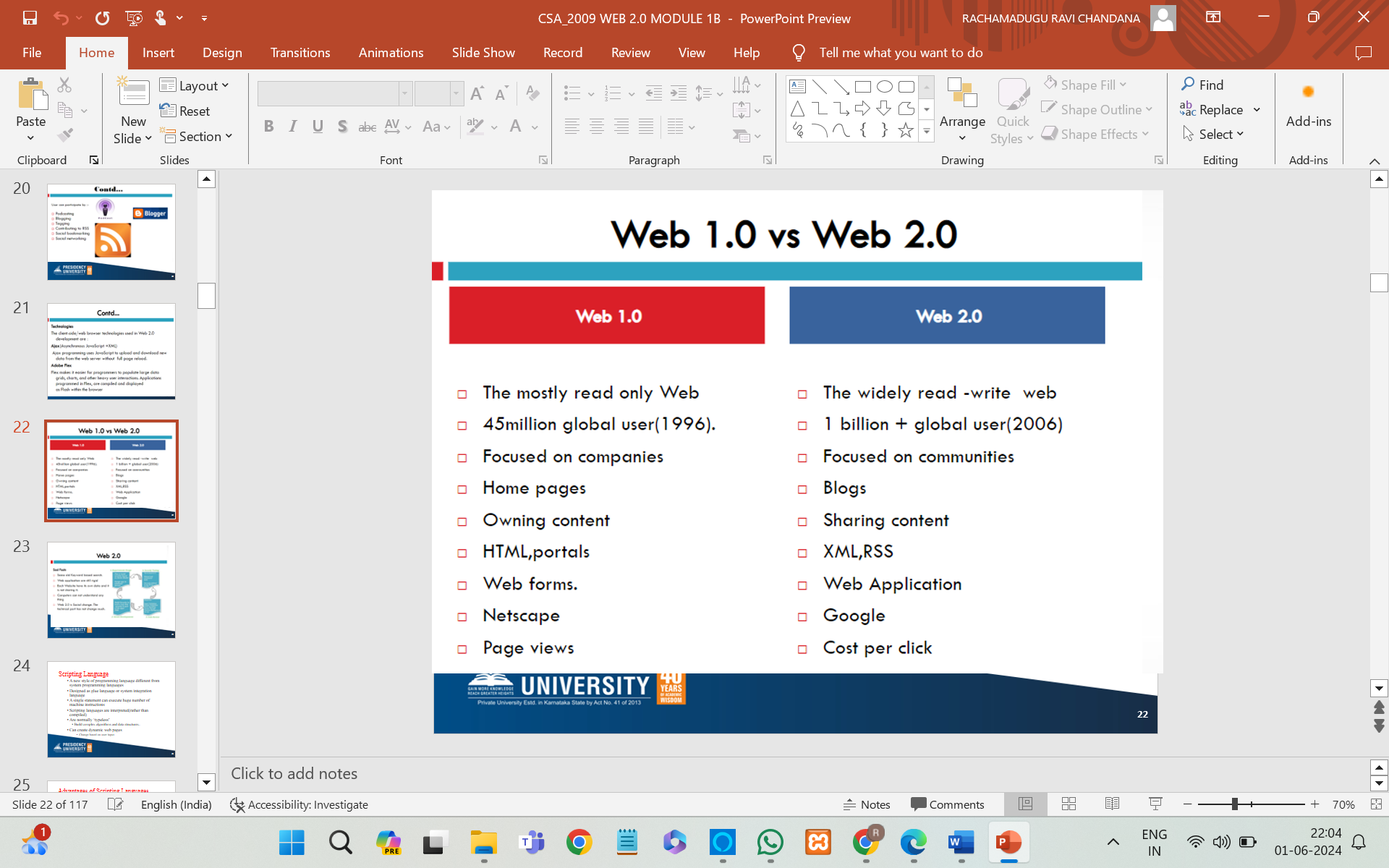
**4 MARKS**

1)



* 2) Server-side Scripting Language
  + Can use huge resources of the server
  + Complete all processing in the server and send plain pages to the client
  + Reduces client-side computation overhead
  + Run on web servers
  + allows for the creation of dynamic web pages that can be customized based on user input or other data.
  + Example: - ASP, JSP, Servlets, PHP

## 3)jQuery Selectors

jQuery selectors are powerful tools used to target and manipulate DOM (Document Object Model) elements. They allow you to find HTML elements based on their id, class, attributes, types, and more. Here’s a basic overview of how to use various jQuery selectors:

jQuery selectors allow you to select and manipulate HTML element(s).

## **The element Selector**

The jQuery element selector selects elements based on the element name.

$("p")

## **The #id Selector**

The jQuery #id selector uses the id attribute of an HTML tag to find the specific element.

$("#test")

## **The .class Selector**

The jQuery .class selector finds elements with a specific class.

To find elements with a specific class, write a period character, followed by the name of the class:

$(".test")

4) **XML Syntax and Rules**

* The syntax rules of XML are very simple and logical. The rules are easy to learn, and easy to use.
* XML Documents Must Have a Root Element

XML documents must contain one **root** element that is the **parent** of all other elements

* The XML prolog is optional. If it exists, it must come first in the document.
* XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.

Opening and closing tags must be written with the same case

* In XML, all elements **must** be properly nested within each other

<b><i>This text is bold and italic</i></b>

## **XML Element:**

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

An element can contain:

* text
* attributes
* other elements
* or a mix of the above
* ex: <price>29.99</price>

## **XML Attributes :**

Attribute values must always be quoted. Either single or double quotes can be used.

For a person's gender, the <person> element can be written like this:

<person gender="female">

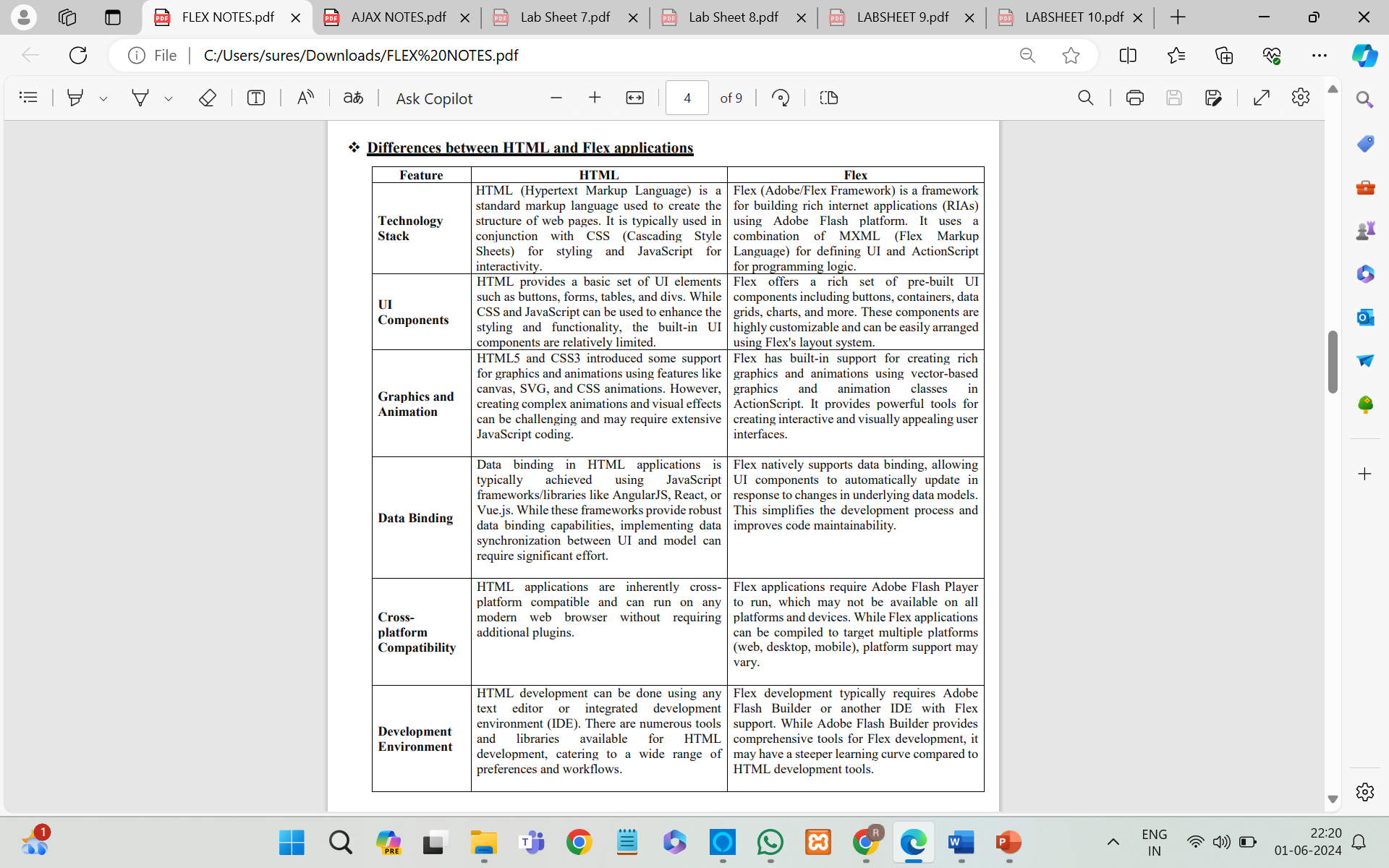
# **XML Namespaces :**

XML Namespaces provide a method to avoid element name conflicts.

When using prefixes in XML, a **namespace** for the prefix must be defined.

The namespace can be defined by an **xmlns** attribute in the start tag of an element.

* a namespace is a container that holds a set of identifiers, such as functions, classes, or variables, to organize and manage code.
* Namespaces are used to avoid naming conflicts between different components or libraries within an application.

5)

7) Periodic Refresh Periodic refresh in AJAX involves automatically updating content on a web page at regular intervals without requiring user interaction. This technique is commonly used to display real-time or near-real-time data, such as news feeds, social media updates, or live statistics. AJAX enables periodic refresh by asynchronously fetching updated content from the server and dynamically updating the page without reloading the entire document.

Here are some common areas where periodic refreshing is utilized:

### **News and Information Websites,Weather Updates**,**Social Media Platforms**,**E-commerce Websites**,**Messaging and Collaboration Tools** ,**Financial Services**,**Online Gaming**,**Transport and Travel**, **Customer Support**, **IoT and Smart Devices**

Periodic refreshing is crucial in any context where timely information is necessary for effective decision-making, user engagement, or system functionality. By keeping data current and ensuring that users have access to the latest information, periodic refreshing enhances the overall user experience and operational efficiency.

8) Submission throttling is a technique used to control the rate at which users or systems can make submissions to a server or service. This involves limiting the number of requests or actions (such as form submissions, API calls, or other data transmissions) that can be made within a specific time frame. The goal is to ensure that the server or service is not overwhelmed by a high volume of requests, which could lead to performance degradation, downtime, or denial of service.

### Why is Submission Throttling Used?

#### 1. **Preventing Abuse and Spam**

Throttling helps prevent abuse by limiting the ability of malicious users or automated scripts (bots) to flood a system with requests. This is particularly important for web forms, APIs, and other entry points that could be exploited to submit spam or carry out brute-force attacks.

#### 2. **Managing Server Load**

By controlling the rate of incoming requests, submission throttling helps manage the server's load, ensuring it remains responsive and operational. This is critical for maintaining the performance and availability of the service, especially during peak usage times or unexpected traffic spikes.

#### 3. **Ensuring Fair Usage**

Throttling mechanisms ensure that all users have fair access to the system's resources. Without throttling, a small number of users could potentially monopolize the server's capacity, degrading the experience for others.

#### 4. **Enhancing Security**

In authentication systems, throttling login attempts can mitigate brute-force attacks by limiting the number of attempts a user can make within a given period. This reduces the risk of unauthorized access.

#### 5. **Improving User Experience**

By preventing server overload and maintaining consistent performance, throttling helps provide a better user experience. Users are less likely to encounter slowdowns or service interruptions caused by excessive demand.

### Examples of Where Submission Throttling is Used

1. **Web Forms**: To prevent spam submissions and ensure that legitimate users can use the form without interference.
2. **APIs**: To control the rate of API requests from clients, ensuring the backend services are not overwhelmed.
3. **Authentication Systems**: To limit the number of login attempts and prevent brute-force attacks.
4. **E-commerce Sites**: To manage high volumes of transactions during sales or promotions.
5. **Social Media Platforms**: To limit the rate of posting or interactions to prevent spam and maintain platform integrity.

### Implementation Strategies

1. **Fixed Window Rate Limiting**: Limits the number of submissions within a fixed time window (e.g., 100 requests per minute).
2. **Sliding Window Rate Limiting**: Provides a more granular approach by using sliding time windows.
3. **Token Bucket Algorithm**: Uses tokens to control the rate. Tokens are added to the bucket at a fixed rate, and each submission consumes a token.
4. **Leaky Bucket Algorithm**: Queues incoming requests and processes them at a fixed rate.
5. **User-Based Throttling**: Limits submissions based on individual user accounts or IP addresses.

### Conclusion

Submission throttling is a crucial mechanism for maintaining the stability, security, and fairness of online services. By controlling the rate of incoming requests, systems can prevent abuse, manage server load, ensure fair resource usage, enhance security, and improve the overall user experience. Implementing effective throttling strategies is essential for any system that handles a high volume of submissions or interactions.

* 1. **MARKS**

**1**) 1. Database Setup:

 Create a MySQL database named "employee\_management" with a table named

"employees" to store employee details such as name, department, position, and

hire date.

2. PHP Scripts for CRUD Operations:

 Develop PHP scripts:

 Add Employee: Insert new employee records into the "employees" table.

 Edit Employee: Update existing employee records in the "employees"

table.

 Delete Employee: Delete employee records from the "employees" table.

 View Employee: Fetch and display employee details from the

"employees" table.

3. User Interface Development:

 Design a simple web interface using HTML and PHP forms:

 Add Employee Form: Collect new employee details and submit them to

the PHP script for insertion.

 Edit Employee Form: Allow users to modify existing employee details

and update them in the database through the PHP script.

 Delete Employee Button: Provide a button or link to trigger the deletion

of employee records via the PHP script.

 View Employee Details: Display employee details fetched from the

database using PHP.

4. Authentication and Authorization:

 Implement basic authentication to ensure only authorized users can access the

system and perform CRUD operations.

Explanation:

 This simple approach uses PHP and MySQL to create an employee management system

with basic CRUD functionality.

 PHP scripts handle database interactions, while HTML forms facilitate user input and

interaction.

 Authentication ensures data security by allowing only authorized users to access and

modify employee records.

 Overall, this solution provides a straightforward way to manage employee records for

the organization.

**2)** A website needs to implement user registration and authentication functionality. How would you use PHP and MySQL to allow users to register, log in, and manage their accounts securely?

**Emp.html**

<!DOCTYPE html>

<html lang="en-us">

<head> <title>Employee Details</title> </head>

<body>

<h1>Employee Details</h1>

<form action="emp.php" method="post">

<br><h3>Search Employee Details Based on Employee ID</h3>

<br>Enter Employee ID:<input type="number" id="Id" name="Id" >

<br><br><input type="submit" value="Search" onclick="SaveEmpDetails()">

</form>

<form action="emp.php" method="post">

<br><h3>Search Employee Details Based on Salary</h3>

<br>Enter Employee Salary Range:<input type="number" id="SS" name="SS" >

Between

<input type="number" id="LS" name="LS" >

<br><br><input type="submit" value="Search" onclick="SaveEmpDetails()">

</form>

</body>

</html>

</body>

</html>

**Connect.php**

<?php

$servername = "localhost";

$username = "root";

$password = "";

// Create connection

$conn = new mysqli($servername, $username, $password);

// Check connection

if ($conn->connect\_error) {

die("Connection failed: " . $conn->connect\_error);

}

else {

echo "Connection established succesfully";

}

$conn->close();

?>

**Created.php**

<?php

$servername = "localhost";

$username = "root";

$password = "";

// Create connection

$conn = new mysqli($servername, $username, $password);

// Check connection

if ($conn->connect\_error) {

die("Connection failed: " . $conn->connect\_error);

}

// Create database

$sql = "CREATE DATABASE 4BCA";

if ($conn->query($sql) === TRUE) {

echo "Database created successfully";

} else {

echo "Error creating database: " . $conn->error;

}

$conn->close();

?>

**Createtb.php**

**<**?php

$servername = "localhost";

$username = "root";

$password = "";

$dbname = "4BCA";

// Create connection

$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection

if ($conn->connect\_error) {

die("Connection failed: " . $conn->connect\_error);

}

// sql to create table

$sql = "CREATE TABLE section (

rollno INT(6) UNSIGNED PRIMARY KEY,

name VARCHAR(30) NOT NULL,

email VARCHAR(50)

)";

if ($conn->query($sql) === TRUE) {

echo "Table SECTION created successfully";

} else {

echo "Error creating table: " . $conn->error;

}

$conn->close();

?>

**Insertdb.php**

<?php

$servername = "localhost";

$username = "root";

$password = "";

$dbname = "6cai2"; // DB Name

$v1=$\_POST["rollno"]; //name attribute value

$v2=$\_POST["name"];//name attribute value

$v3=$\_POST["email"];//name attribute value

// Create connection

$conn = mysqli\_connect($servername, $username, $password, $dbname);

// Check connection

if (!$conn)

{

die("Connection failed: " . mysqli\_connect\_error());

}

$sql = "INSERT INTO section (rollno, name, email) VALUES ('$v1','$v2','$v3')";

if ($conn->query($sql) === TRUE)

{

echo "New record created successfully";

}

else

{

echo "Error: " . $sql . "<br>" . mysqli\_error($conn);

}

mysqli\_close($conn);

?>

**Select.php**

<?php

$servername = "localhost";

$username = "root";

$password = "";

$dbname = "SANDHYA"; // DB Name

// Create connection

$conn = mysqli\_connect($servername, $username, $password, $dbname);

// Check connection

if (!$conn)

{

die("Connection failed: " . mysqli\_connect\_error());

}

$sql = "SELECT rollno, name, email FROM section1";

$result = $conn->query($sql);

if ($result->num\_rows > 0) {

// output data of each row

while($row = $result->fetch\_assoc()) {

echo "<br> RollNo: ". $row["rollno"]. " - Name: ". $row["name"]. " - Email: " . $row["email"] . "<br>";

}

} else {

echo "0 results";

}

$conn->close();

?>

**Delete.php**

<?php

$servername = "localhost";

$username = "root";

$password = "";

$dbname = "SANDHYA"; // DB Name

$conn = mysqli\_connect($servername, $username, $password, $dbname);

if (!$conn)

{

die("Connection failed: " . mysqli\_connect\_error());

}

$sql = "DELETE FROM section1 WHERE rollno='1234'";

if ($conn->query($sql) === TRUE) {

echo "Record deleted successfully";

} else {

echo "Error deleting record: " . $conn->error;

}

$conn->close();

?>

**MODULE-2**

1.You are working on an employee directory system, and the employee data is stored in an XML file. Define an XML schema for employee records with attributes such as name, department, position, and hire date. Develop an XSLT stylesheet to present this employee information in an HTML table.

**Employee.xml**

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

<employees>

<employee>

<name>John Doe</name>

<department>Engineering</department>

<position>Software Engineer</position>

<hireDate>2020-01-15</hireDate>

</employee>

<employee>

<name>Jane Smith</name>

<department>Marketing</department>

<position>Marketing Manager</position>

<hireDate>2018-07-23</hireDate>

</employee>

<employee>

<name>Robert Brown</name>

<department>Sales</department>

<position>Sales Representative</position>

<hireDate>2019-03-12</hireDate>

</employee>

</employees>

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

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:output method="html" encoding="UTF-8" indent="yes"/>

**Xslt stylesheet**

<xsl:template match="/">

<html>

<head>

<title>Employee Directory</title>

<style>

table {

width: 100%;

border-collapse: collapse;

}

table, th, td {

border: 1px solid black;

}

th, td {

padding: 10px;

text-align: left;

}

th {

background-color: #f2f2f2;

}

</style>

</head>

<body>

<h2>Employee Directory</h2>

<table>

<tr>

<th>Name</th>

<th>Department</th>

<th>Position</th>

<th>Hire Date</th>

</tr>

<xsl:for-each select="employees/employee">

<tr>

<td><xsl:value-of select="name"/></td>

<td><xsl:value-of select="department"/></td>

<td><xsl:value-of select="position"/></td>

<td><xsl:value-of select="hireDate"/></td>

</tr>

</xsl:for-each>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

**2)**Imagine you are organizing a conference and storing the schedule information in XML. Design an XML schema for conference sessions with elements like title, speaker, time, and location. Write an XSLT stylesheet to transform this XML into an HTML schedule for the conference.

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

<conference>

<session>

<title>Keynote Address</title>

<speaker>Dr. Alice Johnson</speaker>

<time>09:00 - 10:00</time>

<location>Main Hall</location>

</session>

<session>

<title>Machine Learning Basics</title>

<speaker>John Smith</speaker>

<time>10:15 - 11:15</time>

<location>Room 101</location>

</session>

<session>

<title>Introduction to Cybersecurity</title>

<speaker>Jane Doe</speaker>

<time>11:30 - 12:30</time>

<location>Room 102</location>

</session>

<session>

<title>Lunch Break</title>

<speaker></speaker>

<time>12:30 - 13:30</time>

<location>Cafeteria</location>

</session>

<session>

<title>Advanced AI Techniques</title>

<speaker>Dr. Robert White</speaker>

<time>13:45 - 14:45</time>

<location>Room 103</location>

</session>

<session>

<title>Closing Remarks</title>

<speaker>Dr. Alice Johnson</speaker>

<time>15:00 - 16:00</time>

<location>Main Hall</location>

</session>

</conference>

**Xslt style sheet**

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

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:output method="html" encoding="UTF-8" indent="yes"/>

<xsl:template match="/">

<html>

<head>

<title>Conference Schedule</title>

<style>

table {

width: 100%;

border-collapse: collapse;

margin-bottom: 20px;

}

table, th, td {

border: 1px solid black;

}

th, td {

padding: 10px;

text-align: left;

}

th {

background-color: #f2f2f2;

}

h2 {

text-align: center;

}

</style>

</head>

<body>

<h2>Conference Schedule</h2>

<table>

<tr>

<th>Title</th>

<th>Speaker</th>

<th>Time</th>

<th>Location</th>

</tr>

<xsl:for-each select="conference/session">

<tr>

<td><xsl:value-of select="title"/></td>

<td><xsl:value-of select="speaker"/></td>

<td><xsl:value-of select="time"/></td>

<td><xsl:value-of select="location"/></td>

</tr>

</xsl:for-each>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

**MODULE-3**

1. You're tasked with developing a dynamic dashboard for a business analytics application using Flex. Describe how you would leverage Flex architecture to create a modular and responsive dashboard interface. Discuss the use of MXML for defining layout structure and ActionScript for implementing dynamic data binding and event handling.

1. Flex Architecture with MXML:

- Model: In Flex, the Model represents the data and business logic. You can define your data models in ActionScript classes.

- View: The View in Flex is responsible for the user interface. We can use MXML to define the layout structure of the dashboard. MXML provides a declarative way to define the UI components and their properties.

- Controller: The Controller handles the interaction between the Model and View. It's often implemented in ActionScript to manage data flow and business logic.

2. MXML for Layout Structure:

- MXML is a markup language used in Flex to define the layout structure of the user interface. You can use MXML tags to create UI components like containers, buttons, charts, etc., and arrange them in a visually appealing way.

- MXML allows you to specify properties and event handlers for these components, making it easy to create interactive and dynamic interfaces.

3. ActionScript for Dynamic Data Binding and Event Handling:

- ActionScript is the programming language used in Flex for creating dynamic and interactive applications.

- You can use ActionScript to implement dynamic data binding, where changes in data automatically reflect in the UI components. This is useful for real-time updates on the dashboard based on changing data.

- ActionScript is also used for event handling. You can define event listeners in ActionScript to respond to user interactions like button clicks, mouse movements, etc. This helps in creating a responsive dashboard with interactive elements.

By combining MXML for layout structure and ActionScript for implementing dynamic data binding and event handling, you can develop a powerful and user-friendly dashboard for the business analytics application. This approach ensures modularity, responsiveness, and interactivity in the dashboard design.

1. How does Flex utilize UI (User Interface) components as the fundamental building blocks for creating the visual interface of an application, and what benefits do developers derive from Flex's provision of a rich set of pre-built UI components for designing interactive and visually appealing user interfaces?

Flex uses UI components as the basic elements to build the visual interface of an application. These components are like building blocks that developers can use to create interactive and visually appealing user interfaces easily. By providing a set of pre-built UI components, Flex offers developers several benefits:

1. Ease of Use: Developers can quickly design interfaces by assembling pre-built UI components like buttons, text inputs, dropdowns, and more. This saves time and effort in creating custom UI elements from scratch.

2. Consistency: Flex's pre-built UI components follow a consistent design language, ensuring a uniform look and feel across the application. This consistency enhances user experience and makes the application more professional.

3. Interactivity: The pre-built UI components in Flex come with built-in interactive functionalities, such as handling user input, responding to events like clicks or hovers, and displaying dynamic content. This makes it easier to create engaging user experiences.

4. Customization: While using pre-built components, developers can still customize them to suit the application's specific design requirements. Flex allows for easy styling and customization of UI components to match the branding or theme of the application.

5. Rich Functionality: Flex provides a wide range of UI components that offer rich functionality out of the box. Developers can leverage components like data grids, charts, accordions, and more to enhance the application's capabilities without reinventing the wheel.

In summary, Flex's provision of pre-built UI components simplifies the process of designing interactive and visually appealing user interfaces, offering developers a faster and more efficient way to create engaging applications with consistent design patterns and rich functionality.

**MODULE 4**

1. You're tasked with optimizing the loading time of a single-page application (SPA) that heavily relies on AJAX requests to fetch dynamic content. Explain how you would implement page preloading to enhance the perceived performance and user experience. Consider strategies for prefetching essential resources and prioritizing critical content.

To optimize the loading time of a single-page application (SPA) that relies heavily on AJAX requests, implementing page preloading can significantly enhance perceived performance and user experience. Here's how you can do it:

1. **Identify Critical Resources**: Analyze your SPA to determine which resources are critical for initial rendering and user interaction. This includes CSS files, JavaScript libraries, and data required for the first view of the application.
2. **Prefetch Essential Resources**: Use prefetching techniques to load essential resources in advance. This can be achieved by adding **<link rel="prefetch">** or **<link rel="preload">** tags in the HTML **<head>** section.
3. **Lazy Load Non-Essential Content**: Defer loading of non-critical content or resources that are not immediately needed for the initial view. This can include images, additional JavaScript, or data for secondary views. Lazy loading can be implemented using libraries like **Intersection Observer** or by listening to scroll events.
4. **Prioritize Critical AJAX Requests**: Identify AJAX requests that fetch critical data for rendering the initial view. Prioritize these requests by ensuring they are triggered early in the page lifecycle. You can use asynchronous JavaScript techniques or libraries like **axios** to manage AJAX requests efficiently.
5. **Cache AJAX Responses**: Utilize browser caching mechanisms or service workers to cache AJAX responses whenever possible. This reduces the need for repeated requests and improves overall loading speed, especially for subsequent visits or interactions within the SPA.
6. **Optimize Bundle Size**: Minimize and bundle CSS and JavaScript files to reduce the overall payload size. This improves loading performance, especially on slower networks or devices. Tools like Webpack or Parcel can help automate this process.
7. **Progressive Rendering**: Implement progressive rendering techniques to start rendering content as soon as possible, even before all resources are fully loaded. This gives users a sense of instant response and improves perceived performance.
8. **Monitor and Fine-tune**: Continuously monitor and analyze the performance of your SPA using tools like Google PageSpeed Insights or Lighthouse. Fine-tune your preloading strategy based on real-world usage patterns and user feedback to ensure optimal performance over time.

By implementing these strategies for page preloading, you can enhance the perceived performance and user experience of your single-page application, making it feel faster and more responsive to users.

2) Suppose you're building a web application that heavily relies on AJAX requests for dynamic content loading. Describe a robust fallback pattern you would implement to ensure seamless user experience in case AJAX requests fail. Discuss strategies for graceful degradation and alternative content delivery methods.

Ensuring a seamless user experience in case AJAX requests fail is crucial for the reliability of a web application. Here's a robust fallback pattern along with strategies for graceful degradation and alternative content delivery methods:

1. **Client-Side Error Handling**: Implement robust client-side error handling for AJAX requests. Use **try-catch** blocks or **.catch()** methods to capture errors and handle them gracefully without crashing the application.
2. **Retry Mechanism**: Implement a retry mechanism for failed AJAX requests. You can automatically retry failed requests a certain number of times with increasing delay between retries to mitigate temporary network issues. Libraries like Axios provide built-in support for retries.
3. **Offline Support with Service Workers**: Utilize service workers to cache essential resources and enable offline support. When AJAX requests fail due to network issues, the application can serve cached data from the service worker, providing users with a limited but functional experience even when offline.
4. **Fallback Data**: Provide fallback data or default content to display in case AJAX requests fail. This ensures that users still see meaningful information instead of encountering empty or broken sections of the application. Fallback data can be preloaded along with the initial page load or retrieved from a local cache.
5. **Progressive Enhancement**: Design the application to progressively enhance functionality based on available features and resources. Ensure that critical functionality can still be accessed even if AJAX requests fail or if JavaScript is disabled. This involves building a solid foundation with server-side rendering and enhancing it with client-side interactions.
6. **Server-Side Rendering**: Utilize server-side rendering (SSR) to generate initial HTML content on the server and deliver it to the client. This ensures that users receive meaningful content even if JavaScript fails to execute or AJAX requests cannot be processed. SSR can be combined with client-side hydration for enhanced interactivity.
7. **User Feedback**: Provide clear and informative error messages to users when AJAX requests fail. Explain the issue and suggest possible actions, such as checking the network connection or refreshing the page. Good error messages help users understand what went wrong and how they can resolve the issue.
8. **Monitoring and Analytics**: Implement monitoring and analytics to track AJAX request failures and user interactions. Use this data to identify recurring issues, optimize error handling strategies, and continuously improve the reliability of the application.

By implementing these strategies for fallback patterns, graceful degradation, and alternative content delivery methods, you can ensure a seamless user experience in your web application even when AJAX requests encounter issues.