scheme.
* **Path**: Represents the specific location or path to the resource on the server.
* **Query (Optional)**: Contains parameters or data passed to the resource, often used in web searches.
* **Fragment (Optional)**: Points to a specific section or anchor within the resource.

**Example of a URL**:

bashCopy code

https://www.example.com:8080/products/index.html?category=electronics#section2

In this URL:

* Scheme: **https**
* Host: **www.example.com**
* Port: **8080** (optional)
* Path: **/products/index.html**
* Query: **category=electronics**
* Fragment: **section2** (optional)

A URL provides both the identity and the location of a resource.

**3. URN (Uniform Resource Name):**

A Uniform Resource Name (URN) is another type of URI that is used to provide a unique and persistent name for a resource, without necessarily specifying how to access it. URNs are often used for resources that need long-term identification but may not have a specific location or address. A URN typically follows a specific namespace scheme to ensure uniqueness.

**Example of a URN**:

cssCopy code

urn:isbn:0451450523

In this URN:

* **urn** indicates that it's a URN.
* **isbn** is a namespace that specifies a unique identification scheme for International Standard Book Numbers.
* **0451450523** is the specific identifier for a book.

In this case, the URN provides a unique name for a book but doesn't specify where or how to access it.

**Summary:**

* URI is a general term for identifying resources on the internet.
* URL is a specific type of URI that provides both identification and location information for a resource.
* URN is another type of URI that provides a unique and persistent name for a resource without specifying its location.

In summary, while both URLs and URNs are types of URIs, the key difference lies in whether they provide both a resource's identity and its location (URL) or just its identity (URN).

Via browser only Get request is possible. For others we have to use Postman.

Payload is the data which we send in the request.

End point: Address where API is hosted on the Server.

HTTP methods which are commonly used to communicate with Rest API’s are

**GET, POST, PUT, and DELETE**

GET- The GET method is used to extract information from the given server using a given URI. While using GET request, it should only extract data and should have no other effect on the data. No Payload/Body required

**How to send input data in GET?**  
Ans: Using Query Parameters

POST- A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.

**How to send input data in POST?**  
Ans: Using Form Parameters /Body Payload

PUT- Replaces all current representations of the target resource with the uploaded content.

DELETE- Removes all current representations of the target resource given by a URI.

**Resources:  
Resources represent API/Collection which can be accessed from the Server**

Google.com/maps  
google.com/search  
google.com/images

**Path Parameters:**  
**Path parameters** are variable parts of a URL path. They are typically used to point to a specific resource within a collection, such as a user identified by ID

<https://www.google.com/Images/1123343>  
<https://www.google.com/docs/1123343>  
<https://amazon.com/orders/112>

<https://www.google.com/search?q=newyork&oq=newyork&aqs=chrome..69i57j0l7.2501j0j7&sourceid=chrome&ie=UTF-8>

**Query Parameters:**  
Query Parameter is used to sort/filter the resources.

Query Parameters are identified with?””

https://amazon.com/orders?sort\_by=2/20/2020

**Headers/Cookies**:

Headers represent the meta-data associated with the API request and response. In layman terms, we were sending Additional details to API to process our request.  
Example : Authorization details

**End Point Request URL can be constructed as below**  
Base URL/resource/(Query/Path)Parameters

**1. Resources:**

In the context of web services and APIs, a resource represents an object or piece of data that can be accessed or manipulated. Resources are typically identified by a unique URI (Uniform Resource Identifier). Resources can be anything from a web page, image, document, or any other data entity that the API exposes.

**Example**: In a RESTful API for an e-commerce website, resources could include:

* **/products**: Represents a collection of all products.
* **/products/123**: Represents a specific product with an ID of 123.

**2. Path Parameters:**

Path parameters are placeholders within the URL path that allow you to specify variable values when making a request. They are used to identify or filter resources based on specific criteria. Path parameters are typically denoted by curly braces **{}** in the URL path.

**Example**: In a URL **/users/{userId}/posts**, **{userId}** is a path parameter that can be replaced with an actual user ID when making a request. For example, **/users/42/posts** would fetch posts for the user with an ID of 42.

**3. Query Parameters:**

Query parameters are key-value pairs appended to the end of a URL and are separated from the base URL by a question mark **?**. They are used to provide additional information or filter data when making a request. Multiple query parameters can be included by separating them with ampersands **&**.

**Example**: In a URL **/search?query=api&limit=10**, **query** and **limit** are query parameters. They allow you to search for resources containing "api" and limit the results to 10 items.

**4. Headers/Cookies:**

Headers and cookies are used to send additional information along with an HTTP request. They provide metadata about the request or additional data needed for authentication, caching, or other purposes.

* **Headers**: HTTP headers are key-value pairs included in the request or response to provide information such as content type, encoding, authorization, and more. Common headers include **Content-Type**, **Authorization**, **Accept**, and **User-Agent**.

**Example**: In an HTTP request, you might include an **Authorization** header with a token for authentication:

makefileCopy code

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

* **Cookies**: Cookies are small pieces of data stored by a web server in a user's browser. They can be sent with each HTTP request to provide session information or user-specific data.

**Example**: When a user logs in to a website, a session cookie is set. On subsequent requests, the cookie is sent automatically to identify the user's session.

In summary, resources represent objects or data accessible via APIs, path parameters allow dynamic parts in URLs, query parameters provide filtering or additional data, and headers/cookies convey metadata and user-specific information in HTTP requests and responses. These concepts are fundamental for working with APIs and web services.

POSTMAN:-

<https://www.postman.com/downloads/>

You cannot do API Testing directly.

You need API contract(such as Base URL, Resource, Query Parameters, HTTP Method, Sample Body, Response)

So in Postman first to open a new Project, we call it as a Collection

Add Request and name the request as AddPlace

First we will do the POST call and pass the Enpoint url and then we need to copy the Sample Body from the contract and paste it in the body > raw > JSON

And click on Send. Check for the response code and match