CHAPTER 5
USING WEB PROTOCOLS

DEBABRATA ROY
Asst. Prof. (Dept. of CA), ITER, SOAU
Ph.D. from **NIT Raipur**
MCA from **NIT Agartala** (*Gold Medalist*)
B.Sc(CS) from **NBU** (*Gold Medalist*)
Phone (WP): 8918671070

HYPERTEXT TRANSFER PROTOCOL

- Basically, web protocols define the way clients and servers communicate with each other.
- The protocol used between the browser (or client) and the server is HTTP(Hypertext Transfer Protocol).
- It is an application-layer protocol used primarily with the World Wide Web (WWW) in the client-server model, where a web browser is a client communicating with the web server, which is hosting the website.
- HTTP is a standard and stateless protocol that is used for different purposes, as well as using extensions for request methods, error codes, and headers.
- HTTP is a communication protocol that is employed for delivering data (usually HTML files, multimedia files, etc.) on the World Wide Web through its default TCP port 80.

CONTINUE...

A necessary HTTP request has the following steps:

- Initially, a link to the HTTP server gets opened.
- Then a request is sent.
- It does some processing on the server.
- Once the request processing is done, the response is sent back from the server.
- Finally, the connection is closed.

ARCHITECTURE OF HTTP

HTTP is meant for request/response depending on a client-server architecture where the user requests information through a web browser to the web server, which then responds to the requested data.

- **Web Client:** The client of this client-server architecture makes a request to a specific server through the HTTP (TCP/IP connection) as a request method in the form of a URL. It also contains a MIME-like message that contains a request modifier and client information.
- **Web Server:** This accepts the request and process with a response by a status line, together with the version of the message's protocol as well as the success or error code, followed by a MIME-like message having server information, some metadata, and possible the entity-body content holding the requested information.

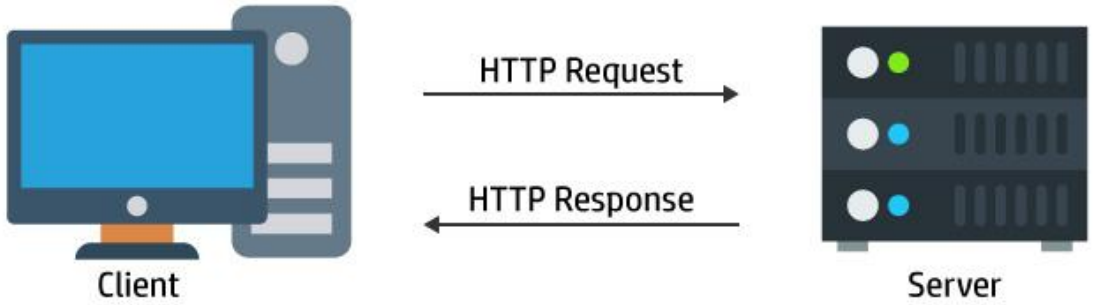


Fig: HTTP Protocol

REQUESTS-RESPONSES

Communication between clients and servers is done by **requests** and **responses:**

- A client (a browser, also called an agent or user agent) sends an **HTTP request** to the web (Server)
- The server receives the requests and runs an application to process the request i.e. often, responding to a request requires the server to read files from a file system, access a database, or perform other server-side logic or operations
- Once the server-side operations are complete, the server creates an HTTP response(Output) and sends it back to the client
- The client (the browser) receives the response

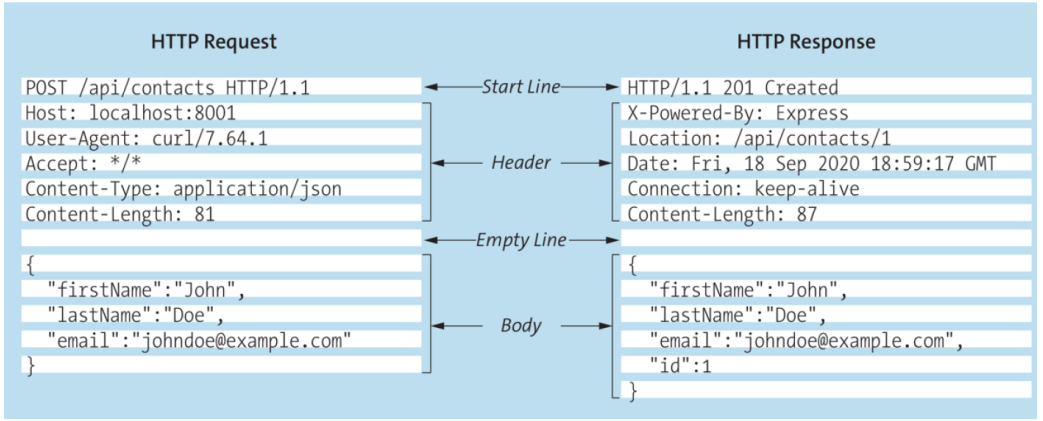


Figure 5.4 Structure of HTTP Request and Response

CONTINUE...

Requests and responses consist of the following sections:

- The start line (also status line or initial line or, depending on the type, request line or response line) describes exactly the request or the response.
- The lines that follow then optionally contain headers (also HTTP headers or, depending on the type, called request headers or response headers), which describe the request or response in more detail.
- Together, the start line and the header are referred to as meta information. To separate this metadata from the rest of the message, they are followed by an empty line.
- This empty line is in turn followed by the body (also payload or, depending on the type, called request body or response body), that is, the message body. In the case of a request, for example, this body can be form data; in the case of a response, for example, this body can be an HTML document, a stylesheet, or JavaScript code.

HTTP REQUEST STRUCTURE

HTTP Request begins with a request line, which has **3 parts**:

- **HTTP Method:** Specifies the action to perform (e.g., GET, POST, PUT, DELETE, CONNECT, TRACE, HEAD, OPTIONS).
  - GET requests retrieve data from the server (like images or CSS).
  - POST requests send data to the server (such as form submissions).
- **Request Target (URL):** The absolute or relative URL that the client wants to access.
- **HTTP Version:** Indicates which HTTP version (such as HTTP/1.1) is being used.

After the request line

Headers: Headers are key–value pairs providing additional information about the request. They fall into 3 categories:

- **General Headers** – apply to the entire message
- **Request Headers** – give extra details about the request
- **Entity Headers** – describe the message body (its type, size, etc.)

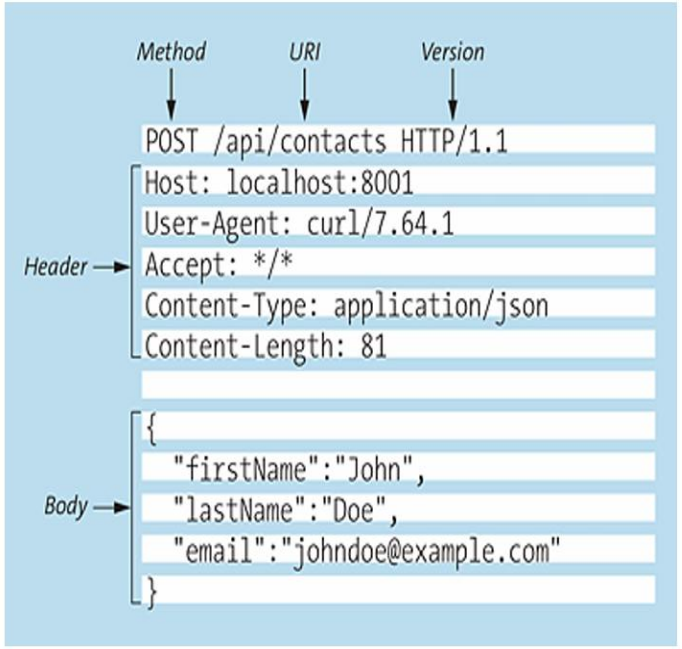


Figure 5.5 Structure of an HTTP Request

After the headers, there is an empty line, followed by: Request ***Body*** Contains data sent to the server (if any):

- POST requests usually have a body (form data, JSON, etc.).
- GET requests usually do not have a body because they only retrieve data.

HTTP RESPONSE STRUCTURE

HTTP response is built similarly to a request but has a different starting structure. It begins with a **response line,** which includes:

- **HTTP Version:** Indicates the protocol version used in the response (e.g., HTTP/1.1, HTTP/2).
- **Status Code:** A three-digit number describing whether the request was successful. *Example:* 404 means the requested resource was not found.
- **Status Text:** A short message explaining the status code. Example: 404 Not Found

Headers Following the response line, various headers provide extra information:

- **General Headers** – apply to the entire message.
- **Response Headers** – provide metadata about the response.
- **Entity Headers** – describe the response body (e.g., content type, length).

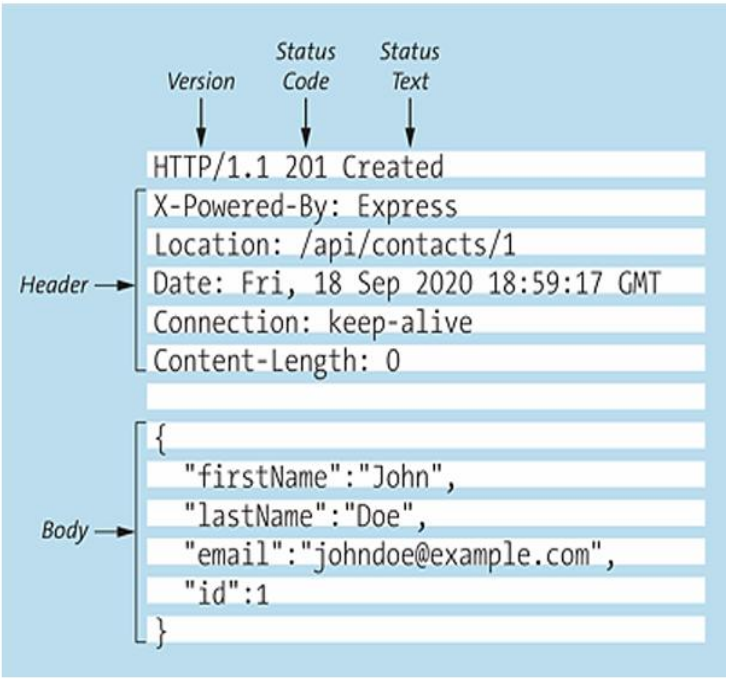


Figure 5.6 Structure of an HTTP Response

Headers follow the same key–value structure as request headers.

- ***Response Body*** Comes after the headers (separated by an empty line).Contains the actual data sent back by the server. Can include: HTML documents, CSS files, JavaScript files, Images or other data formats.

SELECTION OF FREQUENTLY USED REQUEST HEADERS

Header	Description	Example
`Accept`	Specifies which types of content the client can process.	`Accept: text/html`
`Accept-Charset`	Specifies which character sets the client can process or display.	`Accept-Charset: utf-8`
`Accept-Encoding`	Specifies which compressed formats are supported by the client.	`Accept-Encoding: gzip,deflate`
`Accept-Language`	Specifies which languages the client accepts.	`Accept-Language: en-US`
`Authorization`	Contains authentication data for HTTP authentication methods.	`Authorization: Basic bWF4bXVzdGVybWFubjp0b3BzZWNyZXQ=`
`Cookie`	Contains an HTTP cookie previously set by the server via the `Set-Cookie` response header.	`Cookie: user =johndoe;`
`Content-Length`	Specifies the length of the body in bytes.	`Content-Length: 348`
`Content-Type`	Contains the MIME type of the body (see also	`Content-Type: application/x-www-form-urlencoded`

CONTINUE...

Header	Description	Example
Date	Contains the date and time the request was sent.	Date: Tue, 23 Mar 2020 08:20:50 GMT
Host	Contains the domain name of the server.	Host: rheinwerk-publishing.com
User-Agent	Contains information about the user agent used, for example, the browser used.	User-Agent: Mozilla/5.0

Header	Description	Example
Allow	Allowed HTTP methods for the requested resource.	Allow: GET, POST
Content-Language	Language in which the resource is available.	Content-Language: en
Content-Length	Length of the body in bytes.	Content-Length: 567
Content-Location	Storage space for the requested resource.	Content-Location: /examples.html
Content-Type	MIME type of the requested resource.	Content-Type: text/html;
Date	Time of sending the response.	Date: Mon, 06 Apr 2020 08:00:00 GMT

STATUS CODE

- Status code of an HTTP response provides information about whether an HTTP request could be processed successfully or not.
- A status code is a 3-digit number
- It is followed by a short descriptive status text.

Status Code Category	Description
1xx	Request has been received and will be processed.
2xx	Request was successfully processed by the server.
3xx	Request requires further action.
4xx	Request could not be processed due to an error located on the client side.
5xx	Request could not be processed due to an error located on the server side.

CONTINUE...

Status Code	Name	Description
200	OK	The request was successfully processed by the server.
301	Moved Permanently	The requested resource has been permanently assigned a new URL.
400	Bad Request	Incorrect request
401	Unauthorized	Request cannot be processed without authorization.
403	Forbidden	Access to the requested resource is forbidden.
404	Not Found	The requested resource was not found.
405	Method Not Allowed	The HTTP method used (for example, `GET`, `PUT`, or `POST`) is not allowed for the
500	Internal Server Error	Internal server error due to which the request cannot be processed.

MIME TYPES

- To ensure that the recipient of the data sent via HTTP, whether from the server to the client or from the client to the server, knows what the data is, MIME types (more fully, Multipurpose Internet Mail Extensions types) are strings that define exactly what format the transmitted data will have.
- A MIME type always consists of a type and a subtype separated by a slash. *Example:* **type/subtype**
- The type represents the general category into which the particular data type falls, such as text, audio, or video. The subtype specifies the exact nature of the data of the specified type, for example, what kind of text (CSS, HTML, JavaScript, or plain text), what kind of audio format, or what kind of video format.
- Furthermore, an additional parameter can be attached to MIME types, so you can specify further details about the data, using the following construction: **type/subtype; parameter=value**
- For example, for MIME types of type "text," you can use the charset parameter to define which character set is used for the text: **text/plain; charset=UTF-8**

COOKIES

- HTTP handles each request independently and cannot detect whether the same user has visited before. However, many websites require identifying returning users for login management or remembering shopping cart items.
- To provide this continuity, websites use cookies, small data files stored in the browser.
- Cookies allow servers to recognize users across multiple requests and even across different days.
- Cookies are key-value pairs (or name-value pairs) that the browser stores as text files on the user's computer.
- Cookies are placed only if the user allows it and has not taken the appropriate precautions in the browser settings.
- The cookies are then transmitted to the server along with the HTTP request each time the associated web page is called.

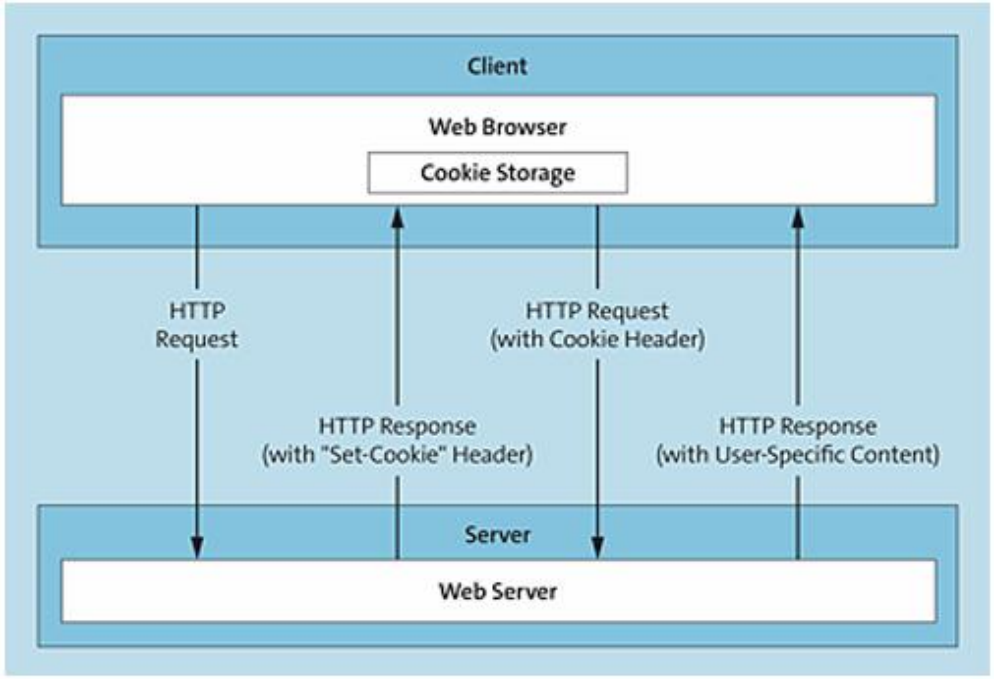


Figure 5.12 The Principle of Cookies

COOKIE FILE CONTENTS

A cookie file mainly stores two required pieces of information:

- **Name** of the cookie (case-insensitive)
- **Value** of the cookie (must be a string)

Additional optional information can also be stored:

Domain: Specifies which website the cookie is sent to.

- *Example:* A cookie for *www.fullstack.guide* is only sent to that exact domain.

Path: Restricts cookies to a specific section of a website.

- *Example:* A cookie with path */cookietests* is sent only for requests to that path.

CONTINUE...

Expiration date:

- Determines how long the cookie remains valid.
- If not set, the cookie is deleted when the browser session ends.

Secure flag:

- Ensures the cookie is sent **only over HTTPS,** not over insecure HTTP. A security flag can be used to optionally specify whether a cookie should only be sent on connections that use Secure Sockets Layer (SSL).

Overall, cookies store small pieces of data that help websites maintain state, control access, and enhance security across browsing sessions.

DISADVANTAGE OF COOKIES

- Cookies for the corresponding domain and path are sent along with each request, which has an overall (albeit minimal) impact on data volume.
- In addition, cookies that are sent via the HTTP protocol (and not via the secure HTTPS protocol) are transmitted unencrypted, which poses a security risk depending on the type of information transmitted.
- The amount of data that can be stored via cookies is limited to 4 kilobytes

INTRODUCTION OF BIDIRECTIONAL COMMUNICATION

- With HTTP, the communication between client and server is initially unidirectional, that is, it goes in one direction from the client to the server.
- The client must first send an HTTP request to the server, which then responds with an HTTP request.
- On the other hand, a server cannot send data to a client on its own via HTTP.
- Modern web applications, however, thrive on the ability for clients to be actively notified by servers about new data, for example, in timelines on social networks, in charts for displaying stock prices, or in news tickers that update automatically

CONTINUE...

Bidirectional communication can be implemented as follows:

- Polling
- Long polling
- Server-sent events (SSEs)
- WebSocket

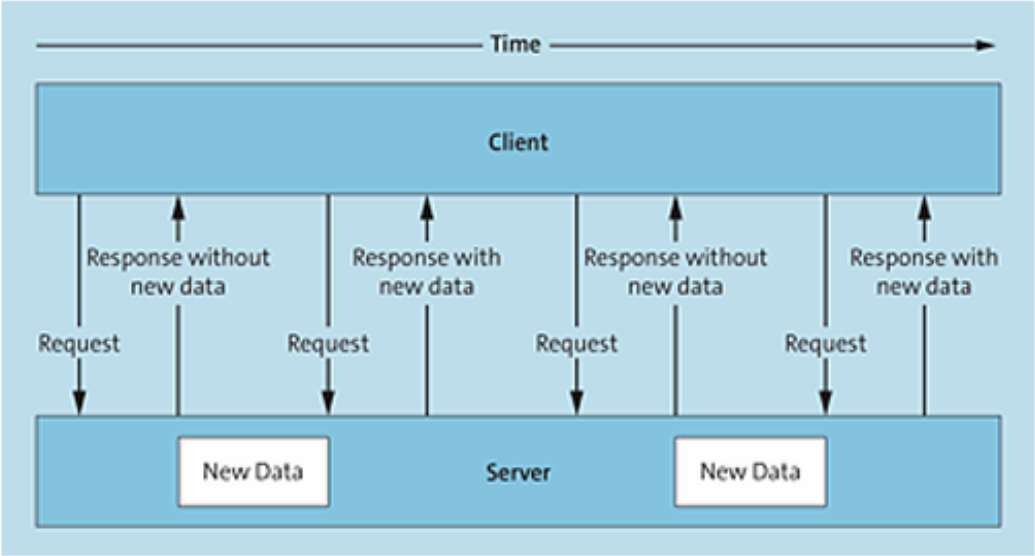


Figure 5.15 The Principle of Polling

Polling: In this approach, a client asks a server in the background (using JavaScript) for new data at regular intervals, which the server then responds to accordingly.

LONG POLLING

- In this method, the client also sends requests to the server to check whether new data is available. In contrast to normal polling, however, the HTTP connection to the server is kept open until either new data becomes available or a previously defined timeout has been exceeded, if no new data is available from the server.
- Long polling at least somewhat prevents the sending of an unnecessarily large number of HTTP requests; however, even in this case, the communication remains unidirectional. Long polling can be resource-intensive.

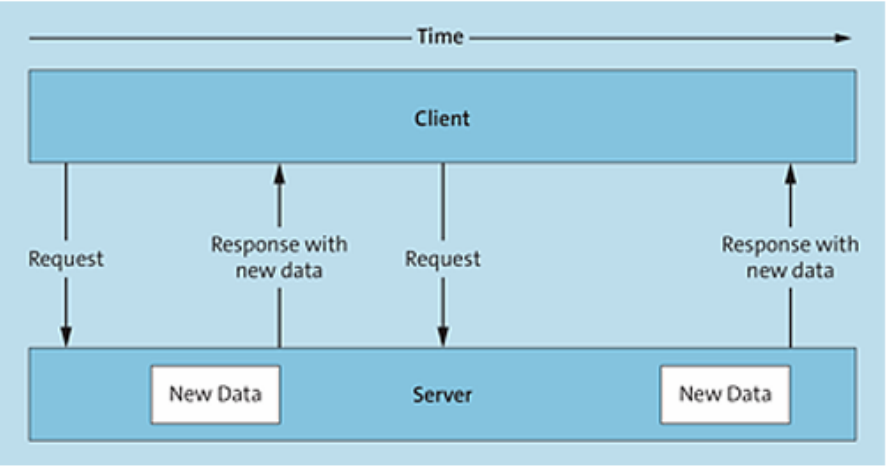


Figure 5.16 The Principle of Long Polling

SERVER-SENT EVENTS

- SSE is a technology that allows a server to actively send data to a client through an HTTP connection. For this purpose, the client must first establish an appropriate connection to the server using JavaScript. SSEs work unidirectionally.
- Once the initial connection has been established by the client, data can only be sent from the server to the client via this connection, but not from the client to the server. (*Examples* like updating a timeline, a stock quote, or a news ticker)
- When data sent from the client to the server through the same connection (bidirectional communication) is required (*Example*, when implementing a chat room), use a different technology, namely, **Proxies**.

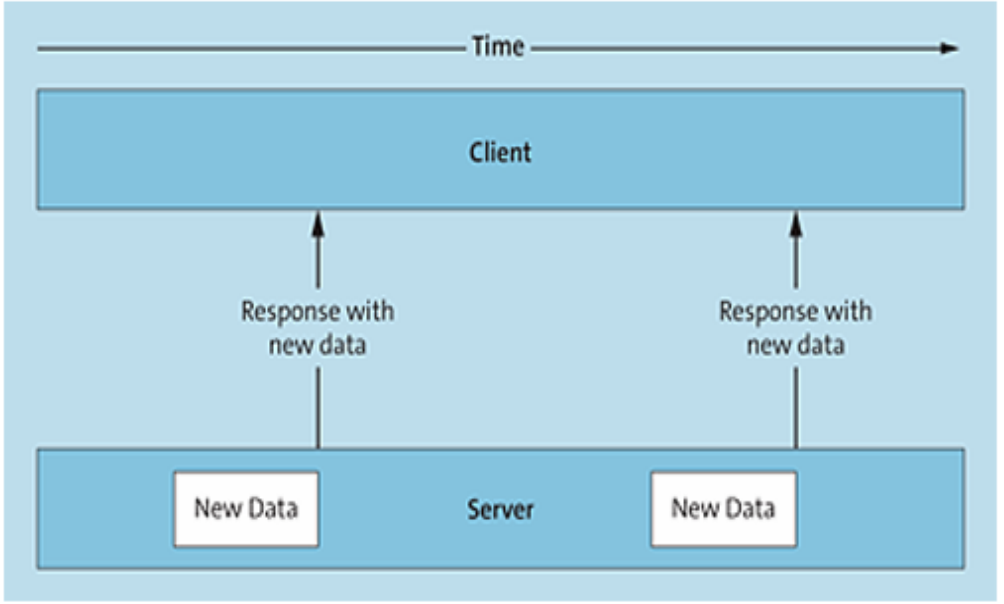


Figure 5.17 In SSEs, the Server Sends Messages to the Client

WEBSOCKET

- WebSocket protocol (a network protocol based on TCP), bidirectional connections can be established between client and server, through which the client can send data to the server and also the server can actively send data to the client.
- For this communication to be possible, the server must be a WebSocket server or a web server that supports this protocol.
- On the client side, need a WebSocket client, this client is available in all modern browsers

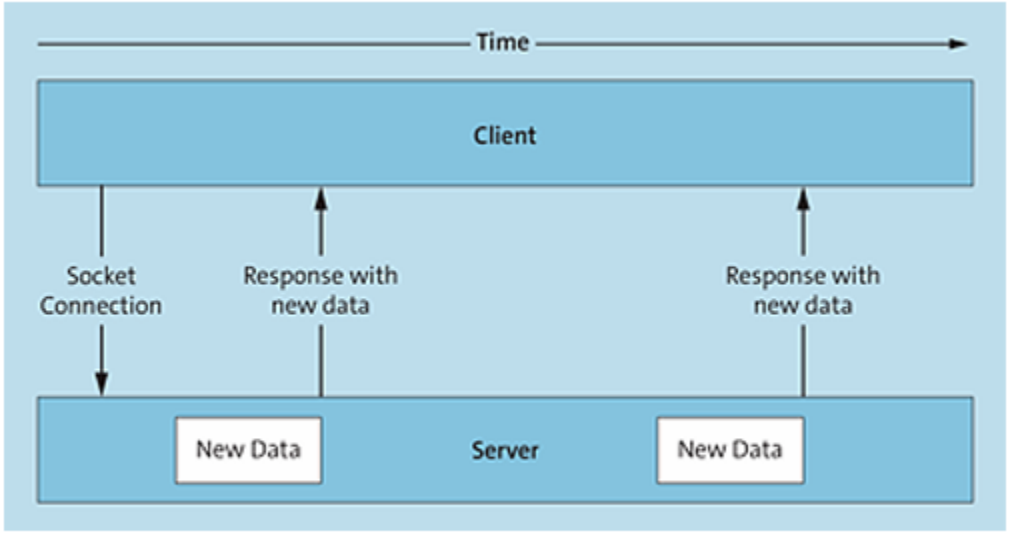


Figure 5.18 The Principle of WebSockets

WEBSOCKET VS. LONG POLLING

Features	WebSockets	Long Polling
Communication Type	Full-duplex bidirectional	Simulated bidirectional
Connection Establishment	Persistent connection	Repeatedly opened and closed
Latency	Low (real-time updates)	Variable (depends on polling frequency)
Server Load	Lower due to persistent connections	Potentially higher due to open connections
Compatibility	Requires native support	Compatible with standard HTTP
Firewall & Proxy Support	May face challenges	Firewall and proxy-friendly

CHAPTER 6
USING WEB FORMATS

DEBABRATA ROY
Asst. Prof. (Dept. of CA), ITER, SOAU
Ph.D. from **NIT Raipur**
MCA from **NIT Agartala** (*Gold Medalist*)
B.Sc(CS) from **NBU** (*Gold Medalist*)
Phone (WP): 8918671070

XML (EXTENSIBLE MARKUP LANGUAGE)

One of the most important exchange formats on the web (for example, for data exchanges between web services is the Extensible Markup Language (XML) format.

- XML is a markup language that can structure data hierarchically and is quite similar to HTML.
- XML is designed to store and transport data.
- XML is designed to be self-descriptive.
- An element can contain: text, attributes, other elements, or a mix of the above

Example:

```
<bookstore>
  <book category="children">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

XML PARSER

- XML parser is a component that converts XML code into a suitable model for further processing within the relevant programming language.
- XML parsers are available for various programming languages, and fortunately, you don't have to reinvent the wheel here and can simply fall back on appropriate libraries.
  - **XML-DOM Parsers**
    - To convert an XML document into a tree-like data structure, what's called the Document Object Model (DOM) or sometimes called a DOM tree, you would use an XML-DOM parser.
    - This parser is accessed within a program using the DOM Application Programming Interface (API)

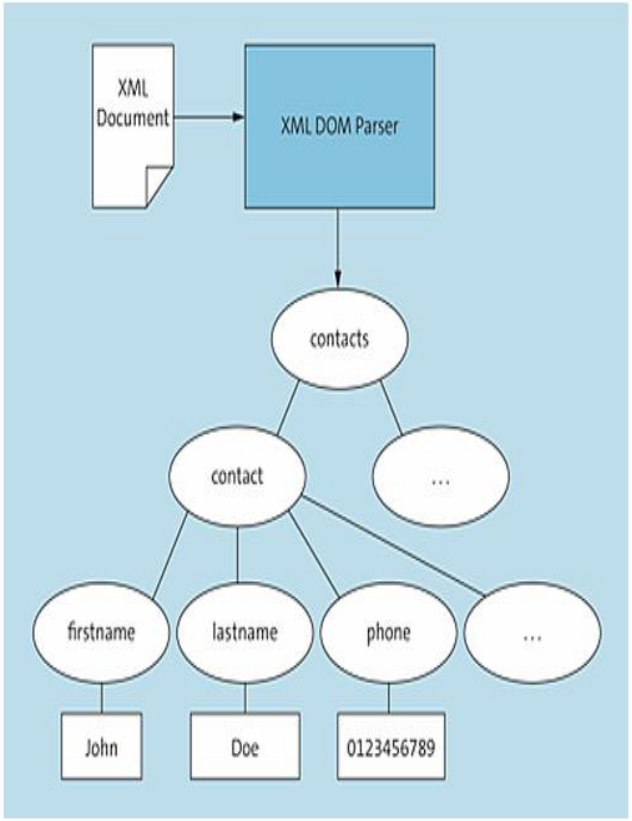


Figure 6.2 Suitable Only for Small/Medium-Sized Documents, XML DOM Parsers Convert XML Documents into Data Structures

XML DTD (DOCUMENT TYPE DEFINITION)

- DTD contains a set of rules that control the structure and elements of XML files.
- It is used to describe the attributes of XML language precisely.
- DTD mainly checks the grammar and validity of an XML document.

❑ **Features**

- It validates the structure of the XML document.
- It checks for the grammar of the XML document.
- It describes the order in which the element occurs.

❑ **Advantages**

- We can define our own format for the XML files by DTD.
- It helps in validation of XML file.

❑ **Disadvantages**

- DTDs are hard to read and maintain if they are large in size.
- It is not object oriented.

XML File: *first.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ITER SYSTEM "first.dtd">
<ITER>
   <dept>
      <ca regNo="AB123">
         <faculty serNo="1234"> D Roy </faculty>
         <faculty serNo="7867"> S Chau </faculty>
      </ca>
   </dept>
   <lab>
      <IWT subCode="CA3117"> 57 </IWT>
   </lab>
</ITER>
```

XML DTD File: *first.dtd*

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT ITER (dept,lab)>
<!ELEMENT dept (ca)>
<!ELEMENT ca (faculty+)>
<!ELEMENT faculty (#PCDATA)>
<!ELEMENT lab (IWT)>
<!ELEMENT IWT (#PCDATA)>
<!ATTLIST ca regNo CDATA #IMPLIED>
<!ATTLIST faculty serNo CDATA #IMPLIED>
<!ATTLIST IWT subCode CDATA #IMPLIED>
```

JSON (JAVASCRIPT OBJECT NOTATION)

- It is a lightweight **Data Interchange Format** that is easy for humans to read and write.
- It is easy for machines to parse and generate.
- JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on 2 Structures:

- A ***collection of name/value pairs***. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ***ordered list of values***. In most languages, this is realized as an array, vector, list, or sequence.

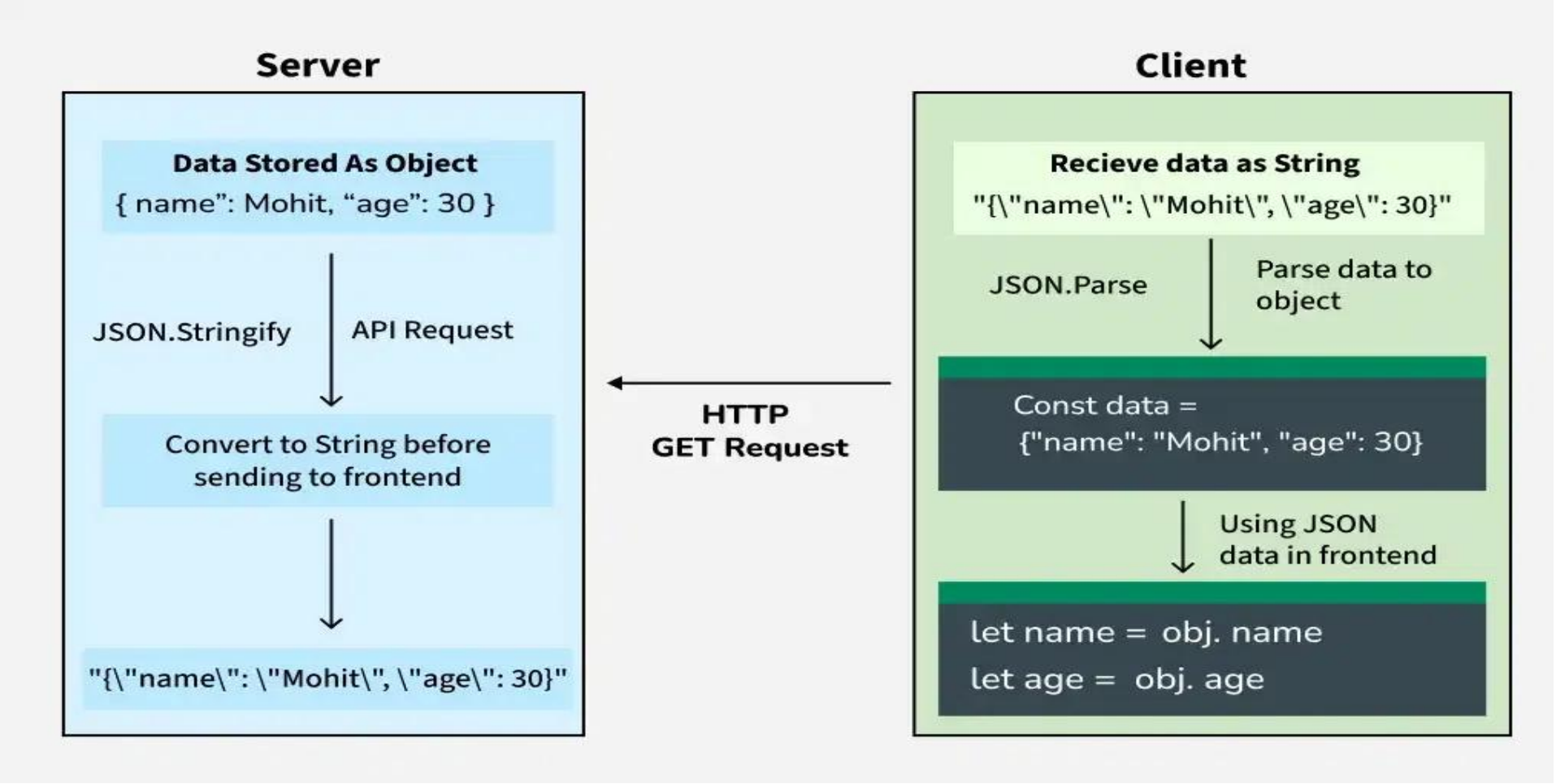
- These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

CONTINUE...

- JSON is suitable for the structured definition of data and is also commonly used as a data exchange format.
- In contrast to XML, however, JSON is much leaner and can be processed much more easily within JavaScript code.
- An essential feature of the JSON format is its curly brackets, which define individual objects.
- Object properties (also keys) are written in double quotes and separated from their values by a colon. *Example:*

```
{
  "message": "Hello World“
}
```

JSON DATA FLOW: SERVER → CLIENT



JSON PARSERS

- To process JSON documents, a suitable JSON parser is needed.
- As with XML, corresponding libraries for JSON exist for various programming languages that we can use for this purpose.
- In the case of JavaScript, parsing JSON documents is even natively built into the language, which means that no need for any external libraries.
- Instead, directly convert a JSON string into a JavaScript object using the JSON.parse() method.

Converting a JSON Text to a JavaScript Object

```
<html>
<body>
    <h2>Create Object from JSON String</h2>
    <p id="demo"></p>
    <script>
        var text = '{"employees":[' +
            '{"firstName":"John","lastName":"Doe" },' +
            '{"firstName":"Anna","lastName":"Smith" },'
+
            '{"firstName":"Peter","lastName":"Jones" }]}';
        obj = JSON.parse(text);
        document.getElementById("demo").innerHTML =
        obj.employees[1].firstName + " "
        + obj.employees[1].lastName;
    </script>
</body>
</html>
```

JSON SCHEMA

- It is a content specification language used for validating the structure of JSON data.
- It helps to specify the objects and what values are valid inside the object's properties.
- JSON schema is useful in offering clear, human-readable, and machine-readable documentation.

Structure of a JSON Schema: Since JSON format contains an object, an array, and name-value pair elements. Name-value pairs are used for providing schema processing elements and validating the JSON content. Schema processing elements include(not limited to).

- **"$schema"** - To specify the version of JSON schema.
- **title** and **description** - To provide information about the schema.
- **required**- It's an array of elements that indicates which elements should be present.
- **additionalProperties**- To indicate whether the existence of specified elements is allowed or not.

SAMPLE OF JSON SCHEMA

Schema:

```
{
  "$id": "https://example.com/person.schema.json",
  "$schema": "https://json-schema.org/draft-07/schema",
  "title": "Voters information",
  "type": "object",
  "properties": {
    "firstName": {
      "type": "string",
      "description": "The person's first name."
    },
    "lastName": {
      "type": "string",
      "description": "The person's last name."
    },
    "age": {
      "description": "Age in years which must be equal to or
                      greater than eighteen in order to vote.",
      "type": "integer",
      "minimum": 18
    }
  }
}
```

Output:

```
{
  "firstName": "Indra",
  "lastName": "Sen",
  "age": 20
}
```

IMAGE FORMATS

- To include photographs or images with high color depth or dynamics (i.e., images with many different colors), a best practice is to use the JPG format (also JPEG for Joint Photographic Experts Group).
- This format can display up to 16 million colors and also supports different compression levels between the following two extremes, which affect the file size of the image as well as its quality:
  - **0% Compression:** Lowest compression, therefore no quality loss but also unchanged file size
  - **100% Compression:** Strongest compression, smallest possible file size but loss of quality

GRAPHICS AND ANIMATIONS IN GIF FORMAT

- The GIF format (meaning Graphics Interchange Format) is particularly suitable for images with a few colors or large monochrome areas, as it can only represent 256 colors (8 bits are available per pixel, or 28 possible values). For such images, the GIF format compresses the data more effectively than the JPG format.

- Compared to other formats, GIF can also do something special: You have probably heard or read the term animated GIF. These small animations are saved frame by frame in a single GIF. Animated GIFs have gained massive importance in recent years, especially through platforms like Giphy (https://giphy.com) and social media in general.

VECTOR GRAPHICS IN SVG FORMAT

- JPG, GIF, and PNG are bitmap formats. Bitmaps consist of small rectangles (pixels), each of which has a color value. All bitmap formats depend on the resolution, so they cannot be scaled arbitrarily without the individual pixels becoming larger and thus visible. You can test this property by opening an image in one of the formats mentioned earlier in a browser and then enlarging the display (and thus the image)
- A vector format that can be displayed by all modern browsers is the Scalable Vector Graphics (SVG) format, an XML-based format for describing vector graphics. The name says it all: SVG graphics are scalable. Thus, once you create a vector graphic, you can scale it to any size without losing quality in the rendering.
- For including video and audio files, the ***<video>*** and ***<audio>*** elements have been available since HTML5.
- Many different video and audio formats exist, but thanks to HTML elements, you can include multiple formats for a single video or audio file.

CHAPTER 7
USING WEB APIS

DEBABRATA ROY
Asst. Prof. (Dept. of CA), ITER, SOAU
Ph.D. from **NIT Raipur**
MCA from **NIT Agartala** (*Gold Medalist*)
B.Sc(CS) from **NBU** (*Gold Medalist*)
Phone (WP): 8918671070

CHANGING WEB PAGES DYNAMICALLY USING THE DOM API

- When you open a webpage, the browser sends an HTTP request to the server.
- The server responds by sending back HTML code.The browser parses the HTML and converts it into a structured model stored in memory. This structure is called the ***Document Object Model (DOM)***.
- The DOM represents all elements of the webpage as objects. JavaScript can be used to access, change, and interact with the DOM.

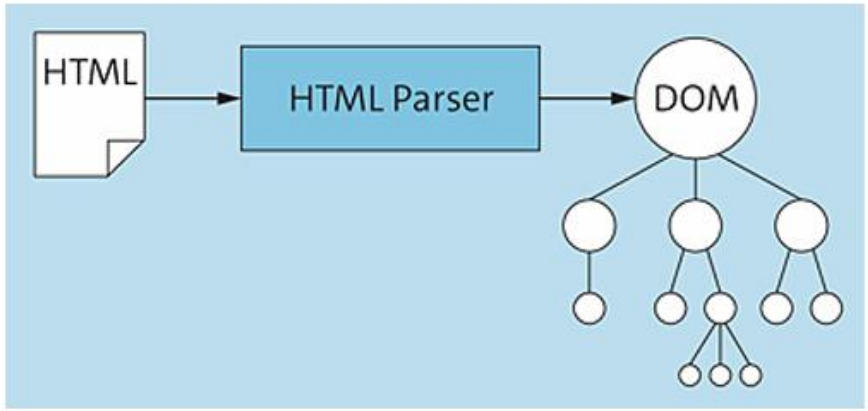


Figure 7.2 Browsers Parse HTML Documents into Their Own Object Model

CONTINUE…

- The DOM represents the components of a web page (i.e., HTML elements and HTML attributes) hierarchically as a tree, known as the DOM tree. Such a DOM tree is composed of nodes, whose hierarchical arrangement reflects the structure of a web page.
- The DOM API thus defines a programming interface to access the DOM tree through a program
- The DOM API provides a set of objects (with methods) through which the content of a web page (or more generally, the content of HTML documents) can be accessed.
- Implementations of the DOM API exist for various programming languages (including Java, Python, and C#). However, in the following sections, we'll focus on the implementation for JavaScript, which is implicitly available to you in every browser

AJAX (ASYNCHRONOUS JAVASCRIPT AND XML)

AJAX (Asynchronous JavaScript and XML) is a technique used in web development that enables web applications to fetch and send data to a server without requiring the webpage to reload. It allows for a seamless and dynamic user experience.

Features:

- ✅ Asynchronous communication → No need to reload the page
- ✅ Faster response times → Improves user experience
- ✅ Uses JavaScript to send/receive data → Works with XML, JSON, or plain text
- ✅ Works with various technologies → HTML, CSS, JavaScript, DOM, XML/JSON, XMLHttpRequest

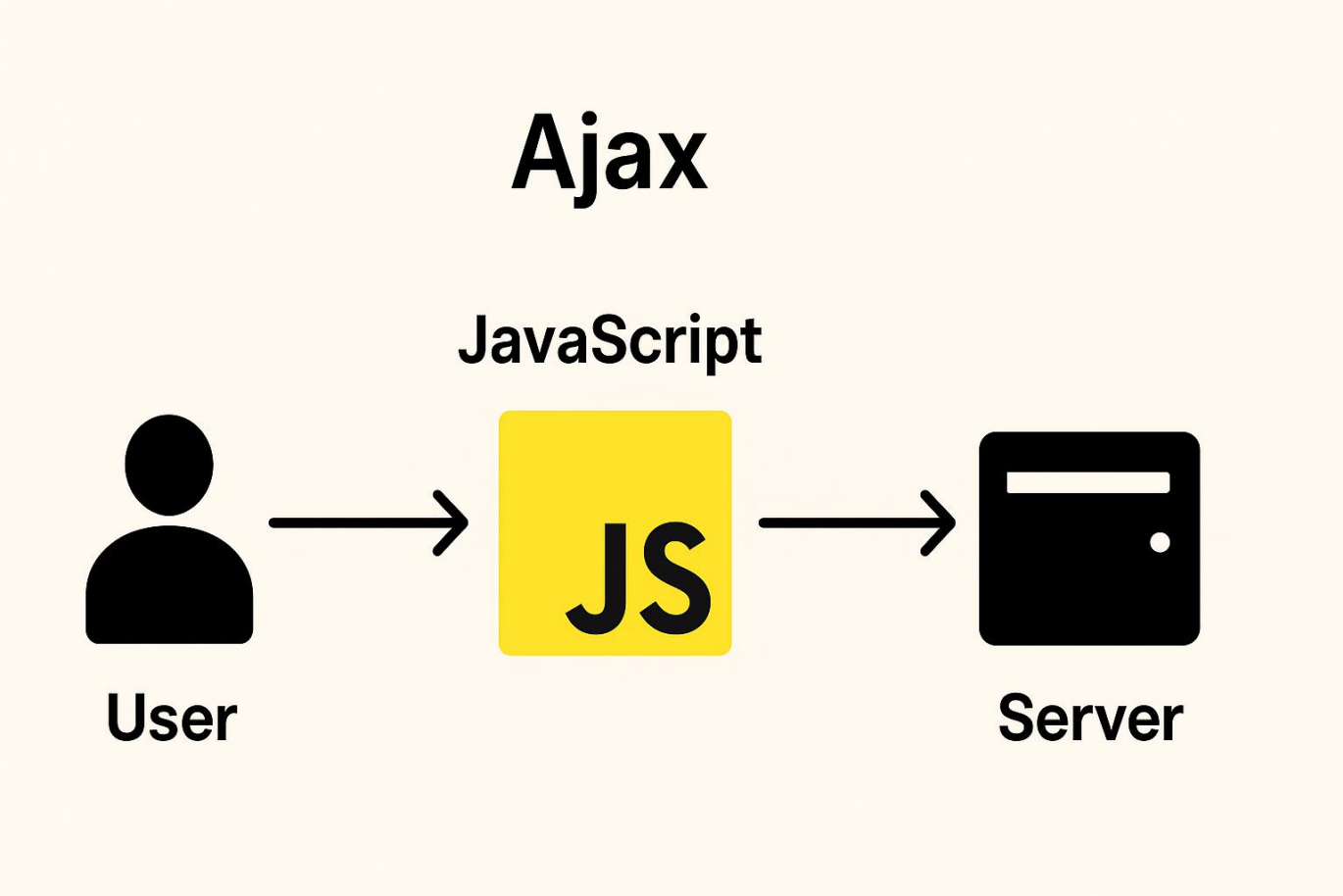
Working Principle:

- A user interacts with the webpage (e.g., clicks a button).
- JavaScript creates an AJAX request to the server.
- The server processes the request and sends data back (XML, JSON, or plain text).
- JavaScript updates the webpage without reloading.

LOADING DATA SYNCHRONOUSLY VIA AJAX AND FETCH API

- When you modify a webpage using the DOM API, the content changes without reloading the page.
- But sometimes you need new data from the server (for example: messages, search results, notifications).
- Normally, loading server data would refresh the whole page, which is slow and inefficient.
- Ajax solves this problem. Ajax enables web pages to request data from a server asynchronously (in the background) without requiring the entire page to reload.
- It enables dynamic, fast, and interactive websites.
- The idea is to use JavaScript to make HTTP requests to the server without completely reloading the web page itself.

AJAX FLOW DIAGRAM



User → JavaScript → Server
User ← JavaScript ← Server

[User Action]
↓
[JavaScript Ajax Request]
↓
[Server Processes Data]
↓
[Server Response (JSON)]
↓
[JavaScript Updates the DOM]
↓
[Page Changes Without Reload]

SYNCHRONOUS COMMUNICATION

- In a traditional web environment, communication between the client and server occurs using the HTTP protocol.
- The client initiates communication by sending an HTTP request to the server.
- The server receives the request, processes it, and returns an appropriate response.
- Actions such as clicking a hyperlink or submitting a form trigger a new HTTP request. The server generates the required content for the next web page and sends it back to the client.
- The communication model is synchronous, meaning the user cannot interact with the webpage while the request is being processed.
- The user must wait until the server completes processing the request, the response is sent back to the client, and the browser receives and renders the updated content.

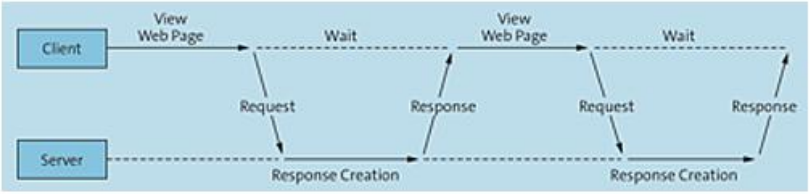


Figure 7.6 The Principle of Synchronous Communication

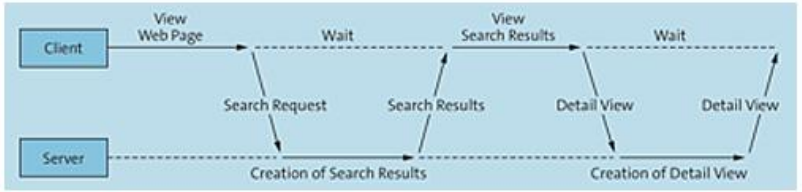


Figure 7.7 Sequence for a Synchronously Implemented Search

ASYNCHRONOUS COMMUNICATION

- Asynchronous communication allows a web page to remain interactive while server requests are being processed.
- Unlike the synchronous model, the client can send multiple requests without waiting for previous ones to finish.
- Responses are handled independently as they arrive, enabling smoother, more user-friendly interactions—such as updating parts of a page dynamically without reloading the entire webpage.

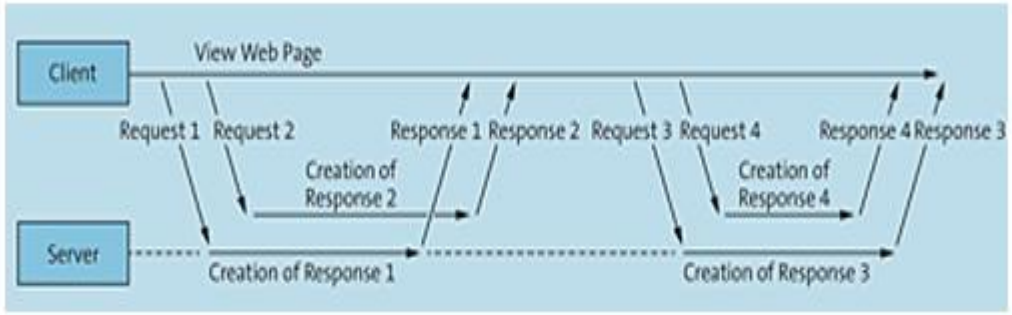


Figure 7.8 The Principle of Asynchronous Communication

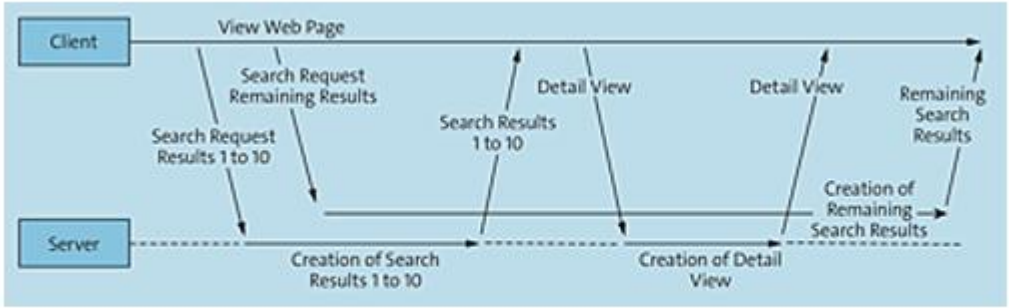


Figure 7.9 Sequence for an Asynchronously Implemented Search

Synchronous Vs. Asynchronous Communication

- Technically, HTTP requests are also sent to the server during asynchronous communication. However, in synchronous communication, the HTTP requests and responses are executed or processed directly by the browser (for example, when clicking on a link).
- In contrast, in asynchronous communication, this communication occurs in the background using JavaScript

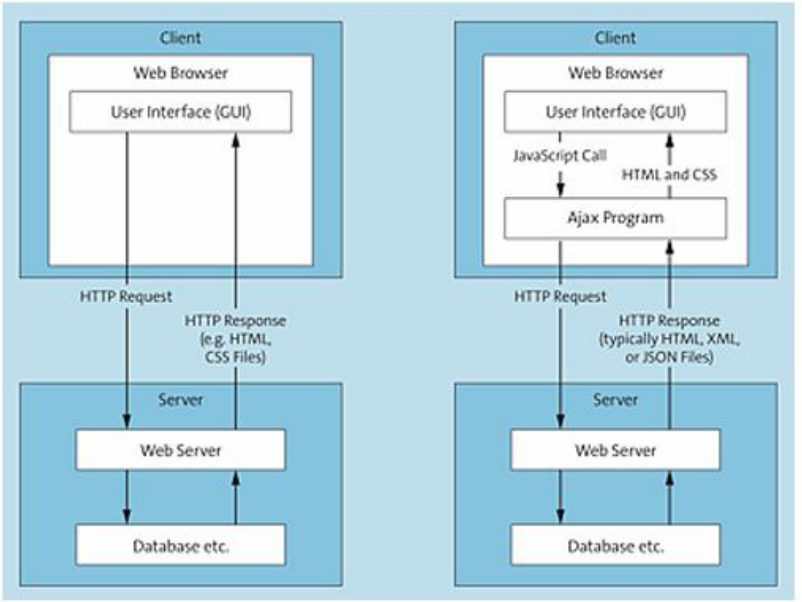


Figure 7.10 Difference between Synchronous and Asynchronous Communication

USE OF AJAX IN WEBSITES

AJAX is widely used in modern web development for dynamic and responsive applications.

- **Live Search (Autocomplete Feature)** Google Search suggestions appear dynamically as you type. *Example:* htmlCopyEdit

```
<input type="text" id="searchBox" onkeyup="showSuggestions(this.value)" placeholder="Search..."> <div id="suggestions"></div>
```

The showSuggestions() function makes an AJAX call to fetch relevant search terms dynamically.

- **Form Submission Without Reloading** Prevents full-page refresh when submitting a form. *Example:* Login forms, feedback forms.

Example Code (JavaScript with AJAX):

```
javascriptCopyEditfunction submitForm() {
    var xhr = new XMLHttpRequest(); xhr.open("POST", "submit.php", true);
    xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    xhr.onreadystatechange = function() { if (xhr.readyState == 4 && xhr.status == 200)
    {document.getElementById("response").innerHTML = xhr.responseText; } };
    var formData = "name=" + document.getElementById("name").value; xhr.send(formData);
}
```

CHAPTER 8
OPTIMIZING WEBSITES FOR ACCESSIBILITY

DEBABRATA ROY
Asst. Prof. (Dept. of CA), ITER, SOAU
Ph.D. from **NIT Raipur**
MCA from **NIT Agartala** (*Gold Medalist*)
B.Sc(CS) from **NBU** (*Gold Medalist*)
Phone (WP): 8918671070

WEBSITES FOR ACCESSIBILITY

Accessibility, specifically the accessibility of web pages (web accessibility), means that websites are built and implemented in such a way that they are accessible for all users, especially for people with disabilities.

The more accessible a website is, the more accessible it becomes for other user groups, such as the following:

- People who use cell phones, smartwatches, smart TVs, or other devices with small screens to access websites
- Older users with altered abilities due to aging
- Users with "temporary limitations," such as broken arms
- Users with "situational limitations," such as bright sunlight or being in an environment where they cannot hear sound
- Users with slow internet connections and/or limited or expensive bandwidth

SEARCH ENGINE OPTIMIZATION AND ACCESSIBILITY GUIDELINES

- In addition, many techniques that lead to accessible web pages or an accessible website also improve how a website rates with search engines through **Search Engine Optimization** (SEO).
- Thus, by being accessible, your website gains a higher rating and improves its position (ranking) within search results.

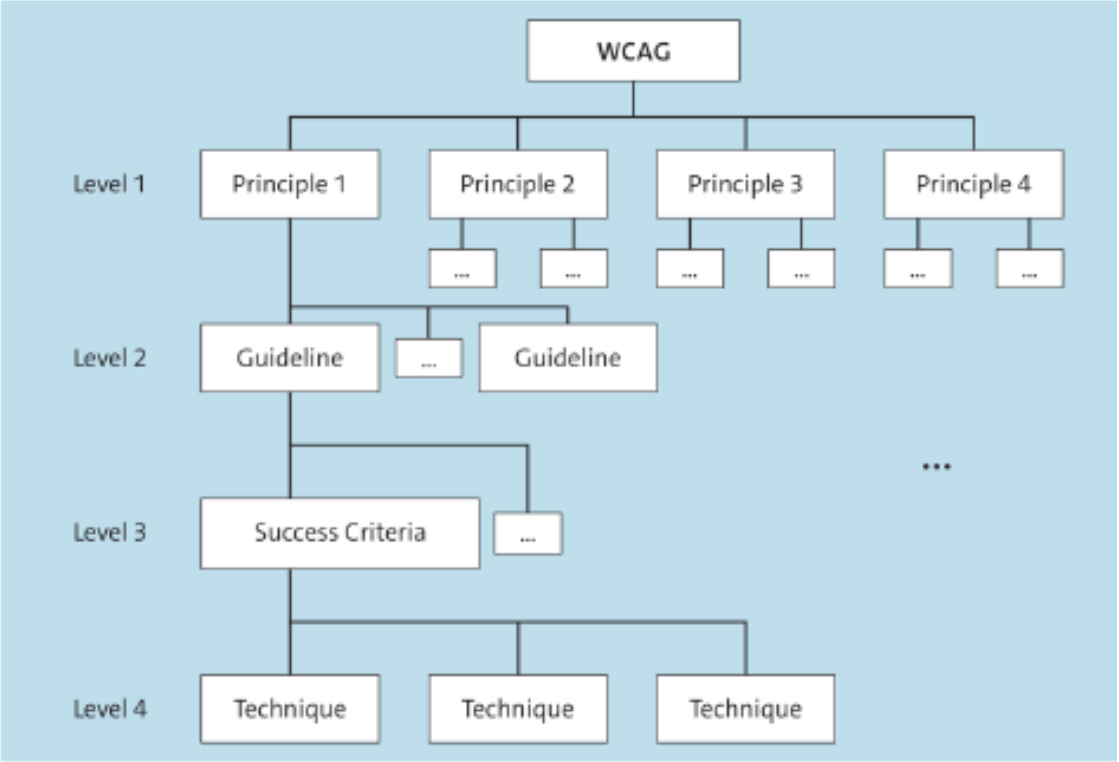


Figure 8.2 Web Content Accessibility Guidelines Hierarchical Structure

ACCESSIBILITY GUIDELINES (CONTINUE…)

Level 1: Principles

The top level comprises four principles, which form the basis of the guidelines. These principles define which basic properties a website must fulfill irrespective of the technology:

- ***Principle 1:*** Perceivable Information and user interface (UI) components must be provided or be presentable to all users in a way they can perceive.
- ***Principle 2:*** Operable UI and navigation components must be functional and operable for users.
- ***Principle 3:*** Understandable Information and the operation of the UI must be understandable for users.
- ***Principle 4:*** Robust Content must be robust enough to be interpreted by a variety of user agents, including assistive technologies.

ACCESSIBILITY GUIDELINES (CONTINUE…)

Level 2: Guidelines

The four principles, in turn, are assigned a total of thirteen guidelines, which represent the essential goals for making a website accessible.

Guideline 1.1: Text alternatives ***Guideline 1.2***: Time-based media ***Guideline 1.3***: Adaptable

Guideline 1.4: Distinguishable

Guideline 2.1: Keyboard accessible ***Guideline 2.2***: Enough time ***Guideline 2.3***: Seizures and physical reactions

Guideline 2.4: Navigable ***Guideline 2.5***: Input modalities

Guideline 3.1: Readable ***Guideline 3.2***: Predictable ***Guideline 3.3***: Input assistance

Guideline 4.1: Compatible

ACCESSIBILITY GUIDELINES (CONTINUE…)

Level 3: Success Criteria

- These guidelines are each further subdivided into success criteria, which contain specific instructions for implementing the guidelines.
- Textual variants for audio and video content
  - Subtitles for videos
  - Sufficient contrast between foreground and background color
  - Supporting the scalability of text
  - Description of the target of links

Level 4: Techniques

- The fourth and final level of WCAG is concerned with concrete techniques that describe how these principles, guidelines, and success criteria can be implemented technically.
- For example, some techniques are related to HTML, the use of Cascading Style Sheets (CSS), or even the implementation of PDF documents.

STRUCTURING WEB PAGES SEMANTICALLY

Webpages are often similar in terms of their structures: For example,

- Most web pages have a **Header Area** (which contains the company logo, for example)
- **Footer Area** (which contains links to legal notes, a contact form, or links social networks, for example)
- **Navigation Area** (where the main navigation is located)
- **Main Area** (which contains the main content of the respective web page)
- **Marginal Areas** that provide additional information.

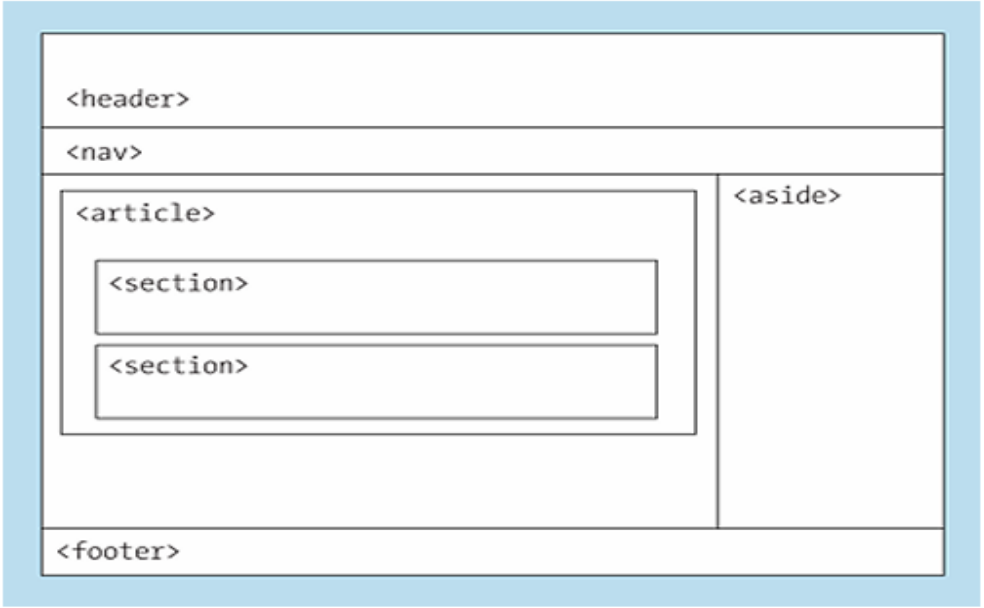


Figure 8.3 The Basic Structure of Many Web Pages

USING HEADINGS CORRECTLY

- Use headings in a correct hierarchical order to structure webpage content.
- Proper heading structure is important for accessibility, especially for screen reader users who rely on headings for navigation.
- Each webpage should contain only one <h1> heading, representing the main title or topic of the page.
- Under <h1>, you may use multiple subheadings, but they must follow a logical sequence (e.g., <h2> → <h3> → <h4>).
- Avoid skipping heading levels (for example, placing <h4> directly under <h2>), as it can confuse users and disrupt the structure.
- A well-organized heading structure improves readability, usability, and overall webpage organization.

MAKING FORMS ACCESSIBLE

- Regarding the accessibility of forms, the most important aspect is that it is clear which label (i.e., which caption) belongs to which form element.
- This clarity is vital because only through this mapping can a screen reader user, for example, recognize what the purpose of a form element is. Without an explicit mapping, the screen reader would try to "guess" the meaning of a form element from the context.
- The explicit assignment of a label to a form element can be done in several ways, including the following:
  - By assigning the form element an ID (id attribute) and referring to this ID in the corresponding label via the for attribute
  - By defining the respective form element as a child element of the label
  - By using the title attribute of the respective form element and defining a caption
  - By using Accessible Rich Internet Applications (ARIA) attributes
  - In addition, you should group logically related form elements using the <fieldset> element and define a suitable heading for each group by using the <legend> element

MAKING TABLES ACCESSIBLE

- Basically, the HTML elements available for defining tables are semantic elements. Thus, for example, browsers (and other user agents such as screen readers) will recognize directly from the element that a <table> element is actually a table. In the same way, they recognize a table heading by the <th> element and a row of a table by the <tr> element.
- The scope attribute, which can be assigned to <th> elements, specify whether the respective table heading refers to the associated column or the associated row (i.e., whether the heading is a vertical or horizontal heading)
- By default, a table heading refers to the associated column (so the heading is vertical), which means that, in this case, the scope attribute is optional.
- A better approach is to explicitly specify the col value anyway to accommodate screen readers and get used to the correct usage directly yourself.

DEFINING LANGUAGE FOR WEB PAGE

- The explicit specification of the language helps screen readers, for example, to directly select a suitable intonation for the speech output.
- Basically, you should define which language is used for a web page via the lang attribute.
- You can use the attribute both directly on the <html> element to define the language for the entire web page and on individual HTML elements to define the language for the text below that element.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <title>Sample page</title>
  <meta charset="utf-8">
</head>

<body>
  This is an English web page.
  <p lang="de">Aber dieser Abschnitt hier wurde auf Deutsch verfasst.</p>
</body>

</html>
```

Listing 8.14 Defining the Language of the Web Page and for a Single HTML Element

PROVIDING KEYBOARD SUPPORT

- A web page should always be fully navigable with the keyboard. Users who do not use a mouse to navigate a web page, but only a keyboard, often use the (Tab) key to navigate from one UI element to the next or to focus on the next UI element.
- In addition, the access key attribute provides the user with keyboard shortcuts for important functionalities of a web page, such as form elements or links.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <title>Keyboard shortcuts</title>
  <meta charset="utf-8">
</head>

<body>
  <a
    href="https://www.rheinwerk-publishing.com/"
    accesskey="1"
    tabindex="1"
  >
    Click here for the Rheinwerk Publishing website
  </a>

  <form action="/api/process-form" method="post">
    <label
      for="name"
      tabindex="2"
    >
      Name
    </label>
    <input
      type="text"
      id="name"
      accesskey="n"
      tabindex="3"
    >
    <input
      type="submit"
      id="submitform"
      accesskey="s"
      tabindex="4"
      value="Submit"
    >
  </form>
</body>

</html>
```

Shortcuts and Tab order for Link

Shortcuts and Tab order for Form

Listing 8.15 Defining Keyboard Shortcuts and Tab Order for Links and Form Elements

CHAPTER 11

IMPLEMENTING MOBILE APPLICATIONS

DEBABRATA ROY
Asst. Prof. (Dept. of CA), ITER, SOAU
Ph.D. from **NIT Raipur**
MCA from **NIT Agartala** (*Gold Medalist*)
B.Sc(CS) from **NBU** (*Gold Medalist*)
Phone (WP): 8918671070

MOBILE APPLICATIONS

Basically, mobile applications are divided into the following 3 types:

- **Native Applications**

  These applications are developed and compiled specifically for a particular mobile operating system (for example, Android or iOS) and run natively on that operating system.

- **Mobile Web Applications**

  These web applications (with the emphasis on "web") are developed like "ordinary" web applications with Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript and run in mobile web browsers.

- **Hybrid Applications**

  These applications combine the two other types, which means that hybrid applications run natively on an operating system but are implemented using HTML, CSS, and JavaScript.

NATIVE APPLICATIONS

Native applications are mobile applications developed and compiled specifically for a mobile operating system, allowing them to utilize the device's functionalities and features to their fullest potential.

Technologies

- Native mobile apps require different technologies depending on the operating system.
- Android apps are typically built using Java or Kotlin with the Android SDK.
- iOS apps (iPhone/iPad) are developed using Swift or Objective-C.
- Windows Mobile apps can be created using Visual C#.

Advantages

- Functionalities and features of the end device or mobile operating system can be optimally used because these features are made accessible via corresponding programming interfaces for the various programming languages.
- Perform much better than other types of applications because the entire application is compiled for the specific operating system before execution

CONTINUE...

Disadvantages

- Each mobile operating system (Android, iOS, Windows Mobile) requires a separate version of the app, meaning multiple codebases and specialized developers (e.g., Java/Kotlin for Android, Swift/Objective-C for iOS, C# for Windows).
- It is a high development effort.
- This increases both workload and costs, making native apps more expensive compared to other mobile app development approaches.

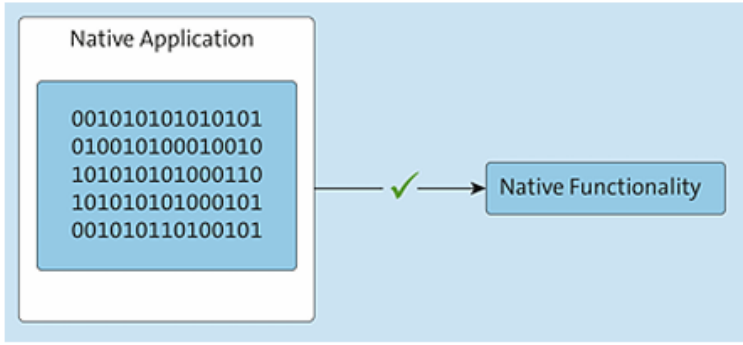


Figure 11.2 Native Applications Must Be Developed for Each Operating System but Can Access Native Functionalities with High Overall Performance

MOBILE WEB APPLICATIONS

Mobile web applications (mobile web apps with emphasis on "web") are ordinary web applications optimized for use on mobile devices.

Technologies

- Mobile web applications are developed in HTML5, CSS3, and JavaScript using responsive design principles and then run on the respective end device within a browser installed there.
- Special techniques and standardized Application Programming Interfaces
- (APIs) allow the application to be implemented in such a way that it feels as much as possible like a "real" mobile application.

Advantages

- Updates can be carried out relatively easily because the updates don't need to occur on the end device, only centrally on the corresponding web server, which will then have a direct effect on all "distributed installations."
- Only need to create one version of the application, which is then executable on all common mobile operating systems. This approach reduces the initial development effort and associated development costs on the other hand.
- "Only" HTML, JavaScript, and CSS knowledge is required from the developer to implement this type of application, as opposed to requiring knowledge of Java, Swift, or C#.

CONTINUE…

Disadvantages

- Access to native hardware functions like GPS, cameras, microphones, or similar is restricted for security reasons.
- Cannot be installed on the end device like a native application, but must always be executed in the browser.
- The user does not really get the feeling of a "real" native application.

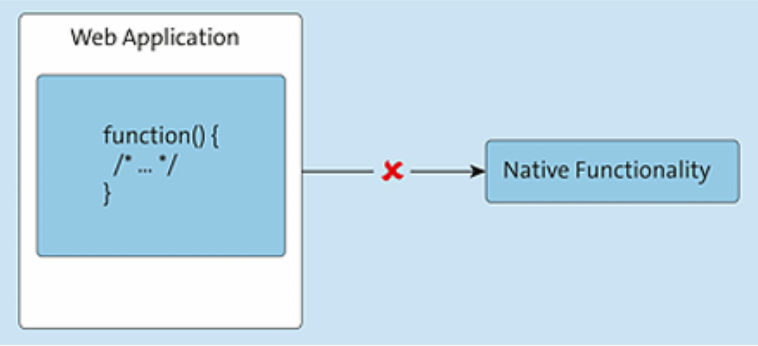


Figure 11.3 Mobile Web Applications Usually Cannot Access Native Functionalities

HYBRID APPLICATIONS

- Hybrid applications combine features of both native and web apps.
- They use a native "wrapper" that contains a web view—special components in Android or iOS that can display web content. The app's structure is native, but most of its interface and functionality are built using web technologies such as HTML, CSS, and JavaScript. This allows hybrid apps to blend native capabilities with the flexibility of web applications.
- In other words, in hybrid applications, the "frame" of the application is presented natively, but the actual content is presented as a web application.

Technologies

- Basically, two categories of hybrid applications exist: One variant requires a native part that forms the "frame" of the application and provides, among other things, what's called a web view component in which the actual application is loaded as a web application.
- This web application is developed classically in HTML, CSS, and JavaScript. In the other variant, the application is also developed in HTML, CSS, and JavaScript, but then partially "translated" into corresponding native components of the mobile operating system during "building."
- Various tools are available for developing web applications. The more modern ones include Flutter (https://flutter.dev), NativeScript (https://nativescript.org), Ionic (https://ionicframework.com), and React Native (https://reactnative.dev)

CONTINUE…

Advantages

- Can be installed on a mobile device like a native application
- Largely platform independent
- "Only" knowledge of HTML, CSS, and JavaScript is required for development.

Disadvantages

- Hybrid application development frameworks provide APIs for various hardware features and mobile platform features that are kept as generic as possible (i.e., using the lowest common denominator of different mobile platforms).
- Even though hybrid apps provide many native features through special APIs that connect to JavaScript, they still cannot use every capability of the device. In other words, hybrid apps do not fully access all platform features the way native apps do.

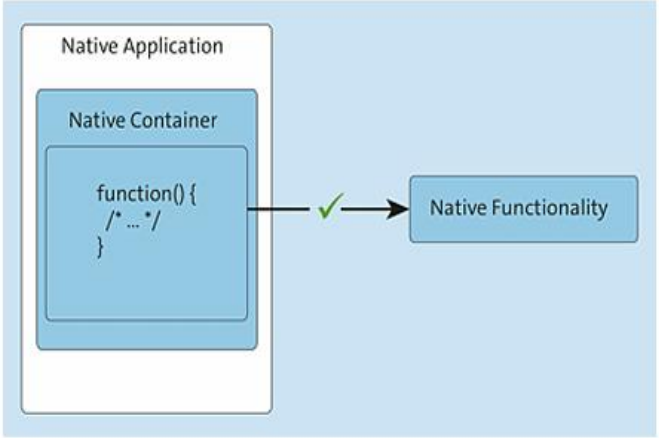


Figure 11.4 Hybrid Applications: Native Container Allowing a Native Functionality to Be Accessed from the JavaScript Code

COMPARING DIFFERENT APPROACHES

Aspect	Native	Web	Hybrid
Development	A separate application must be created for each platform (Android, iOS, etc.). Alternatively, you can use a library like React Native.	Only one application needs to be developed, which can then be run on all platforms.	As with mobile web applications, all that is required is to develop an application, which can then be installed in the same way as a native application thanks to native components.
Programming languages and other languages	Objective-C, Swift, Java with Android Software Development Kit (SDK), Visual C#, or JavaScript or JSX in the case of React Native.	HTML5, CSS3, and JavaScript	HTML5, CSS3, and JavaScript plus appropriate native components per platform.

Aspect	Native	Web	Hybrid
Platform independence	No	Yes	Yes
Development effort and costs	High	Low	Medium
Maintenance effort and costs	High	Low	Medium
Reusability of the source code	Low	High	High

THANK YOU!

Best of luck for your upcoming Examination