**Technical Interview Questions**

**1. HTML**

**1.1. General Layout of HTML File**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta name="description" content="For SEO purposes.">

<title>Your Page Title</title>

<link rel="stylesheet" href="styles.css">

<link rel="icon" href="favicon.ico" type="image/x-icon">

<script src="script.js" defer></script>

</head>

<body>

<header>

<h1>Welcome to My Website</h1>

<nav>

<ul>

<li><a href="index.html">Home</a></li>

</ul>

</nav>

</header>

<main>

<section>

<h2>Subheading</h2>

<p>Paragraph</p>

</section>

</main>

<footer></footer>

<!-- Inline JavaScript (optimised performance placed at the end) -->

<script></script>

</body>

</html>

**1.2. Semantic HTML tags?**

* Tags that define the meaning of the content they contain.
* Eg. <header>, <article>, and <footer>
* On the other hand, tags like <div> and <span> are typical examples of non-semantic HTML elements.

**2. CSS**

**2.1. CSS Specificity?**

* Set of rules that browsers use to determine which CSS styles are applied to an element when there are multiple competing styles.
* Goal to make stylesheets predictable and maintainable.
* Can use !important to override specificity but can cause issues with management.

|  |  |  |
| --- | --- | --- |
| **Value** | **Selector** | **Example** |
| a. (1,0,0,0) | Inline styles (inside a style attribute). | <div style="color: red;"></div> |
| b. (0,1,0,0) | IDs. | #header { color: blue; } |
| c. (0,0,1,0) | Classes, attributes, and pseudo-classes (:hover, :focus) | .content { color: green; }  [type="text"] { color: yellow; }  :hover { color: pink; } |
| d. (0,0,0,1) | Elements and pseudo-elements (::before, ::after) | p { color: black; }  ::before { content: ''; } |

**2.2. Box Model?**

* Describes how elements on a web page are structured and how their dimensions are calculated consisting of several components, each contributing to the element's overall size and how it interacts with other elements.
* Content = height, width
* Padding = top, right, bottom, left
* Border = width, style, colour
* Margin = top, right, bottom, left



**3. JavaScript**

**3.1. What is JavaScript?**

* Programming language used to create dynamic content on web pages.
* Interpreted language 🡪 typically executed in the browser directly without the need of prior compilation.
* Scripting language 🡪 often used for scripting, automating tasks within web pages and web applications.
* Event-driven 🡪 code that is responsive to user actions.
* Client-side scripting 🡪 manipulate DOM to change structure and content of webpages dynamically.
* Server-side scripting 🡪 use Node.js to allow for server-side development, allowing developers to use one programming language for both client and server-side code.

**3.2. Higher Order Functions?**

* JavaScript functions that take in or return a function.
* Eg. map(), filter()

**3.3. Arrow Functions?**

* Introduced in ES6 and are more precise way of writing functions.
* Key differences from normal functions:
  + Removal of function keyword.
  + If only consist of a return statement following a single line of code, we can remove the curly braces and remove the return keyword.
  + If function has only one argument, we can remove the parenthesis.
  + Cannot use arrow functions as object constructors.
  + Eg. const Car = (color) => { this.color = color;};

**3.4. Difference between == and ===?**

* === ensures that both the value and type are the same.
* Helps in avoiding unexpected behaviour caused by type coercion.
* Makes the code easier to debug and in general it’s in best practice to use.

**3.5. Difference between null and undefined?**

* null == undefined (true because both are considered loosely equal)
* null === undefined (false because they are of different types)
* null is technically an object with no value and undefined are uninitialized variables and function parameters that are not provided.

**3.6. Array vs. Objects**

* Arrays for when you need to store a collection of values in a specific order and access them by index.
* Objects for when you need to represent structured data with named properties and associated values, or when you need to store data in a non-sequential manner.

**3.7. Functions**

* Reusable block of code that performs a specific task or calculates a value.
* Functions can be called with zero or more arguments, depending on how they are defined and return values.
* Eg. function hello(name) { return “Hi” + name + “!”; }
* ES6: const hello = (name) => { return `Hi ${name}!`; }

**3.8. JSON (JavaScript Object Notation)**

* Data format useful for transmission between client and server as it represents data as JavaScript data structures.

**3.9. DOM Event Bubbling?**

* Default behaviour of the DOM in browsers where the event of an element receives is propagated to its parent and ancestors, upward in the DOM tree until it gets to the root element (html).
* Eg. if you have a click event listener on a button and it’s contained in the body, div, and span, respectively. The event is triggered on the button, span, div and body.
* Can prevent this by using stopPropagation() method 🡪 button.addEventListener(‘click’, (event) => { event. stopPropagation() });

**3.10. Difference between Var, Let, and Const?**

|  |  |  |  |
| --- | --- | --- | --- |
| **Features** | **Var** | **Let** | **Const** |
| Scope | Function-scoped | Block-scoped | Block-scoped |
| Hoisting | Hoisted, initialized to ‘undefined’. | Hoisted but not initialized. | Hoisted but not initialized. |
| Redeclaration | Allowed within the same scope. | Not allowed within the same scope. | Not allowed within the same scope. |
| Mutability | Mutable | Mutable | Immutable binding |

**4. Rendering**

**4.1. Server-Side Rendering (SSR) vs. Client-Side Rendering (CSR).**

* SSR: The server generates the full HTML (+CSS and JS functions) for a web page and sends it to the client's browser. The browser then renders the HTML to display the content to the user.
* CSR: the browser downloads a minimal HTML shell and then uses JavaScript to build and render the content dynamically on the client side.

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Server-Side Rendering (SSR)** | **Client-Side Rendering (CSR)** |
| Initial Load Time | Faster | Slower |
| Subsequent Navigation | Slower (full page reloads) | Faster (partial updates, no full reloads) |
| SEO | Better (pre-rendered HTML) | Challenging (requires additional setup like pre-rendering or server-side rendering for critical pages) |
| Server Load | Higher (renders HTML on server) | Lower (renders HTML on client) |
| Interactivity | Less dynamic (requires full reloads) | More dynamic (single-page application feel) |
| Browser Compatibility | Better (less reliance on JavaScript) | Relies heavily on JavaScript |
| Frameworks | Next.js, Ruby on Rails | React.js, Vue.js, Angular |

**4.2. In what scenarios would you prefer Client-Side Rendering over Server-Side Rendering?**

* Web pages requiring dynamic page generation eg. online quiz.
* Advanced UI Components: Applications that utilize sophisticated UI components (e.g., drag-and-drop, animations, real-time data visualization) are well-suited for CSR since it allows for more dynamic and responsive interfaces.

**5. APIs**

**5.1. REST API.**

* Representational State Transfer is a common API standard used to mediate the interaction between the client and the server. Hence, a RESTful API:

1. Organises resources into a set of unique URIs (Uniform resource identifiers) 🡪 <https://example.com/api/users> , <https://example.com/api/users/123>,
2. Client interacts with the server via a HTTP request 🡪 POST/users HTTP/1
3. CRUD (Create, Read, Update, Delete), in the body of these request there could be an option HTTP body usually in JSON.
4. Server receives and processes the request from the client sending back a response with a status code 🡪 HTTP/1 200 OK
   * + - 200-level: Success
       - 400-level: Something wrong with our request
       - 500-level: Something wrong at the server level
5. Allow for retrying of the request if failed, however it is important to note that some requests are not idempotent.
6. Stateless: client and server shouldn’t store any information about each other, and each request/response is independent from all others.
7. Pagination is used when an API endpoint with a large amount of data is returned 🡪 /products?limit=25&offset=50.
8. Versioning provides backwards compatibility if there are breaking changes 🡪 /v1/products, /v2/products

**5.2. HTTP Request Methods.**

|  |  |  |
| --- | --- | --- |
| **Methods** | **Purpose** | **Characteristics\*** |
| POST | Submit data to the server to create a new resource. | Not Idempotent |
| GET | Retrieve data from the server. | Safe, Idempotent |
| PUT | Update an existing resource or create a new resource if it does not exist. | Idempotent |
| DELETE | Remove a resource. | Idempotent |
| PATCH | Partially update an existing resource. | Not Necessarily Idempotent |
| HEAD | Get the headers of a resource, like GET but without the response body. | Safe, Idempotent |

\*Safe = does not alter the resource

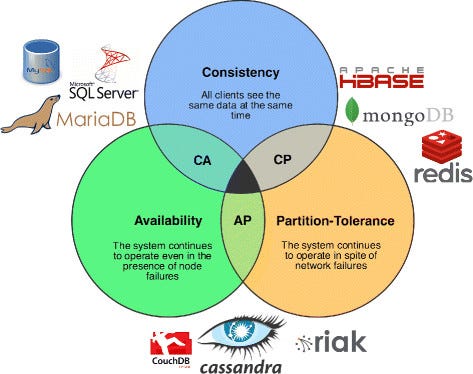
\*Idempotent = same request with the same data will produce the same result

**6. Database**

**6.1. Relational Database (SQL) vs. Non-Relational Database (NoSQL).**

|  |  |  |
| --- | --- | --- |
| **Properties** | **SQL** | **NoSQL** |
| Data Model | Relational (tables with rows and columns). | Various models: Document, Key-Value, Column-Family, Graph |
| Schema | Fixed. | Dynamic |
| Scalability | Vertical scaling (adding more power to a single server). | Horizontal scaling (adding more servers). |
| Transactions | ACID (Atomicity, Consistency, Isolation, Durability). | BASE (Basically Available, Soft state, eventually consistent) |
| Query Language | SQL (Structured Query Language). | Varies by database: MongoDB uses MQL, Cassandra uses CQL |
| Consistency | Strong consistency. | Eventual consistency |
| Development | Requires upfront schema design. | Can start without defining the schema |
| Use Cases | Suitable for structured data, complex queries, transactions. | Suitable for unstructured data, large-scale distributed data. |
| Examples | MySQL, Postgres, Oracle Database, Microsoft SQL Server. | MongoDB, Cassandra, Redis, Couchbase, DynamoDB. |
| Joins | Support complex joins. | Generally, does not support joins. |
| Community & Support | Large, established communities and support. | Rapidly growing communities and support |

**6.2. CAP Theorem.**



**6.3. ACID Properties.**

* **A**tomicity: transaction performed in its entirety or not at all.
* **C**onsistency: takes database from one consistent state to another.
* **I**solation: not interfered with by other transactions.
* **D**urability: changes must persist in the database.

**6.4. Basic SQL Operations.**

**DISTINT** 🡪 eliminates duplicate tuples.

A black text on a white background

Description automatically generated

**AS** 🡪 required if selecting from > 1 table.

**LIKE 🡪 %** replaces an arbitrary number of zero or more, **\_** replaces a single character.

**BETWEEN, AND**

**IS NULL** (each individual NULL value considered to be different from every other NULL).

**DESC**, **ASC**

* SQL uses a three-valued logic: TRUE, FALSE and UNKNOWN.
* **NULL = NULL** comparison is avoided.
* UNION, EXCEPT (difference), INTERSECT.