1. WEB 3.0 - THE DECENTRALIZED WEB (NEXT EVOLUTION)

Defining Web 3.0 is challenging because its exact origin point is debatable - some argue it's already here, others say it's years away. However, key anticipated features are emerging:

Fundamental Principles:

- Human-centric design: Giving users back control over large tech businesses
- Users reclaim total control over their own data.
- Enhanced security: User data is safeguarded via encryption
- Moving away from data concentration in corporate "behemoths with questionable motives" is known as decentralisation.

Technological developments:

- Interoperability: Applications become independent of devices.
- Smooth connectivity between smartphones, TVs, automobiles, and household equipment is known as the Internet of Things (IoT).
- Cross-platform functionality: applications that can operate on any kind of hardware.

THE VISION

Matteo Gianpietro Zago, the creator of Essentia.one, claims that Web 3.0 would address the privacy and security issues that arose during Web 2.0 by returning control and data "to the rightful owners"—the users themselves.

Important distinctions:

Web 1.0:

- User Role: Consumer who can only read
- There is no interaction.
- Content: Static
- Webmasters have authority over the power structure.
- Technology: Simple HTML
- For instance, the 1990s and early 2000s

Web 2.0:

- User Role: Participant and content creator
- High levels of interactivity (social, cooperative)
- Content: User-generated and dynamic
- Power Structure: Data is controlled by IT corporations
- Technology: SaaS, social media
- For instance, the mid-2000s to the present

Web 3.0:

- User Role: Controller and Owner of Data
- Interactivity: Cross-device seamless
- Content: Decentralised and customised
- Users have control over their data in the power structure.
- Technology: IoT integration, blockchain, and artificial intelligence
- For instance, the Emerging/Future Era

2. FUNCTIONAL DIFFERENCES BETWEEN FRONT-END AND BACK-END

Front-end (Layer of Presentation):

What you see on a website is called the front-end. It is the area that users view and select. Your web browser is where it operates. Front-end developers create a visually appealing website. HTML, CSS, and JavaScript are the three tools they employ. The structure is built via HTML. CSS uses fonts and colours to make it aesthetically pleasing. When you click, JavaScript causes things to move and react. Creativity is necessary for front-end job. Design and user experience are important considerations.

Data Access Layer (BACK-END):

Users cannot see the back-end. You are unable to see it. It operates on a distant server. The website is created by back-end developers. They make use of languages like Ruby, PHP, and Python. Databases are another tool they use. Your data is saved on the back end. When you need data, it gets it. It handles all the arithmetic and reasoning behind the scenes. Logic is necessary for back-end work. You need to consider data storage and speed.

Important distinction:

What you see is the front-end. You don't see the back-end. It looks good on the front end. It works because of the back-end. They collaborate. The front-end requests the back-end for assistance when you click a button. The back-end completes the task and responds. The outcome is then displayed on the front end.

3. THE PROCESS FROM TYPING A URL TO VIEWING A WEBPAGE

When you type a URL like "https://www.example.com" into your browser's address bar and press Enter, a complex series of steps happens in just seconds to bring that webpage to your screen.

STEP 1: URL PARSING AND DNS RESOLUTION

First, your browser breaks down the URL to understand what you're asking for. It identifies the protocol (https://), the domain name (www.example.com), and any specific page or resource you want. Since computers communicate using IP addresses rather than domain names, your browser needs to find the IP address for "example.com". It sends a request to a DNS (Domain Name System) server, which acts like a phonebook for the internet. The DNS server responds with the IP address, such as 93.184.216.34.

STEP 2: ESTABLISHING A TCP CONNECTION

Now that your browser knows where to send the request, it establishes a connection with the web server at that IP address. This happens through a "TCP handshake" - a three-step process where your computer and the server exchange messages to agree they're ready to communicate. If you're using HTTPS, an additional TLS/SSL handshake occurs to create an encrypted, secure connection between your browser and the server.

STEP 3: SENDING THE HTTP REQUEST

Once the connection is established, your browser sends an HTTP request to the server. This request includes several components:

HTTP REQUEST EXAMPLE:

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: text/html,application/json
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

The first line contains the HTTP method (GET means you want to retrieve data), the specific resource you're requesting (/index.html), and the HTTP version. The following lines are headers that provide additional information. The "Host" header tells the server which website you want (important because one server might host multiple sites). The

"User-Agent" identifies your browser and operating system. The "Accept" headers tell the server what types of content your browser can handle.

STEP 4: SERVER PROCESSES THE REQUEST

The web server receives your HTTP request and processes it. The server examines what you're asking for and determines how to respond. If you requested a static file like an HTML page, the server retrieves it from storage. If you requested something that requires computation (like search results or a personalized dashboard), the back-end code runs, potentially querying databases to gather the necessary data. The server then prepares an HTTP response.

STEP 5: RECEIVING THE HTTP RESPONSE

The server sends back an HTTP response to your browser. This response includes a status code indicating whether the request was successful, headers with metadata about the response, and the actual content you requested.

HTTP RESPONSE EXAMPLE:

```
HTTP/1.1 200 OK
Date: Mon, 16 Dec 2025 10:30:45 GMT
Server: Apache/2.4.41
Content-Type: text/html; charset=UTF-8
Content-Length: 2548
Cache-Control: max-age=3600
Connection: keep-alive

<!DOCTYPE html>
<html>
<head>
   <title>Welcome to Example.com</title>
   <link rel="stylesheet" href="styles.css">
</head>
<body>
   <h1>Hello World!</h1>
   <p>This is an example webpage.</p>
</body>
```

```
    </html>
```

The first line shows the HTTP version and status code. "200 OK" means the request was successful. The status line is followed by response headers. The "Content-Type" header tells your browser that the response contains HTML. The "Content-Length" indicates the size of the content. After the headers comes a blank line, followed by the actual response body - in this case, the HTML code for the webpage.

STEP 6: BROWSER RENDERS THE PAGE

Your browser receives the HTTP response and begins processing the HTML. It reads through the HTML and starts building the webpage. When it encounters references to other resources (like CSS stylesheets, JavaScript files, or images), it makes additional HTTP requests to fetch those resources. Each of these follows the same request-response cycle. The browser applies CSS styling to make the page look attractive, executes JavaScript to add interactivity, and displays images. Finally, the complete webpage appears on your screen, ready for you to view and interact with.

THE HTTP REQUEST-RESPONSE CYCLE IN SUMMARY:

This entire process is called the HTTP request-response cycle. It's a fundamental pattern of communication on the web. Your browser (the client) makes a request, the server processes it and sends a response, and your browser displays the result. This cycle happens repeatedly - every time you click a link, submit a form, or load a new resource, another request-response cycle occurs. The connection between your browser and the server can remain open (keep-alive) to make subsequent requests faster, avoiding the need to establish a new connection each time.

This seemingly simple action of typing a URL involves DNS lookups, network connections, HTTP communication, server-side processing, and browser rendering - all working together seamlessly to deliver the web experience we use every day.