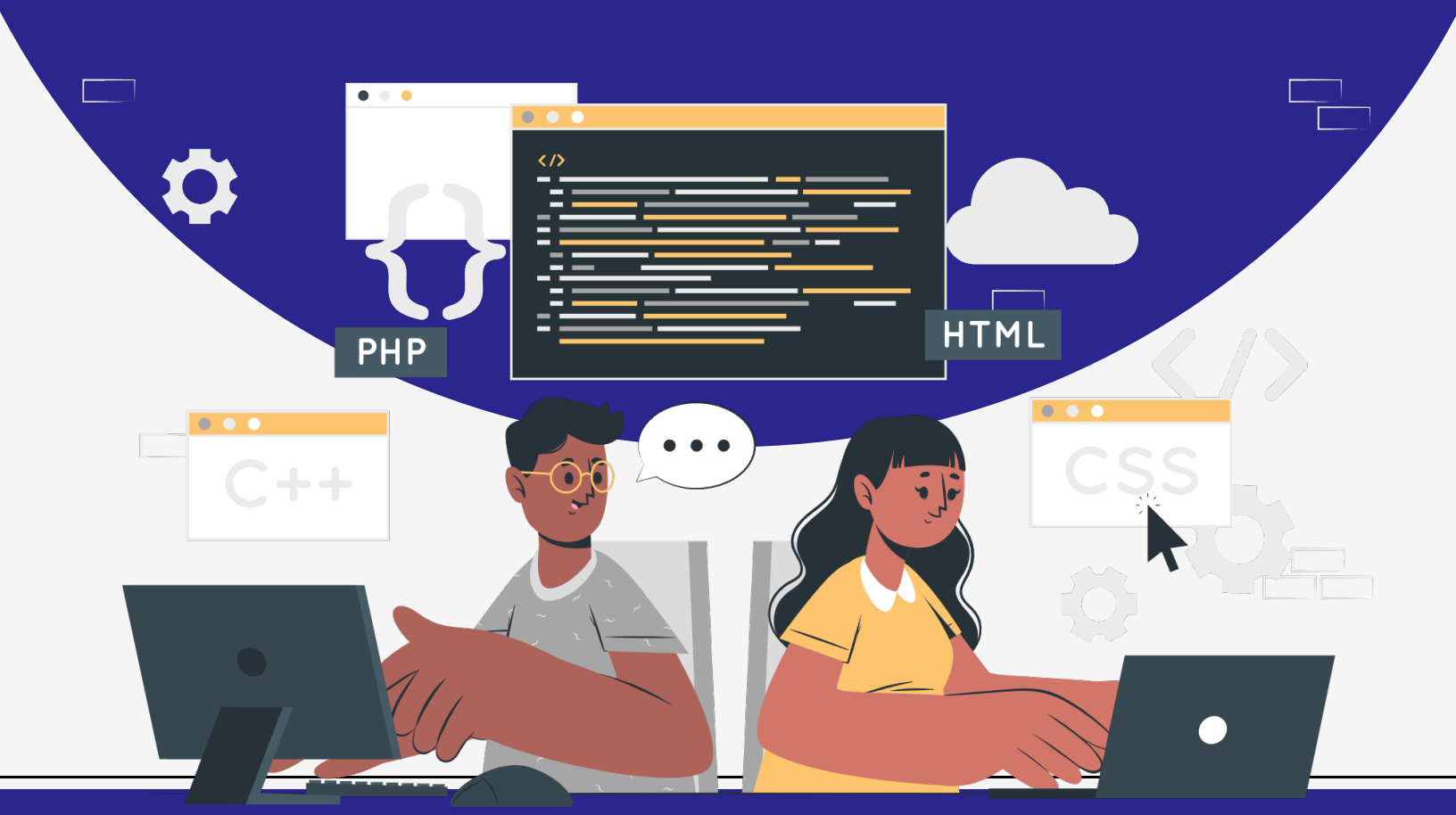
Lesson:

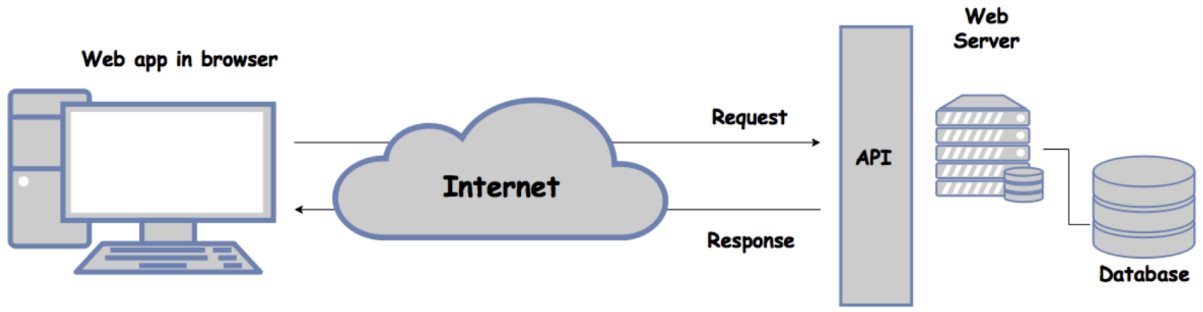
Rest API



PW SKILLS

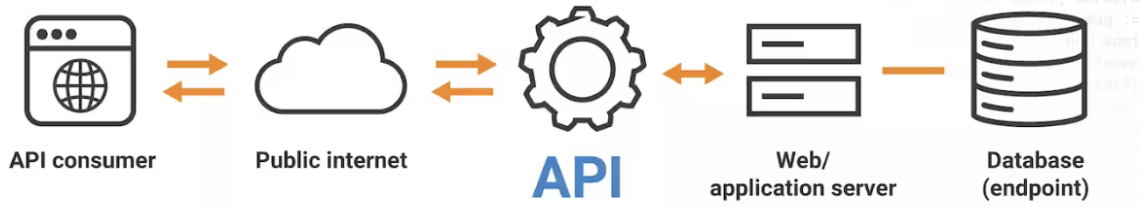
Topics Covered:

1. What is API
2. What is SOAP?
3. Why REST?
4. What is URI?
5. HTTP Methods
6. RESTFUL WEB SERVICES



What is API?

An API, which stands for Application Programming Interface, is a software intermediary that allows two applications to communicate with each other using a set of definitions and protocols APIs define how different software components interact by specifying the requests and responses between them. They play a crucial role in enabling seamless communication between various applications, facilitating data sharing and functionality across different systems. APIs are essential in modern technology, powering interactions like those seen in rideshare apps, mobile payments, and smart home devices etc.



What is SOAP?

- The **Simple Object Access Protocol** (SOAP) is a lightweight protocol for exchanging information in a decentralized, distributed environment. It allows applications to communicate with each other over the internet, even if they are written in different programming languages. SOAP messages are XML-based and can be carried over various standard protocols like HTTP, enabling the exchange of structured data between nodes.
- The key components of a SOAP message include the Envelope (identifying the message as a SOAP message), Header (containing additional information like authentication credentials), Body (details of the actual message), and optional Fault (for error handling).
- SOAP provides a standard way for processes to communicate, making it easier for developers to create web APIs and integrate services across different platforms and languages

Why REST?

SOAP was one of the earliest protocols used for implementing web services and is known for its robustness and reliability. However, it can be relatively verbose due to its XML-based nature, and its complexity may not be necessary for all use cases. As a result, alternative approaches such as Representational State Transfer **(REST)** have gained popularity for building web services, especially for simpler, resource-oriented interactions. Nonetheless, SOAP remains widely used in enterprise environments where compatibility with existing systems and support for advanced features like transactions and security are critical.

One of the key distinctions between REST and SOAP lies in the data format they use for message exchange.

1. **REST (Representational State Transfer):** RESTful services commonly use JSON (JavaScript Object Notation) as the data format for message payloads. JSON is a lightweight, human-readable data interchange format that is easy to parse and generate. It is based on key-value pairs and supports arrays and nested structures. JSON is widely supported by programming languages and web frameworks, making it a popular choice for RESTful APIs.

2. **SOAP (Simple Object Access Protocol):** SOAP services, on the other hand, typically use XML (eXtensible Markup Language) for message exchange. XML is a markup language similar to HTML, but it is more flexible and extensible. SOAP messages are structured using XML elements, attributes, and namespaces. While XML is verbose compared to JSON, SOAP offers features such as formal message exchange patterns, security, and transaction support.

The choice between JSON and XML often depends on factors such as simplicity, readability, interoperability, and the requirements of the application or system being developed. JSON is favored for its lightweight and easy-to-read syntax, making it ideal for web APIs and data exchange between client-side and server-side components in modern web applications. Meanwhile, XML and SOAP are commonly used in enterprise environments, especially when compatibility with existing systems, standardized messaging formats, and advanced features such as transaction management and security are required.

XML Example:

```
<book>
  <title>Harry Potter and the Philosopher's Stone</title>
  <author>J.K. Rowling</author>
  <genre>Fantasy</genre>
  <year>1997</year>
</book>
```

JSON Example:

```
{
  "title": "Harry Potter and the Philosopher's Stone",
  "author": "J.K. Rowling",
  "genre": "Fantasy",
  "year": 1997
}
```

In both examples, the data represents details about a book, including its title, author, genre, and publication year.

1. **In the XML example,** the data is structured using tags enclosed in **angle brackets < >**, with each tag representing a different attribute of the book. XML allows for hierarchical organization of data and the use of attributes and namespaces.

2. **In the JSON example,** the data is represented as a collection of key-value pairs enclosed in **curly braces { }**. Each key represents an attribute of the book (e.g., "title", "author"), and its corresponding value provides the actual information (e.g., "Harry Potter and the Philosopher's Stone", "J.K. Rowling"). JSON is lightweight and easy to read, making it a popular choice for data interchange in web APIs and modern applications.

N:B - CRUD operations (Create, Read, Update, Delete) are fundamental actions performed on data in a system. RESTful APIs (Representational State Transfer) provide a standardized approach for implementing these operations over HTTP.

What is URI?

URI stands for Uniform Resource Identifier. It is a string of characters that uniquely identifies a particular resource, typically on the internet. URIs serve as addresses or names for resources and are used to locate, identify, and interact with them.

A URI can take various forms, including:

1. **URL (Uniform Resource Locator):** A subset of URIs that specifies the location of a resource on the internet. URLs typically consist of a protocol (e.g., HTTP, HTTPS), domain name, path, and optional query parameters.

   Example: https://www.example.com/products?id=123

2. **URN (Uniform Resource Name):** Another subset of URIs that serves as a persistent identifier for a resource, regardless of its location. URNs are often used to identify abstract or conceptual resources.

   Example: urn:isbn:0451450523

In RESTful APIs, URI (Uniform Resource Identifier) is used to identify resources and specify the location of those resources on the server. Two common components of URIs that are frequently used in RESTful APIs are path parameters and query parameters.

1. Path Parameters:
Path parameters are variables that are part of the URL path itself. They are used to identify a specific resource or specify a subset of resources within a collection. Path parameters are typically denoted by placeholders in the URI path.

Example:
Suppose we have a RESTful API for managing users, and we want to retrieve information about a specific user. We can use a path parameter to specify the user's unique identifier in the URI path:

```
GET /users/{userId}
```

In this example, {userId} is a path parameter representing the unique identifier of the user. For instance, to retrieve information about user with ID 123, the URI would be:

```
GET /users/123
```

2. Query Parameters:
Query parameters are additional parameters appended to the end of the URI to provide additional information to the server. They are separated from the base URI by a question mark (?) and are specified as key-value pairs.

Example:
Continuing with our user management API example, let's say we want to filter users based on certain criteria, such as their role or status. We can use query parameters to specify these filters:

```
GET /users?role=admin&status=active
```

In this URI, role and status are query parameters, and admin and active are their respective values. This request retrieves a list of users who have the role "admin" and are in the "active" status.

HTTP Methods

HTTP (Hypertext Transfer Protocol) is the foundation of data communication on the web. At its core, HTTP defines a set of methods indicating the desired action for a given resource. This comprehensive guide will unravel the intricacies of popular HTTP methods.

HTTP methods, also known as verbs or operations, represent the actions a client wants to take on a resource. Here are the primary HTTP methods:

- GET Method
- POST Method
- PUT Method
- DELETE Method

GET Method:
The GET method requests data from a specified resource. It should only retrieve data and have no other effect on the server.

The use cases include
- Retrieving information from a server
- Safe and idempotent operation

POST Method:
The POST method submits data to be processed to a specified resource. It results in the creation of a new resource. The use cases include

- Creating a new resource on the server
- Submitting form data

PUT Method:
The PUT method updates or creates a resource if it doesn't exist. It is idempotent, meaning multiple identical requests have the same effect as a single request. The use cases include

- Modifying existing resources
- Creating resources if it doesn't exist

DELETE Method:
The DELETE method requests the removal of a resource. It is idempotent, meaning multiple identical requests have the same effect as a single request. The use cases include

- Deleting a specific resource
- Cascading deletes in certain scenarios

RESTFUL WEB SERVICES

RESTful web services adhere to the principles of REST (Representational State Transfer), which defines a set of architectural constraints for designing networked applications. RESTful web services use standard HTTP methods (GET, POST, PUT, DELETE) to perform CRUD operations (Create, Read, Update, Delete) on resources identified by URIs (Uniform Resource Identifiers). The message components of RESTful web services include:

1. **HTTP methods:**

RESTful web services use standard HTTP methods to perform operations on resources:

- GET: Retrieves data from the server.
- POST: Creates a new resource on the server.
- PUT: Updates an existing resource on the server.
- DELETE: Deletes a resource from the server.
- PATCH: Partially updates an existing resource.
- HEAD: Retrieves the headers of a resource without the body.
- OPTIONS: Retrieves the supported HTTP methods for a resource.

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

- URIs uniquely identify resources on the server.
- They consist of a base URL and a path that specifies the location of the resource.
- Example: https://api.example.com/users/123

3. Headers:

- HTTP headers provide additional information about the request or response.
- Common headers include Content-Type, Accept, Authorization, etc.
- Example:

```
Content-Type: application/json
Authorization: Bearer <token>
```

4. Request Body:

- The request body contains data sent from the client to the server, typically in JSON or XML format.
- It is used for creating or updating resources.
- Example (JSON):

```
{
  "username": "john_doe",
  "email": "john@example.com"
}
```

5. Response Body:

- The response body contains data sent from the server to the client, typically in JSON or XML format.
- It may include the requested resource or an error message.
- Example (JSON):

```
{
  "id": 123,
  "username": "john_doe",
  "email": "john@example.com"
}
```

6. Status Codes:

- HTTP status codes indicate the result of the request.
- Common status codes include 200 (OK), 201 (Created), 400 (Bad Request), 404 (Not Found), 500 (Internal Server Error), etc.
- Example:

```
HTTP/1.1 200 OK
```

PW SKILLS

THANK YOU !