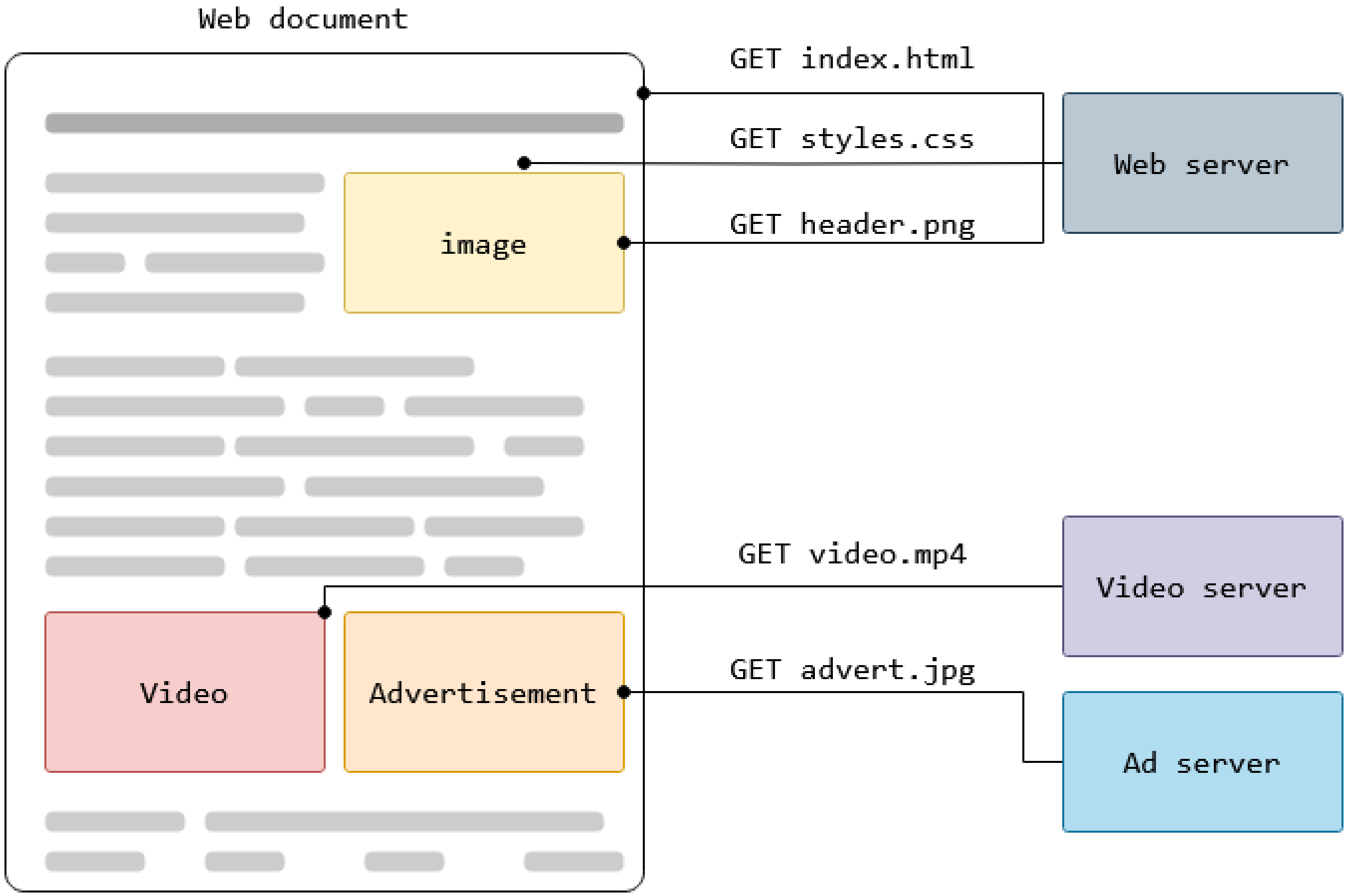
# Web Programming II (Advanced Web Development)

By Djakba Faustin

# Advanced Web Architecture and Protocols

HTTP Review: Methods, Status Codes, Requests, and Responses.

**HTTP** is a protocol for fetching resources such as HTML documents. It is the foundation of any data exchange on the Web and it is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser. A complete document is typically constructed from resources such as text content, layout instructions, images, videos, scripts, and more.


Web document
GET index.html
GET styles.css
GET header.png
image
Web server
GET video.mp4
Video server
Video
Advertisement
GET advert.jpg
Ad server

Clients and servers communicate by exchanging individual messages (as opposed to a stream of data). The messages sent by the client are called *requests* and the messages sent by the server as an answer are called *responses*

## Components of HTTP-based systems

**1-Client: the user-agent**

The *user-agent* is any tool that acts on behalf of the user. This role is primarily performed by the Web browser, but it may also be performed by programs used by engineers and Web developers to debug their applications

## Components of HTTP-based systems

**2-The Web server**

On the opposite side of the communication channel is the server, which *serves* the document as requested by the client.

## Components of HTTP-based systems

**3-proxies**

Between the Web browser and the server, numerous computers and machines relay the HTTP messages. Due to the layered structure of the Web stack, most of these operate at the transport, network or physical levels, becoming transparent at the HTTP layer and potentially having a significant impact on performance. Those operating at the application layers are generally called **proxies**.

## HTTP request methods

HTTP defines a set of **request methods** to indicate the purpose of the request and what is expected if the request is successful. Although they can also be nouns, these request methods are sometimes referred to as *HTTP verbs*. Each request method has its own semantics, but some characteristics are shared across multiple methods, specifically request methods can be safe, idempotent, or cacheable.

## State Management (Sessions and Cookies) and Security Considerations.

State management in web applications, particularly through sessions and cookies, involves maintaining user-specific data across multiple requests.

**Sessions**

Sessions store user data on the server-side, identified by a unique session ID, typically stored in a cookie on the client. This approach centralizes data management and offers enhanced security for sensitive information, as the data itself is not exposed to the client.

## State Management (Sessions and Cookies) and Security Considerations.

**Cookies**

Cookies are small pieces of data stored directly in the user's browser. They are commonly used for:

•**Session Management:** Storing session IDs or tokens to link client requests to server-side session data.

•**Personalization:** Remembering user preferences like language or theme.

•**Tracking:** Analyzing user behavior for analytics or targeted advertising.

## Security Considerations

**Use Secure and HttpOnly Flags**: Set the Secure flag to ensure cookies are only sent over HTTPS connections, and use HttpOnly for cookies that don’t need to be accessed by JavaScript.

**Implement SameSite Restrictions**: Set appropriate SameSite attributes (Strict, Lax, or None) based on your cross-origin requirements.

**Set Proper Expiration Times**: Balance security and user convenience when setting cookie and session expiration times.

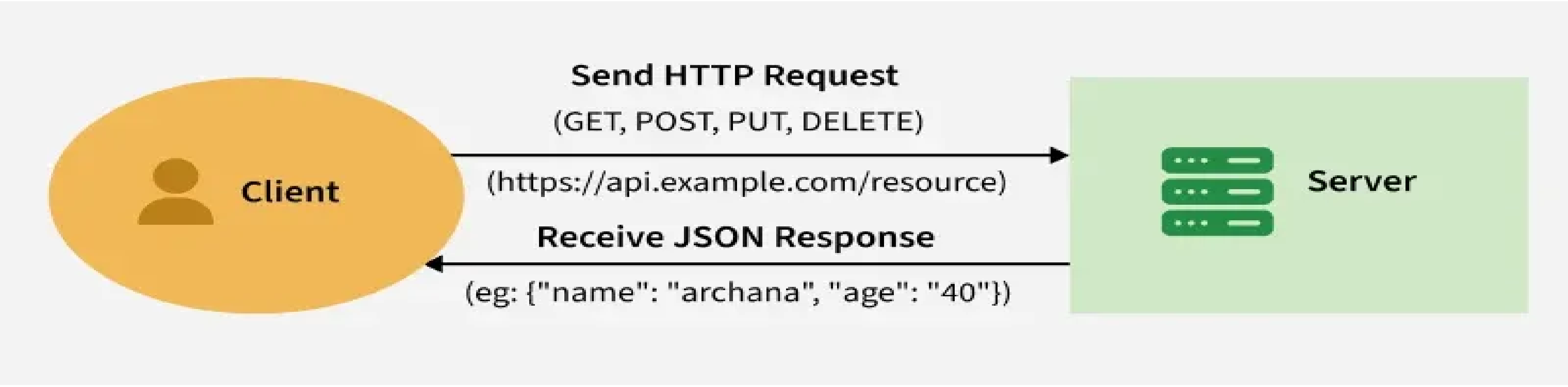
## Security Considerations

**Protect Against CSRF**: Implement anti-CSRF tokens for important actions, especially when SameSite restrictions aren’t sufficient.

**Session Fixation Protection**: Generate new session IDs when authentication levels change, particularly after login.

## Introduction to Web APIs and RESTful Services.

REST API stands for Representational State Transfer API. It is a type of API (Application Programming Interface) that allows communication between different systems over the internet. REST APIs work by sending requests and receiving responses, typically in JSON format, between the client and server.



•A request is sent from the client to the server via a web URL, using one of the HTTP methods.

•The server then responds with the requested resource, which could be HTML, XML, Image, or JSON, with JSON being the most commonly used format for modern web services.

•These methods map to CRUD operations (Create, Read, Update, Delete) for managing resources on the web.

## Key Characteristics of RESTful Services:

**HTTP Methods:** They utilize standard HTTP methods like GET (retrieve data), POST (create data), PUT (update data), and DELETE (remove data) to perform operations on resources.

**Resource-Oriented:** They focus on resources (e.g., users, products) and their interactions, rather than actions or procedures.

**Stateless:** Each request is independent, simplifying server design and improving scalability.

**Lightweight:** They typically use lightweight data formats like JSON (JavaScript Object Notation) or XML (Extensible Markup Language) for data exchange.

## RESTful Services

REST (Representational State Transfer) is an architectural style for designing networked applications. RESTful services, also known as RESTful APIs, are Web APIs that adhere to the principles of REST. These principles include:

- **Client-Server Architecture:** The client (e.g., a web browser or mobile app) and the server (where the resources reside) are independent, allowing for separate development and evolution.
- **Statelessness:** Each request from a client to the server must contain all the information necessary to understand the request. The server does not store any client context between requests.

**.Cacheability:** Responses from the server can be explicitly or implicitly marked as cacheable, allowing clients to reuse stored responses for subsequent requests.

**.Uniform Interface:** A uniform interface simplifies the overall system architecture and improves visibility.

**.Layered System:**

A client cannot ordinarily tell whether it is connected directly to the end server or to an intermediary.

**.Code on Demand (*Optional*)**

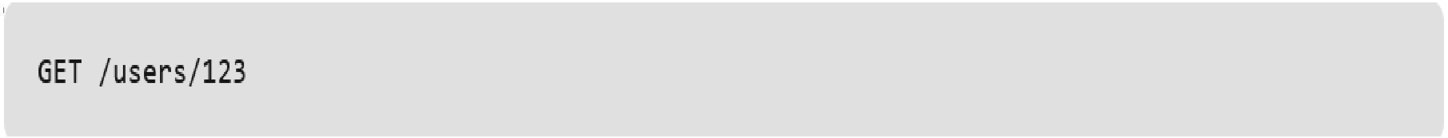
REST also allows client functionality to be extended by downloading and executing code in the form of applets or scripts.

## Common HTTP Methods Used in REST API

In HTTP, there are five methods that are commonly used in a REST-based Architecture, i.e., POST, GET, PUT, PATCH, and DELETE.

## 1. GET Method

The HTTP GET method is used to **read** (or retrieve) a representation of a resource. In the safe path, GET returns a representation in XML or JSON and an HTTP response code of 200 (OK). In an error case, it most often returns a 404 (NOT FOUND) or 400 (BAD REQUEST).



```
GET /users/123
```

## 2. POST Method

The POST method is commonly used to create new resources. It is often used to create subordinate resources related to a parent resource. Upon successful creation, the server returns HTTP status 201 (Created) along with a Location header pointing to the newly created resource

```
POST /users
{
  "name": "Anjali",
  "email": "gfg@example.com"
}
```

## 3. PUT Method

PUT is an HTTP method used to update or create a resource on the server. When using PUT, the entire resource is sent in the request body, and it replaces the current resource at the specified URL. If the resource doesn’t exist, it can create a new one.

```
PUT /users/123
{
  "name": "Anjali",
  "email": "gfg@example.com"
}
```

## 4. PATCH Method

PATCH is an HTTP method used to partially update a resource on the server. Unlike PUT, PATCH only requires the fields that need to be updated to be sent in the request body. It modifies specific parts of the resource rather than replacing the entire resource.

```
PATCH /users/123
{
  "email": "new.email@example.com"
}
```

## 5. DELETE Method

It is used to **delete** a resource identified by a URI. On successful deletion, return HTTP status 200 (OK) along with a response body.

```
DELETE /users/123
```

## Real-World Implementation Examples

### Django (Python)

```python
# In settings.py
SESSION_COOKIE_SECURE = True
SESSION_COOKIE_HTTPONLY = True
SESSION_COOKIE_AGE = 86400  # 24 hours in seconds

# In a view
def set_session_data(request):
    request.session['username'] = 'john_doe'
    return HttpResponse('Session data set')
def get_session_data(request):
    username = request.session.get('username', 'Guest')
    return HttpResponse(f'Hello, {username}!')
```

## Real-World Implementation Examples

### Laravel (PHP)

```php
// Setting session data
Route::get('/set-data', function (Request $request) {
    $request->session()->put('username', 'john_doe');
    return 'Session data set';
});

// Getting session data
Route::get('/get-data', function (Request $request) {
    $username = $request->session()->get('username', 'Guest');
    return "Hello, {$username}!";
});
```