10)Create a HTML page with Audio file?

Creating an HTML page with an audio file is relatively straightforward. Below is a simple example to help you get started.

### Step-by-Step Guide

1. **Create the HTML File:**
   - Open a text editor (e.g., Notepad, Sublime Text, VSCode).
   - Save the file with an `.html` extension, e.g., `index.html`.

2. **Add the Basic HTML Structure:**
   - Include the necessary HTML tags to structure your webpage.

3. **Embed the Audio File:**
   - Use the `<audio>` tag to embed the audio file in your HTML page.

Here is an example of how the HTML file should look:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Audio Example</title>
</head>
<body>
```

```
    <h1>Audio File Example</h1>
    <audio controls>
        <source src="path_to_your_audio_file.mp3" type="audio/mpeg">
        Your browser does not support the audio element.
    </audio>
</body>
</html>
```

### Explanation:

- **DOCTYPE Declaration:** `<!DOCTYPE html>` defines the document type and version of HTML.
- **HTML Tag:** `<html lang="en">` begins the HTML document and sets the language to English.
- **Head Section:**
  - `<meta charset="UTF-8">` specifies the character encoding for the HTML document.
  - `<meta name="viewport" content="width=device-width, initial-scale=1.0">` ensures proper rendering on different devices.
  - `<title>Audio Example</title>` sets the title of the webpage.
- **Body Section:**
  - `<h1>Audio File Example</h1>` is a header for the page.
  - `<audio controls>` is the audio element with controls enabled (play, pause, volume).
  - `<source src="path_to_your_audio_file.mp3" type="audio/mpeg">` specifies the path to the audio file and its MIME type.
  - "Your browser does not support the audio element." provides fallback text if the browser does not support the audio tag.

### Steps to View the HTML Page:

1. **Save the File:**

   - Make sure the HTML file and the audio file are in the same directory, or adjust the path accordingly if they are in different locations.

2. **Open the File in a Browser:**

   - Open the saved `.html` file in a web browser (e.g., Chrome, Firefox, Edge).

This will display a webpage with an embedded audio player that you can use to play your audio file.

9)Create a HTML page with Video file ?

Creating an HTML page that includes a video file is straightforward. Below is an example of a basic HTML page that embeds a video using the `<video>` tag. This example assumes you have a video file named `sample-video.mp4` in the same directory as your HTML file.

```html
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Video Example</title>

</head>

<body>

  <h1>Sample Video</h1>

  <video width="640" height="360" controls>

    <source src="sample-video.mp4" type="video/mp4">

    Your browser does not support the video tag.

  </video>

</body>

</html>
```

### Explanation:

- `<!DOCTYPE html>`: Declares the document type and version of HTML.

- `<html lang="en">`: Sets the language of the document to English.

- `<head>`: Contains meta-information about the document.

  - `<meta charset="UTF-8">`: Specifies the character encoding for the HTML document.
  - `<meta name="viewport" content="width=device-width, initial-scale=1.0">`: Ensures the page is responsive by setting the viewport to the device's width.
  - `<title>Video Example</title>`: Sets the title of the web page.
- `<body>`: Contains the content of the HTML document.
  - `<h1>Sample Video</h1>`: A heading for the video section.
  - `<video width="640" height="360" controls>`: The video element with specified width and height, and controls enabled (play, pause, volume).
    - `<source src="sample-video.mp4" type="video/mp4">`: Specifies the source of the video file and its type.
    - `Your browser does not support the video tag.`: Fallback text displayed if the browser does not support the `<video>` element.

To see the video on the webpage, ensure that the video file (`sample-video.mp4`) is in the same directory as your HTML file. Open the HTML file in a web browser to view the embedded video.

8)Create a HTML page with Google Map?

Creating an HTML page with an embedded Google Map involves using the Google Maps JavaScript API. Below is a simple example of an HTML page that embeds a Google Map centered on a specific location.

First, you need to obtain a Google Maps API key by following these steps:

1. Go to the [Google Cloud Console](https://console.cloud.google.com/).
2. Create a new project or select an existing project.
3. Navigate to the "APIs & Services" section.
4. Enable the "Maps JavaScript API".
5. Create credentials to get your API key.

Once you have your API key, you can use it in your HTML file as shown below:

```html
<!DOCTYPE html>
<html>
<head>
  <title>Google Map Example</title>
  <style>
    /* Set the size of the div element that contains the map */
    #map {
      height: 100%;
    }
    /* The height of the HTML and body elements needs to be set to 100% */
    html, body {
      height: 100%;
      margin: 0;
      padding: 0;
    }
  </style>
</head>
<body>
  <div id="map"></div>
  <script>
    // Initialize and add the map
    function initMap() {
      // The location to center the map (e.g., New York City)
      const location = { lat: 40.7128, lng: -74.0060 };
      // The map, centered at the specified location
      const map = new google.maps.Map(document.getElementById("map"), {
        zoom: 10,
        center: location,
      });
      // The marker, positioned at the specified location
      const marker = new google.maps.Marker({
        position: location,
        map: map,
      });
    }
  </script>
  <!-- Load the Google Maps JavaScript API with your API key -->
  <script async defer
src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&callback=initMap
"></script>
</body>
</html>
```

```

Replace `YOUR_API_KEY` with the actual API key you obtained from the Google Cloud Console.

### Explanation:

1. **HTML Structure**:

   - The `head` section contains the page title and a `style` block to set the height of the map container.

   - The `body` section includes a `div` element with an ID of `map` which will contain the map.

2. **CSS**:

   - The `#map` element is styled to take up 100% of the height of its container.

   - Both `html` and `body` elements are set to 100% height to ensure the map fills the viewport.

3. **JavaScript**:

   - A function `initMap` initializes the map. It creates a map centered at a specific location (New York City in this example) and adds a marker at that location.

   - The Google Maps JavaScript API script is loaded asynchronously with the callback set to `initMap` to ensure the map initializes correctly once the API is loaded.

Save this code in an HTML file and open it in a web browser to see the map.
```

11)Create a HTML page to upload a file?

Sure, here's a simple HTML page with a file upload feature:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>File Upload</title>
</head>
<body>
  <h2>Upload a File</h2>
  <form action="upload.php" method="post" enctype="multipart/form-data">
    <input type="file" name="fileToUpload" id="fileToUpload">
    <input type="submit" value="Upload" name="submit">
  </form>
</body>
</html>
```

This HTML page includes a form with a file input field. When the form is submitted, it sends the file to a PHP script named "upload.php" using the POST method with multipart/form-data encoding. Make sure to create the "upload.php" file to handle the file upload on the server side.

7)List all Users,Update user and Delete user(Popup for confirmation eg:Are you sure do you want to delete)

Here's a simple example of how you can achieve listing all users, updating a user, and deleting a user with a confirmation popup using HTML, CSS, and JavaScript:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>User Management</title>
  <style>
    /* Styles for User List */
    .user-list {
      list-style-type: none;
      padding: 0;
      margin: 0;
    }
    .user-item {
      border: 1px solid #ccc;
      border-radius: 5px;
      margin-bottom: 10px;
      padding: 10px;
    }
    .user-item button {
      margin-left: 10px;
    }
    /* Styles for Confirmation Popup */

    .popup {

      display: none;

      position: fixed;

      top: 50%;

      left: 50%;

      transform: translate(-50%, -50%);

      background-color: #fff;

      padding: 20px;

      border: 1px solid #ccc;

      border-radius: 5px;

      box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

    }

    .popup-overlay {

      position: fixed;

      top: 0;

      left: 0;

      width: 100%;

      height: 100%;

      background-color: rgba(0, 0, 0, 0.5);

    }

  </style>

</head>

<body>


  <h2>User Management</h2>
  <!-- User List -->
  <ul class="user-list">
    <li class="user-item" data-id="1">
      User 1 <button class="edit-btn">Edit</button> <button class="delete-btn">Delete</button>
    </li>
    <li class="user-item" data-id="2">
      User 2 <button class="edit-btn">Edit</button> <button class="delete-btn">Delete</button>
    </li>
    <!-- Add more users dynamically or through backend -->
  </ul>

  <!-- Confirmation Popup -->
  <div class="popup" id="confirmationPopup">
    <p>Are you sure you want to delete this user?</p>
    <button id="confirmDelete">Yes, Delete</button>
    <button id="cancelDelete">Cancel</button>
  </div>
  <div class="popup-overlay" id="overlay"></div>

  <script>
    // Get all delete buttons
    const deleteButtons = document.querySelectorAll('.delete-btn');

    // Confirmation Popup
    const confirmationPopup = document.getElementById('confirmationPopup');
    const overlay = document.getElementById('overlay');
    const confirmDeleteBtn = document.getElementById('confirmDelete');
```

```
});

const cancelDeleteBtn = document.getElementById('cancelDelete');

// Function to show confirmation popup
function showConfirmationPopup() {
  confirmationPopup.style.display = 'block';
  overlay.style.display = 'block';
}

// Function to hide confirmation popup
function hideConfirmationPopup() {
  confirmationPopup.style.display = 'none';
  overlay.style.display = 'none';
}

// Attach event listeners to all delete buttons
deleteButtons.forEach(button => {
  button.addEventListener('click', () => {
    showConfirmationPopup();
  });
});

// Event listener for confirm delete button
confirmDeleteBtn.addEventListener('click', () => {
  // Delete user logic goes here
  console.log('User deleted');
  hideConfirmationPopup();
});
```

```
    // Event listener for cancel delete button
    cancelDeleteBtn.addEventListener('click', () => {
      hideConfirmationPopup();
    });
  </script>
</body>
</html>
```

This code snippet provides a user management interface where users can be listed, updated, and deleted. The delete operation prompts a confirmation popup before proceeding with the deletion.

```
}
```

6)Create a User Creation Screen by using all elements in it(like List, Radio button, Drop down, CheckBox)

Sure, here's a simple example of a user creation screen using HTML and CSS:

```html
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>User Creation Screen</title>
 <style>
  body {
   font-family: Arial, sans-serif;
  }
  .container {
   max-width: 400px;
   margin: 0 auto;
   padding: 20px;
   border: 1px solid #ccc;
   border-radius: 5px;
   box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
  }
  .form-group {
   margin-bottom: 20px;
  }
```

```
label {
  display: block;
  font-weight: bold;
  margin-bottom: 5px;
}
input[type="text"], select {
  width: 100%;
  padding: 8px;
  border: 1px solid #ccc;
  border-radius: 3px;
  box-sizing: border-box;
}
.checkbox-group {
  margin-top: 10px;
}
.checkbox-group label {
  display: block;
}
button {
  background-color: #4CAF50;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 3px;
  cursor: pointer;
}
button:hover {
  background-color: #45a049;
```

```
    }
  </style>
</head>
<body>
  <div class="container">
    <h2>User Creation</h2>
    <form>
      <div class="form-group">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username">
      </div>
      <div class="form-group">
        <label for="email">Email:</label>
        <input type="text" id="email" name="email">
      </div>
      <div class="form-group">
        <label for="gender">Gender:</label>
        <select id="gender" name="gender">
          <option value="male">Male</option>
          <option value="female">Female</option>
          <option value="other">Other</option>
        </select>
      </div>
      <div class="form-group">
        <label>Interests:</label>
        <div class="checkbox-group">
          <label><input type="checkbox" name="interests" value="sports">Sports</label>
          <label><input type="checkbox" name="interests" value="music">Music</label>
          <label><input type="checkbox" name="interests" value="movies">Movies</label>
        </div>
      </div>
      <div class="form-group">
        <label for="country">Country:</label>
        <select id="country" name="country">
          <option value="USA">USA</option>
          <option value="Canada">Canada</option>
          <option value="UK">UK</option>
          <option value="Australia">Australia</option>
          <!-- Add more options as needed -->
        </select>
      </div>
      <button type="submit">Create User</button>
    </form>
  </div>
</body>
</html>
```

This HTML/CSS code creates a basic user creation form with text inputs, dropdowns, checkboxes, and a submit button. You can customize it further according to your needs.

3Create a file with depertment as root,year as subroot and student as an element

To create a file with a structure where the department is the root, the year is the sub-root, and students are ***elements***, we can use an XML format. Here's an example of how you might structure this:

XML

```
<departments>
  <department name="Computer Science">
    <year name="2024">
      <student>
        <name>John Doe</name>
        <id>CS2024-01</id>
        <gpa>3.8</gpa>
      </student>
      <student>
        <name>Jane Smith</name>
        <id>CS2024-02</id>
        <gpa>3.9</gpa>
      </student>
    </year>
    <year name="2023">
      <student>
        <name>Emily Johnson</name>
        <id>CS2023-01</id>)
        <gpa>3.7</gpa>
      </student>
    </year>
  </department>
  <department name="Mechanical Engineering">
    <year name="2024">
      <student>
        <name>Michael Brown</name>
        <id>ME2024-01</id>
        <gpa>3.6</gpa>
      </student>
    </year>
  </department>
</departments>
```

4) between Authorization and Authentication?

Difference Authorization and authentication are two crucial concepts in the realm of information security, often used together but serving different purposes. Here's a breakdown of their differences:

## Authentication

1. Definition: Authentication is the process of verifying the identity of a user or system. It answers the question, "Who are you?"
2. Purpose: Ensures that the entity is who it claims to be.
3. Process: Typically involves checking credentials like usernames and passwords, biometric data, or security tokens.
4. Examples:
    1. Entering a password to log into an email account.
    2. Using a fingerprint scan to unlock a smartphone.
    3. Entering a one-time password (OTP) sent to a mobile device.

## Authorization

1. Definition: Authorization is the process of determining if an authenticated entity has permission to access specific resources or perform certain actions. It answers the question, "What are you allowed to do?"
2. Purpose: Ensures that the authenticated entity has the appropriate permissions to access resources or perform operations.
3. Process: Involves checking permissions and roles associated with the authenticated entity against the resource or action they are attempting to access or perform.
4. Examples:
    1. Allowing an employee to access specific files on a company's network based on their role.
    2. Granting access to a particular feature within a software application.
    3. Restricting access to certain areas of a building based on security clearance.

## Key Differences

1. 1》 Order: Authentication always precedes authorization. First, the system verifies the identity (authentication) and then determines the permissions (authorization).
2. 2》 Scope: Authentication deals with identity verification, while authorization deals with access control and permissions.
3. 3》 Data Used: Authentication typically uses credentials like passwords, biometric data, or tokens. Authorization uses access control lists (ACLs), role-based access control (RBAC), and policies.

In summary, authentication is about validating identity, and authorization is about granting or denying permissions based on that validated identity.

5)Create a Login *Screen?*

Creating a login screen involves both frontend design and backend logic. Here's a basic example using HTML, CSS, and JavaScript for the frontend, and a simple backend logic explanation using Node.js with Express for illustration.

## Frontend (HTML and CSS)

### *HTML (login.html)*

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="login-container">
    <h2>Login</h2>
    <form id="loginForm">
      <label for="username">Username:</label>
      <input type="text" id="username" name="username" required>
      <label for="password">Password:</label>
      <input type="password" id="password" name="password" required>
      <button type="submit">Login</button>
    </form>
    <div id="message"></div>
  </div>
  <script src="script.js"></script>
</body>
```

```
</html>
```

*CSS (styles.css)*

```
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
}

.login-container {
    background-color: #fff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    text-align: center;
}

.login-container h2 {
    margin-bottom: 20px;
}

.login-container label {
    display: block;
    margin: 10px 0 5px;
}
```

```
.login-container input {
  width: 100%;
  padding: 10px;
  margin-bottom: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
}

.login-container button {
  padding: 10px 20px;
  background-color: #007bff;
  color: #fff;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.login-container button:hover {
  background-color: #0056b3;
}

#message {
  margin-top: 10px;
  color: red;
}
```

### *JavaScript (script.js)*

```
document.getElementById('loginForm').addEventListener('submit', function(e) {
  e.preventDefault();
```

```
    const username = document.getElementById('username').value;
    const password = document.getElementById('password').value;

    fetch('/login', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ username, password })
    })
    .then(response => response.json())
    .then(data => {
        if (data.success) {
            window.location.href = '/dashboard';
        } else {
            document.getElementById('message').innerText = data.message;
        }
    })
    .catch(error => {
        console.error('Error:', error);
        document.getElementById('message').innerText = 'An error occurred. Please try again.';
    });
});
```

## Backend (Node.js and Express)

### *Node.js and Express Setup*

1. *Install Node.js and npm*: Ensure Node.js and npm are installed on your system.

2. *Create a new project*: Initialize a new Node.js project using `npm init`.
3. *Install Express*: Install Express and other necessary dependencies using `npm install express body-parser`.

### *Server Code (server.js)*

```
const express = require('express');
const bodyParser = require('body-parser');

const app = express();
const port = 3000;

// Middleware
app.use(bodyParser.json());
app.use(express.static('public'));

// Dummy user data
const users = {
  user1: 'password123',
  user2: 'password456'
};

// Login endpoint
app.post('/login', (req, res) => {
  const { username, password } = req.body;

  if (users[username] && users[username] === password) {
    res.json({ success: true });
  } else {
    res.json({ success: false, message: 'Invalid username or password' });
  }
});

// Start the server
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}/`);
});
```

## Folder Structure

```
project-folder/
│
├── public/
│   ├── login.html
│   ├── styles.css
│   └── script.js
│
├── server.js └── package.json
```

## Steps to Run

4. Save the HTML, CSS, and JavaScript files in the `public` directory.
5. Run `npm install express body-parser` in the project folder.
6. Run the server using `node server.js`.
7. Open a web browser and navigate to `http://localhost:3000/login.html`.

1)Difference between JSON and XML?

JSON (JavaScript Object Notation) and XML (eXtensible Markup Language) are both popular data interchange formats used in web development and beyond, but they have some key differences:

1. **Syntax**: JSON uses a simpler syntax than XML, making it easier for humans to read and write. JSON data is structured as key-value pairs, while XML uses tags and attributes to define data hierarchy.

2. **Extensibility**: XML is more extensible than JSON. XML allows for the creation of custom tags and attributes, making it suitable for representing complex data structures and metadata. JSON, on the other hand, has a fixed data model.

3. **Data Types**: JSON supports a limited set of data types, including strings, numbers, booleans, arrays, and objects. XML has no built-in data types and relies on the application to interpret the data.

4. **Parsing**: JSON parsing is generally faster and easier than XML parsing, especially in JavaScript environments where JSON is a native data type.

5. **Usage**: JSON is commonly used for transmitting data between a server and a web application, especially in AJAX requests. XML is often used for configuration files, document storage, and data exchange between heterogeneous systems.

Ultimately, the choice between JSON and XML depends on factors such as data complexity, interoperability requirements, and personal preference.

2)Create 3 XML and JSON files for department,year,student ?

Both JSON and XML can be used to receive data from a web server.

Certainly! Below are examples of XML and JSON files for departments, years, and students

Xml

```
<?xml version="1.0" encoding="UTF-8"?>

<departments>

  <department>

    <id>1</id>

    <name>Computer Science</name>

  </department>

  <department>

    <id>2</id>

    <name>Mathematics</name>

  </department>

  <department>

    <id>3</id>

    <name>Physics</name>

  </department>

</departments>
```

*years.xml*

```
<?xml version="1.0" encoding="UTF-8"?>

<years>

  <year>

    <id>1</id>

    <name>First Year</name>

  </year>

  <year>

    <id>2</id>

    <name>Second Year</name>
  </year>
  <year>
    <id>3</id>
    <name>Third Year</name>
  </year>
</years>
```

*students.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<students>
  <student>
    <id>101</id>
    <name>John Doe</name>
    <department>Computer Science</department>
    <year>First Year</year>
  </student>
  <student>
    <id>102</id>
    <name>Jane Smith</name>
    <department>Mathematics</department>
    <year>Second Year</year>
  </student>
  <student>
    <id>103</id>
    <name>Emily Johnson</name>
    <department>Physics</department>
    <year>Third Year</year>
  </student>
</students>
```

# JSON Files

### *departments.json*

```json
{
  "departments": [
    {
      "id": 1,
      "name": "Computer Science"
    },
    {
      "id": 2,
      "name": "Mathematics"
    },
    {
      "id": 3,
      "name": "Physics"
    }
  ]
}
```

### *years.json*

```json
{
  "years": [
    {
      "id": 1,
      "name": "First Year"
    },
    {
      "id": 2,
      "name": "Second Year"
    },
    {
      "id": 3,
      "name": "Third Year"
    }
  ]
}
```

*students.json*

```
{
  "students": [
    {
      "id": 101,
      "name": "John Doe",
      "department": "Computer Science",
      "year": "First Year"
    },
    {
      "id": 102,
      "name": "Jane Smith",
      "department": "Mathematics",
      "year": "Second Year"
    },
    {
      "id": 103,
      "name": "Emily Johnson",
      "department": "Physics",
      "year": "Third Year"
    }
  ]
}
```

These examples should give you a clear structure for XML and JSON files for departments, years, and students.