Topics and Questions from the Document on Web Technologies:

1. HTTP and Protocols

- **What is HTTP?** Explain the structure of HTTP request message.
- Differentiate between **HTTP** and **HTTPS**.
- Describe the structure of **HTTP request and response messages** with an example.
- **What are HTTP request methods?** List the four types.

---

2. HTML

- **Form Creation:**
  - Create forms with various input elements like text boxes, multiline fields, checkboxes, radio buttons, and buttons (Submit, Reset).
  - Example:
    - Placement form: Collect name, SRN, college details, department (CSE, ECE, MECH), semester options (IV, V, VI).
    - Create a shopping form with fields for items like wireless mouse, USB, payment options (Net Banking, Google Pay, COD).
  - Validation: Add constraints (e.g., 10-digit numbers).
- **Table Creation:**
  - Create tables with borders and captions like "Student Details".
  - Write HTML code for creating specified table layouts.
- **Other Elements:**
  - Create hyperlinks for images.
  - Use new HTML5 elements like `<audio>`, `<video>`, and `<progress>`.

---

3. CSS

- **Ways to Include CSS:**
  - Inline, Internal, and External CSS.

    - Accept or reject statements like: "The only way to apply CSS is by using external stylesheets."
- **CSS Box Model:**
    - Explain the **CSS Box Model** with a neat diagram and its significance.
- **CSS Rules:**
    - Change text color on mouse hover.
    - Create embedded, inline, and class-based styles.

---

4. JavaScript and DOM Manipulation

- **Basic Concepts:**
    - Explain event bubbling in the DOM.
    - Write examples of JavaScript built-in objects.
    - Differences between `var`, `let`, and `const`.
- **DOM Manipulation:**
    - Methods to access DOM elements (e.g., `getElementById`, `querySelector`).
    - Add rows to tables dynamically using JavaScript.
    - Handle mouseover events (change color based on number being odd/even).
- **Event Handling:**
    - Prevent default behavior using `event.preventDefault`.
    - Stop propagation using `event.stopPropagation`.
- **Hoisting:**
    - Explain the concept of hoisting and how to avoid it.

---

5. jQuery

- **Selectors and Effects:**
    - Perform tasks like:
        - Change paragraph color.
        - Increase font size on hover.
        - Fade in and out elements with animations.
- **Practical Code Tasks:**
    - Hide or show text when clicking buttons.

---

6. React

- **Component Lifecycle:**
  - Explain lifecycle methods like `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` with diagrams.
- **Stateful vs Stateless Components:**
  - Differences between stateful and stateless components.
- **Practical Tasks:**
  - Create class components rendering styled elements like `<h1>` and `<p>`.
  - Render an unordered list using React's `map()` method.
- **Props and Keys:**
  - Explain the significance of the `key` property.
  - Discuss how props are passed to components.

---

7. AJAX

- Define **AJAX** and explain key XHR object properties.
- Write synchronous and asynchronous code examples.

---

8. JSON and XML

- Differences between **JSON** and **XML**.
- Represent student details (e.g., ID, name, branch, CGPA) in both JSON and XML formats.

---

9. Node.js

- **Basics of Node.js:**
  - Features of Node.js.
  - Explanation of buffers with examples.
- **File System Module:**
  - Read and write files using `fs` module.
  - Handle errors appropriately.
- **Server Creation:**
  - Create a basic HTTP server that serves HTML files.

    - Handle requests with custom URLs and error messages.

---

10. Express.js

- **Middleware:**
    - Types of middleware and their roles in Express.js applications.
    - Example: Router-level middleware.
- **Routing:**
    - Dynamic routes using colons in URL paths (e.g., `/hello/:id`).
    - Route example:
        - `/pes` → "Hello PES"
        - `/pes/:CSE` → "Branch name is CSE"
        - `/pes/:CSE/:123` → "ID: 123 and name: CSE"
- **File Uploads:**
    - Handle multiple file uploads with error management.
- **Error Handling:**
    - Manage errors effectively in Express.js.

---

11. MongoDB

- **Queries:**
    - List all documents from a collection.
    - Filter documents based on conditions (e.g., `code = "UE20CS204"`).
- **Database Connectivity:**
    - Connect to a MongoDB database using Node.js.
    - Insert multiple documents into collections like "Fruits" or "Students."

---

12. RESTful APIs

- **Definition and Constraints:**
    - Explain REST API and its design principles (statelessness, uniform interface).
- **API Implementation:**
    - Write server-side scripts for handling GET and POST requests.

    - Example: Manage flight details in MongoDB with routes like:
        - `/flights` (GET): Return all flight details.
        - `/flights/:from/:to` (GET): Return specific flights.
        - `/flights` (POST): Add flight details.

---

13. Miscellaneous

- Explain **web workers** and methods/events used for communication.
- **Pug Template**:
    - Use Pug to create templates for forms like a library card.

Covered Topics and Subtopics:

1. **HTTP and Protocols**
    - Definition and structure of request/response.
    - Comparison of HTTP and HTTPS.
    - HTTP methods and their role in RESTful APIs.
2. **HTML**
    - Form creation (various examples like placement forms, shopping carts, and registration forms).
    - Table design for structured data (e.g., student details).
    - Special tags like `<audio>`, `<video>`, hyperlinks.
3. **CSS**
    - Box Model explanation with diagrams.
    - Inclusion methods (inline, internal, external CSS).
    - Styling tasks such as hover effects and dynamic styling rules.
4. **JavaScript and DOM Manipulation**
    - Core concepts: hoisting, event handling, string manipulation.
    - DOM manipulation: dynamically adding rows, changing styles, handling events.
    - Async tasks: Fetch API for loading audio/video.
5. **jQuery**
    - Handling animations (e.g., sliding divs).
    - Dynamically changing styles or elements on the page.
6. **React**
    - Component lifecycle (methods like `componentDidMount` and `componentDidUpdate`).
    - Controlled and uncontrolled components.

- Props and key properties for rendering lists dynamically.

7. **Node.js**
    - Core concepts: single-threaded architecture, event emitters, buffers.
    - File handling (reading/writing files asynchronously).
    - Server creation to handle GET/POST requests.
8. **Express.js**
    - Middleware functions and their types.
    - Handling file uploads and creating dynamic routes.
9. **MongoDB**
    - Server-side interaction with MongoDB (CRUD operations).
    - Example queries for collections like "students" and "faculty".
10. **RESTful APIs**
    - Design constraints of REST APIs.
    - Writing API routes for dynamic data handling (e.g., flight details, restaurant ratings).
11. **Miscellaneous Topics**
    - Chess game object-oriented design (extending prototypes).
    - JavaScript mini-projects (e.g., guessing games, Minecraft inventory management).
    - Handling cookies and session data.

---

1. Chess Game Object-Oriented Design (Extending Prototypes)

Problem:

- Design a `ChessPiece` class with properties like name, color, and position.
- Extend it to create a `King` class with an additional property, `castled`, and a method `setCastled`.

Code:

```
// Base ChessPiece Class
function ChessPiece(name, color, position) {
    this.name = name;
    this.color = color;
    this.position = position;
}

ChessPiece.prototype.display = function() {
    console.log(`${this.name} (${this.color}) is at position
${this.position}`);
};

ChessPiece.prototype.moveTo = function(newPosition) {
    console.log(`Moving ${this.name} to ${newPosition}`);
    this.position = newPosition;
};

// Extended King Class
function King(name, color, position, castled = false) {
    ChessPiece.call(this, name, color, position);
    this.castled = castled;
}

// Inherit from ChessPiece prototype
King.prototype = Object.create(ChessPiece.prototype);
King.prototype.constructor = King;

King.prototype.setCastled = function() {
    if (this.position === "G8" || this.position === "C8") {
        this.castled = true;
    }
    console.log(`Castled status: ${this.castled}`);
};

// Usage
let king = new King("King", "Black", "E8");
king.display();
king.moveTo("G8");
king.setCastled();
king.display();
```

---

2. JavaScript Mini-Projects

a. Guessing Game

Problem:

- Generate a random date between `01-01-2021` and `31-12-2021`.
- Let the user guess the date. Compare and provide feedback.

Code:

```
// Generate Random Date
function getRandomDate() {
    let year = 2021;
    let month = Math.floor(Math.random() * 12);
    let day = Math.floor(Math.random() * 28) + 1;
    return new Date(year, month, day);
}

// Main Game Function
function playGuessingGame() {
    const randomDate = getRandomDate();
    console.log(`Random date (for testing): ${randomDate}`);

    const userGuess = prompt("Enter a date (MM/DD/YYYY):");
    const guessedDate = new Date(userGuess);

    if (guessedDate.getTime() === randomDate.getTime()) {
        alert("You guessed it right!");
    } else if (guessedDate > randomDate) {
        alert("Your guess is greater than the random date.");
    } else {
        alert("Your guess is less than the random date.");
    }
}

// Start Game
playGuessingGame();
```

b. Minecraft Inventory Management

Problem:

- Build an axe using 2 sticks and 3 stones. Calculate how many axes can be crafted.

Code:

```
class MinecraftInventory {
    constructor(sticks, stones) {
        this.sticks = sticks;
        this.stones = stones;
    }

    calculateAxes() {
        let stickSets = Math.floor(this.sticks / 2);
        let stoneSets = Math.floor(this.stones / 3);
        return Math.min(stickSets, stoneSets);
    }

    displayAxeCount() {
        const axeCount = this.calculateAxes();
        console.log(`You can craft ${axeCount} axes.`);
    }
}

// Example Usage
let inventory = new MinecraftInventory(10, 9);
inventory.displayAxeCount();
```

---

3. Handling Cookies and Session Data

Problem:

- Implement a middleware function to handle cookies.
- Increment a "visit count" stored in a cookie.

Code:

```
const express = require('express');
const cookieParser = require('cookie-parser');

const app = express();

// Use cookie-parser middleware
app.use(cookieParser());

// Middleware to handle visits
app.use((req, res, next) => {
    let visitCount = parseInt(req.cookies.visitCount) || 0;
    visitCount++;
    res.cookie('visitCount', visitCount, { maxAge: 900000, httpOnly: true
});
    console.log(`Visit count: ${visitCount}`);
    res.send(`You have visited this site ${visitCount} times.`);
});

// Start Server
app.listen(3000, () => {
    console.log('Server running on http://localhost:3000');
});
```

Based on the **past year questions (PYQ)** from your uploaded files, here are the likely **repeated or similar questions** you might encounter in future assessments for Web Technologies:

---

1. Core Web Development Concepts

- **HTTP:**
  - Explain the structure of an HTTP request and response message.
  - Differentiate between HTTP and HTTPS.
  - List and explain HTTP methods (e.g., GET, POST, PUT, DELETE).
- **REST APIs:**
  - Define RESTful API and explain its design constraints.
  - Map HTTP methods (CRUD operations) to REST APIs.
  - Write code for basic routes using Express.js for handling GET and POST requests.
- **HTML:**
  - Create forms with:
    - Input fields (text, checkbox, radio buttons).
    - Validation rules (e.g., mandatory 10-digit number field).
    - Buttons (Submit, Reset).
  - Create tables for structured data (e.g., Student Details, Shopping Cart).
  - Use of multimedia elements: `<audio>` and `<video>` tags.
- **CSS:**
  - Explain the CSS Box Model with a neat diagram.
  - Demonstrate various ways of applying CSS (inline, internal, external).
  - Style elements dynamically using CSS rules (hover effects, font changes).

---

2. JavaScript Concepts

- **Event Handling:**
  - Handle events like `onclick`, `onmouseover`, `onmouseout`.

  - Difference between `preventDefault()` and `stopPropagation()`.
- **DOM Manipulation:**
  - Dynamically add rows to a table and style them based on conditions.
  - Write JavaScript to:
    - Change element styles based on conditions.
    - Display specific data in a `div` when a button is clicked.
- **Vanilla JS Mini-Projects:**
  - Implement a **guessing game** (e.g., guessing random dates or numbers).
  - Dynamically generate and style elements like tables and forms.
- **Asynchronous JavaScript:**
  - Use `fetch` API to load resources like JSON, audio, or images dynamically.
  - Explain `XMLHttpRequest` (XHR) object properties (e.g., `responseText`, `readyState`).

---

3. jQuery

- Use jQuery to:
  - Slide elements up or down.
  - Fade elements in or out dynamically.
  - Change styles dynamically (e.g., border color, font size).

---

4. React

- **Lifecycle Methods:**
  - Explain `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`.
- **Components:**
  - Create controlled and uncontrolled components for forms.
  - Use `props` to pass data to components.
  - Render lists dynamically using `map()` with unique `key` properties.
- **Interactive Components:**
  - Create components to:
    - Display dynamically computed values (e.g., total price, Minecraft axes).
    - Apply styles dynamically based on state.

---

5. Node.js

- **Core Concepts:**
  - Is Node.js single-threaded? Explain its event-driven architecture.
  - Explain buffers in Node.js with examples.
  - Read and write files asynchronously using Node.js.
- **Server Creation:**
  - Create a basic server that:
    - Serves HTML files dynamically based on URL.
    - Handles errors (e.g., File Not Found).

---

6. Express.js

- **Middleware:**
  - Explain middleware types (application-level, router-level, error-handling).
  - Write middleware to:
    - Parse cookies and increment a visit counter.
    - Handle errors during file uploads.
- **Routes:**
  - Write routes for GET and POST requests.
  - Handle dynamic URL parameters using colons (e.g., `/users/:id`).
- **File Uploads:**
  - Handle multiple file uploads using the `express-fileupload` library.

---

7. MongoDB

- **CRUD Operations:**
  - Write queries for:
    - Fetching all documents.
    - Filtering documents based on conditions (e.g., `department = "CSE"`).
    - Inserting new documents dynamically.
  - Example:
    - Connect to a MongoDB instance.
    - Insert records into collections like "students" or "flights".

---

8. Miscellaneous

- **Object-Oriented JavaScript:**
  - Extend prototypes (e.g., Chess game implementation with `ChessPiece` and `King`).
  - Create reusable classes with additional properties and methods.
- **Mini Projects:**
  - Minecraft inventory management.
  - Implementing interactive forms with JavaScript or React.
- **Session Handling:**
  - Increment visit counts using cookies or session data.

Most Likely Repeat Questions

1. **HTTP request/response structure** – almost always asked.
2. **Form creation with HTML** – variations of this are frequent.
3. **CSS Box Model explanation** – appears consistently in different papers.
4. **React lifecycle methods** – common React question.
5. **CRUD operations in MongoDB** – frequent in server-side coding questions.
6. **File handling in Node.js** – reading/writing files asynchronously is a common task.
7. **File uploads in Express.js** – popular server-side implementation question.
