Q1: What is DNS and its purpose?

DNS stands for Domain Name System, a crucial component of the internet that converts human-friendly domain names like example.com into numerical IP addresses that computers use to identify each other on the network. This process is essential for navigating the web, as it allows users to access websites using easy-to-remember names instead of complex numerical addresses

It operates behind the scenes to ensure that when you type a website's name into your browser, you're directed to the correct server hosting that site. Without DNS, the internet as we know it would be much less user-friendly, requiring users to memorize and enter long strings of numbers to access websites.

```
// Example of DNS lookup using Node.js
const dns = require('dns');

dns.lookup('example.com', (err, address, family) => {
 if (err) throw err;
 console.log('IP address:', address);
 console.log('IP family:', family);
});
```

Q2: Explain HTTP and its main functions.

HTTP, or Hypertext Transfer Protocol, is the foundation of communication on the web. It defines how messages are formatted and transmitted between a client and a server. HTTP methods (like GET, POST, PUT, DELETE) specify the action to be performed. Status codes (like 200 for success, 404 for not found) indicate the result of the request. Headers provide additional information, such as content type and authentication. HTTP is stateless, meaning each request is independent of the previous one, which simplifies communication but requires each request to contain all necessary information.

Q3: What components are included in an HTTP request?

An HTTP request typically includes the following components:

- Protocol version: Specifies the version of HTTP being used (e.g., HTTP/1.1)
- Verb: Indicates the request type (e.g., GET, POST, PUT, DELETE)
- Headers: Additional metadata (e.g., Content-Type, Authorization)
- Body: Optional data sent with the request, often in POST or PUT requests

```
// Example of an HTTP GET request using the Fetch API
fetch('https://api.example.com/data', {
 method: 'GET',
 headers: {
'Content-Type': 'application/json',
'Authorization': 'Bearer token'
}
}).then(response => response.json()).then(data => console.log(data));
```

Q4: What are the components of an HTTP response?

An HTTP response includes several key components:

- Status: A three-digit code indicating the outcome (e.g., 200 for success) along with a status text (e.g., 'OK' for 200)

- Headers: Metadata about the response (e.g., Content-Type, Content-Length)
- Body: The content of the response, such as HTML, JSON, or binary data

Q5: What are Static Site Generators (SSGs)?

Static Site Generators (SSGs) are tools that pre-render web pages into static HTML files based on templates and content, which can then be served directly to users. This approach improves security since the pages are pre-built and don't rely on server-side processing for each request, and it enhances performance because static files can be served quickly from a content delivery network (CDN). However, SSGs can have longer build times for large sites and may lack dynamic content out of the box, making them particularly well-suited for blogs, documentation, and marketing websites where the content doesn't change frequently.

Some key benefits and examples of using SSGs include:

- Improved Performance: SSGs deliver pre-rendered HTML pages, reducing the time needed to load and render content.
- Enhanced Security: By eliminating the need for server-side processing, SSGs reduce the attack surface for malicious exploits.
- Scalability: Static sites can be easily scaled as they are served from a CDN, handling a large number of requests efficiently.
- SEO Optimization: Pre-rendered HTML ensures that search engine crawlers can easily index the content, improving visibility.

Popular SSGs include GatsbyJS, Next.js, Jekyll, and Hugo.

Q6: Describe GraphQL and its use cases.

GraphQL is a powerful query language for APIs that allows clients to request exactly the data they need, making it more efficient and flexible than traditional REST APIs. Unlike REST, where multiple endpoints might be needed to retrieve related data, GraphQL enables fetching all required data in a single request, including nested and related data structures. This reduces the number of API calls and minimizes the amount of data transferred, making applications faster and more efficient.

Some key features and use cases of GraphQL include:

- Efficient Data Retrieval: Fetch only the data you need with a single query, reducing over-fetching or under-fetching of data.
- Real-time Updates: Combine with subscriptions to get real-time data updates, making it ideal for applications requiring live data.
- Better Performance: Reduces the number of network requests by fetching related data in a single query.
- Strongly Typed Schema: Ensures that queries are validated against a schema, reducing errors and improving development experience.

```
# Example of a GraphQL query to fetch user details and their posts
{
 user(id: "1") {
 id
 name
 email
 posts {
 id
 title
 content
}
```

```
}
}
```

```
// Example response from the GraphQL server
{
"data": {
"user": {
"id": "1",
"name": "John Doe",
"email": "john.doe@example.com",
"posts": [
{
"id": "101",
"title": "GraphQL Basics",
"content": "GraphQL is a query language for APIs..."
},
{
"id": "102",
"title": "Advanced GraphQL",
"content": "Let's dive deeper into GraphQL..."
}
]
}
}
}
```

This example demonstrates how GraphQL allows you to fetch nested and related data in one request, which would typically require multiple endpoints in a REST API. The response structure mirrors the query, making it easier to understand and work with.

Q7: What is Emmet, and how does it improve productivity?

Emmet is a plugin for code editors like Visual Studio Code that speeds up HTML and CSS coding with dynamic snippets. It allows developers to write large portions of code quickly with shortcuts and abbreviations.

```
<!-- Example of Emmet abbreviation -->
div.container>ul>li*5
```

Q8: What does the MERN stack consist of?

The MERN stack consists of the following components:

- MongoDB: A NoSQL document database used for storing data in a flexible, JSON-like format.
- Express: A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
- React: A JavaScript library for building user interfaces, particularly single-page applications.
- Node.js: A runtime environment that allows you to run JavaScript on the server, enabling full-stack JavaScript development.

Q9: What is JAMstack, and why is it popular?

JAMstack stands for JavaScript, APIs, and Markup. It's a modern web development architecture focused on fast, secure, and scalable web applications. The front end is typically built using static site generators, and the backend is handled via APIs, reducing the need for traditional server-based architectures.

Q10: Which tools are commonly used for debugging?

Common tools for debugging include browser developer tools (like Chrome DevTools), Postman for API testing, and ESLint for identifying and fixing JavaScript errors.

```
// Example of using console.log for debugging
console.log('Debugging output:', myVariable);
```

Q11: Define an API and its purpose.

An API, or Application Programming Interface, is a set of rules and protocols that allow different software applications to communicate with each other. It defines the methods and data structures for interacting with the software components, enabling integration and functionality sharing.

```
// Example of an API request using Fetch API
fetch('https://api.example.com/data')
.then(response => response.json())
.then(data => console.log(data));
```

Q12: What is a REST API and how does it work?

A REST API is an API that adheres to the principles of REST (Representational State Transfer). It uses standard HTTP methods like GET, POST, PUT, DELETE and is stateless, meaning each request from the client contains all the information needed for the server to fulfill it.

```
// Example of a REST API GET request using Fetch API
fetch('https://api.example.com/items', {
 method: 'GET',
 headers: {
'Content-Type': 'application/json'
}
}).then(response => response.json()).then(data => console.log(data));
```

Q13: What does CRUD stand for, and why is it important?

CRUD stands for Create, Read, Update, Delete. These are the four basic operations for interacting with persistent storage, such as databases. CRUD operations are fundamental to building APIs and applications that manage data.

```
// Example of CRUD operations using Fetch API
// Create a new item
fetch('https://api.example.com/items', {
 method: 'POST',
 headers: {
'Content-Type': 'application/json'
```

```
},
 body: JSON.stringify({ name: 'New Item' })
}).then(response => response.json()).then(data => console.log(data));

// Read an item
fetch('https://api.example.com/items/1').then(response => response.json()).then(data => console.log(data));

// Update an item
fetch('https://api.example.com/items/1', {
 method: 'PUT',
 headers: {
'Content-Type': 'application/json'
},
 body: JSON.stringify({ name: 'Updated Item' })
}).then(response => response.json()).then(data => console.log(data));

// Delete an item
fetch('https://api.example.com/items/1', {
 method: 'DELETE'
}).then(response => response.json()).then(data => console.log(data));
```

Q14: What is REST, and why is it widely used?

REST stands for Representational State Transfer. It is a standard for developing web services that builds upon HTTP's existing infrastructure. RESTful services are stateless and follow a client-server architecture, making them simple, scalable, and easy to implement.

```
// Example of a RESTful API call using Fetch API
fetch('https://api.example.com/resources')
.then(response => response.json())
.then(data => console.log(data));
```

Q15: What is CORS and how does it work?

CORS, or Cross-Origin Resource Sharing, is a security feature implemented in web browsers to control how resources are shared across different origins. It uses HTTP headers such as `Access-Control-Allow-Origin` to specify which domains are allowed to access resources on a server, `Access-Control-Allow-Methods` to specify which HTTP methods are permitted, and `Access-Control-Allow-Headers` to specify which headers can be used in the actual request.

```
// Example of setting CORS headers in an Express server
app.use((req, res, next) => {
 res.header('Access-Control-Allow-Origin', '*');
 next();
});
```

Q16: How does HTTPS differ from HTTP?

HTTPS is the secure version of HTTP. It encrypts data transmitted between the client and server using SSL/TLS protocols, providing confidentiality, integrity, and authenticity, which is crucial for secure communication over the internet.

Q17: Describe the process of loading a webpage after entering a URL.

The process of loading a webpage involves several steps:

- The user types the URL in the browser.
- The browser sends a DNS query to resolve the domain name to an IP address.
- A TCP connection is established with the server at the resolved IP address.
- The browser sends an HTTP request to the server.
- The server processes the request and sends an HTTP response.
- The browser parses the HTML content and renders the webpage.

Q18: What are the pros and cons of single page applications (SPAs)?

Single page applications (SPAs) offer several advantages:

- Smooth user experience without page reloads.
- Faster subsequent page loads due to loading all necessary resources initially.

However, SPAs also have some drawbacks:

- SEO challenges due to dynamic content.
- Longer initial load times if the JavaScript bundle is large.

Q19: What is RxJS, and why is it useful?

RxJS is a library for reactive programming using observable sequences. It simplifies handling asynchronous operations like events, promises, and data streams, making it ideal for complex data flow management.

```
import { fromEvent } from 'rxjs';
import { map } from 'rxjs/operators';

// Example of using RxJS to handle button clicks
const button = document.querySelector('button');
const clicks$ = fromEvent(button, 'click').pipe(map(() => 'Button clicked!'));
clicks$.subscribe(console.log);
```

Q20: What is a service worker, and what are its uses?

A service worker is a script that runs in the background of a browser, enabling offline capabilities, background syncing, and push notifications. It acts as a proxy between the web application and the network, allowing control over how network requests are handled. The service worker lifecycle includes installation and activation phases, and it's important to note that service workers do not have direct access to the DOM.

```
// Example of registering a service worker
if ('serviceWorker' in navigator) {
 navigator.serviceWorker.register('/service-worker.js')
.then(registration => console.log('Service Worker registered:', registration))
.catch(error => console.error('Service Worker registration failed:', error));
}
```

Q21: Why are bundlers like Webpack used in web development?

Bundlers like Webpack are used to manage, transform, and optimize assets in a web application. They compile multiple modules, including JavaScript, CSS, and images, into a single or small set of files, reducing HTTP requests and improving load performance.

```
// Example of a basic Webpack configuration
module.exports = {
 entry: './src/index.js',
 output: {
 filename: 'bundle.js',
 path: __dirname + '/dist',
},
 module: {
 rules: [
{ test: /.js$/, use: 'babel-loader' },
{ test: /.css$/, use: ['style-loader', 'css-loader'] },
],
},
};
```

Q22: How does HTTP2 improve upon HTTP1?

HTTP2 introduces several enhancements over HTTP1, including multiplexing (handling multiple requests simultaneously over a single connection), server push (sending resources before they are requested), and header compression, which reduces latency and improves performance.

Q23: What is Docker, and how does it benefit development?

Docker is a platform that enables OS-level virtualization to package and distribute software in containers. Containers encapsulate an application and its dependencies, ensuring consistency across development, testing, and production environments.

```
# Example of a simple Dockerfile
FROM node:14

# Set the working directory
WORKDIR /app

# Copy package.json and install dependencies
COPY package*.json ./
RUN npm install

# Copy the rest of the application code
COPY . .

# Expose the application port
EXPOSE 3000
```

```
# Start the application
CMD ["node", "index.js"]
```

Q24: What is a CDN, and how does it improve content delivery?

A CDN (Content Delivery Network) is a globally distributed network of servers that delivers content to users based on their geographic location. By caching content closer to users, CDNs reduce latency, improve load times, and enhance security.

```
// Example of setting up a CDN with Cloudflare in a web application
const script = document.createElement('script');
script.src = 'https://cdn.example.com/myfile.js';
document.head.appendChild(script);
```

Q25: What is JWT, and how is it used in authentication?

JWT, or JSON Web Token, is a compact, URL-safe token used for securely transmitting information between parties. It is commonly used in authentication processes to verify user identity by encoding claims in a JSON object that is signed using a secret or public/private key pair.

```
// Example of creating and verifying a JWT using jsonwebtoken in Node.js

const jwt = require('jsonwebtoken');

// Secret key for signing the JWT
const secretKey = 'mySecretKey';

// Creating a JWT
const token = jwt.sign({ userId: 123, role: 'admin' }, secretKey, { expiresIn: '1h' });
console.log('Generated Token:', token);

// Verifying a JWT
jwt.verify(token, secretKey, (err, decoded) => {
 if (err) {
 console.error('Invalid token:', err);
} else {
 console.log('Decoded Payload:', decoded);
// Proceed with authenticated user's information
}
});

// Example output:
// Generated Token:
// eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
// .eyJ1c2VySWQiOjEyMywicm9sZSI6ImFkbWluIiwiaWF0IjoxNjE5Njk2NTQ2LCJleHAiOjE2MTk3MDAxNDZ9
```

```
// .6YF7BQ8e4uylItXU3jN6gI3OeB0CNpm9nViPq-Kb7Jw
// Decoded Payload: { userId: 123, role: 'admin', iat: 1619696546, exp: 1619700146 }
```

Q26: What is serverless architecture?

Serverless is a cloud-native development model where developers write and deploy code without managing servers. The cloud provider automatically allocates and scales resources as needed. AWS Lambda is a popular example of a serverless service.

```
// Example of an AWS Lambda function with API Gateway

// This function handles HTTP requests and returns a JSON response
exports.handler = async (event) => {
 const { httpMethod, path } = event;

 if (httpMethod === 'GET' && path === '/hello') {
 return {
 statusCode: 200,
 headers: { 'Content-Type': 'application/json' },
 body: JSON.stringify({ message: 'Hello, world!' }),
};
}

 return {
 statusCode: 404,
 headers: { 'Content-Type': 'application/json' },
 body: JSON.stringify({ error: 'Not Found' }),
};
};

// Sample AWS Lambda configuration for API Gateway
// This configuration maps HTTP GET requests to the Lambda function
// {
//   "httpMethod": "GET",
//   "path": "/hello",
//   "body": null,
//   "headers": { ... },
//   "queryStringParameters": { ... }
// }
```

Q27: What is FOUC, and how can it be prevented?

FOUC, or Flash of Unstyled Content, occurs when a web page is displayed with default browser styles before the external CSS has fully loaded. This can lead to a poor user experience as the layout may shift during loading.

```
<!-- Example of preventing FOUC using inline critical CSS -->
```

```
<!-- Inline critical CSS in the <head> of your HTML -->
<style>
/* Critical CSS to be loaded immediately */
 body {
 font-family: Arial, sans-serif;
 background-color: #f4f4f4;
 margin: 0;
 padding: 0;
}

.header {
 background-color: #333;
 color: white;
 text-align: center;
 padding: 1rem;
}

/* Hide content until fully styled */
.content {
 visibility: hidden;
}
</style>

<!-- Load external CSS after critical CSS -->
<link rel="stylesheet" href="styles.css" onload="document.querySelector('.content').style.visibility = 'visible';">

<div class="header">
<h1>My Website</h1>
</div>
<div class="content">
<p>This content will be visible once the CSS is fully loaded.</p>
</div>
```

Q28: What is TCP/IP, and what are its key layers?

TCP/IP (Transmission Control Protocol/Internet Protocol) is the suite of communication protocols used to connect network devices on the internet. It organizes the communication process into four layers, each with specific functions.

- Application Layer: Protocols like HTTP, FTP.
- Transport Layer: Host-to-host communication, e.g., TCP.
- Network Layer: Routing of packets, e.g., IP.
- Physical Layer: Hardware-level communication.

Q29: What is 'The Cloud', and what are its benefits?

'The Cloud' refers to servers accessed over the internet, along with the software and databases that run on those servers. Cloud services, provided by companies like AWS, Google Cloud, and Microsoft Azure, offer scalability, flexibility, and cost efficiency, allowing businesses to operate without maintaining physical servers.

```
# Example of deploying a service on AWS
$ aws deploy create-deployment --application-name my-app --s3-location bucket=my-bucket,key=my-app.zip
```