VIIT, PUNE

**BAN**



LAB MANUAL

**WEB TECHNOLOGY LAB**

FOR

Course: **CA22233** Pattern: NEP 2023

Issue Date: Copy No: 01

**BANSILAL RAMNATH AGARWAL CHARITABLE TRUST’S VISHWAKARMA INSTITUTE OF INFORMATION TECHNOLOGY**

**Sr.No.2/3/4, Kondhwa (BK), Pune -48**

****

# Department of Computer Engineering

[Artificial Intelligence]

## LAB MANUAL

FOR

## A.Y. 2024-25

**Web Technology Lab**

Subject Code: **CA22233**

**Teaching Scheme Examination Scheme**

Practical: 2 Hrs. / Week Term Work: 20 Marks

Practical Incharge HOD

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sr. no.** | **Name of Assignment** | **Page No.** |
| 1 | Using HTML5/CSS3 develop informative and responsive web page for Library Management System. |  |
| 2 | Create a user registration form for Library Management System  in HTML and write a JavaScript Program to apply form validation and to get user registration details. |  |
| 3 | Manipulate and design the user registration form using CSS3 selector with jQuery functions. |  |
| 4 | Create a CRUD application using AJAX. |  |
| 5 | Develop an application for a Library Management System using Angular js and MongoDB. The application able to  perform the Create, Delete, Modify the data in database. |  |
| 6 | Develop a Mail applications using PHP and SQL |  |
| 7 | Develop an application for a Library Management System using Node.js and MongoDB. The application able to perform the  CRUD operations |  |
| 8 | Develop dashboard for Library Management System having working functionalities of Login, Book Issue and return. Use any suitable technology like angular, node, php or any you are  comfortable |  |
| Annexure | Distribution List |  |

Prepared by, Lab Coordinator

# Assignment No - 1

**PROBLEM STATEMENT:** “Using HTML5/CSS3 develop informative and responsive web page for Library Management System.”

### OBJECTIVES:

* To develop webpage with simple formatting styles.
* To apply page background colors.
* To apply html body test color.
* To apply various font attributes and styles on text.
* To apply various CSS styles on HTML elements

### SYSTEM REQUIREMENTS:

**Minimum Hardware Requirements**

MEMORY (RAM): 2GB

Hard Disk Capacity: 40GB

Processor Speed: 1.6 GHz or faster processor

**Minimum Software Requirements**

Operating System: Windows / Linux

Editor: Notepad/sublime/Visula Studio Code/ Atom

Browser: Chrome/ Microsoft Edge / Mozila Firefox / Safari /Opera

### THEORY:

**Web site:** A set of interconnected web pages, usually including a homepage, generally located on the same server, and prepared and maintained as a collection of information by a person, group, or organization.

**Web Page:** A web page is a document that's created in html that shows up on the internetwhen you type in or go to the web page's address.

**Types of Web Pages:**

1. **Static web page**: is delivered exactly as stored, as web content in the web server's file system. Contents cannot be changed.
2. **Dynamic web page**: is generated by a web application that is driven by server-side software or client-side scripting. Dynamic web pages help the browser (the client) to enhance the web page through user input to the server. Contents can be changed as evolution over time.

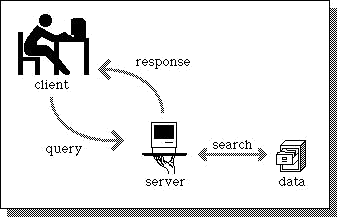
**Browsers & their types**

A web browser (commonly referred to as a browser) is a software application for retrieving, presenting and traversing information resources on the World Wide Web.

The major web browsers are Google Chrome, Firefox, Internet Explorer, Opera, and Safari.

**Client –Server Model**

The client–server model is a distributed application structure in computing that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network. A server is a host that is running one or more server programs which share their resources with clients. A client requests a server's content or service function.



**Web –Server**

Web server refers to either the hardware (the computer) or the software (the computer application) that helps to deliver web content that can be accessed through the Internet. The most common use of web servers is to host websites, but there are other uses such as gaming, data storage or running enterprise application.

**Working of different types of web pages**

The different types of web pages are :

**Advocacy:** An advocacy web page is one sponsored by an organization to influence opinion. URL ends with .org

**Business and marketing:** It is one sponsored by a commercial enterprise to sell ormarket their services. URL ends with .com

**News:** It provides timely information about current events and issues.

**Informational:** This includes reports, research findings, schools and college information. URL ends with .edu or .gov.

**Personal:** It is created by an individual for his /her own personal need.URL has tidle(~).

**General structure of a Web Page**

A basic HTML page contains a Head section and a Body section. The contents of the head section are normally invisible in a web browser and mainly consists of some Metatags. The Body consist of those HTML elements that you want to have displayed in your browser.

<html>

<head>

</head>

<body>

</body>

</html>

**Scripting language:** A scripting language or script language is a programming language that supports the writing of scripts, programs written for a special runtime environment that can interpret and automate the execution of tasks which could alternatively be executed one- by-one by a human operator.

**URL:** A uniform resource locator (URL), also known as web address, is a specific character string that constitutes a reference to a resource. In most web browsers, the URL of a web page is displayed on top inside an address bar. An example of a typical URL would be "<http://en.example.org/wiki/Main_Page>".

**Popular Search Engines**

* Yahoo Search
* Google Search
* Bing
* Info.com
* Search.com
* Infospace

**WWW:** The World Wide Web (WWW) is a system of interlinked hypertext documents accessed via the Internet. With a web browser, one can view web pages that may contain text, images, videos, and other multimedia, and navigate between them via hyperlinks.

**ILLUSTRATING HTML TAGS AND THEIR ATTRIBUTES**

**HTML:**Hyper Text Markup Language is the most widely used language to write webpages.it is a markup language.

**Hypertext :**Refers to the way in which web pages are linked together.

**Markup Language:** The user simply markups a text document with tags that tell a webbrowser how to structure it to display.

**Creating HTML document :**To begin coding HTML user needs only two things:

1. A simple text editor (notepad).
2. A web browser.

**Simple steps to create a basic HTML document:**

1. Open notepad or another text editor.
2. At the top of the page type <html>
3. Add the opening header tag <head>
4. On the next line type <title> give title for page </title>
5. Go to next line and type closing header tag </head>
6. Go to next line and type opening body tag<body>
7. Go to next line and type closing body tag</body>
8. Finally, go to next line and type</html>
9. In the file menu,choose save as.
10. In the save as type option box,choose all files.
11. Name the file filename.html
12. Click save.

**HTML document structure:** An HTML document starts and ends with <html> and

</html> tags.These tags tell the browser that the entire document is composed in HTML.Inside these two tags,the document is split into 2 sections:

1. The <head>……</head> elements contains information about the document such as title of the document etc.
2. The <body>…. </body> elements contains the real content of the document that you see on your screen.

### ATTRIBUTES:

An attribute is used to define the characteristics of an element and is placed inside the element’s opening tag. All attributes are made up of 2 parts: a name and a value.

-The name is the property you want to set.

-The value is what you want the value of the property to be. Example: <font face=”arial” color=”red”>

**HTML (Hypertext Markup Language):**

HTML is the standard markup language used to create web pages. It provides the structure and

content of a web page. In the program, we have used various HTML elements:

<!DOCTYPE html>: This declaration defines the document type and version of HTML being used, which is HTML5 in this case.

<html>: The <html> element is the root element of an HTML document and contains all other HTML elements.

<head>: The <head> element contains meta-information about the document, such as the title of the page, and can include links to external resources like CSS files.

<title>: The <title> element sets the title of the web page, which appears in the browser's title bar or tab.

<body>: The <body> element contains the visible content of the web page, including text, images, and other elements.

<h1> and <p>: These are headings and paragraphs, respectively. <h1> represents the main heading of the page, and <p> represents paragraphs of text.

<center> (deprecated): The <center> element was used to center-align the content. However, it's considered outdated and should be avoided in favor of CSS for alignment.

<style>: The <style> element is used to embed CSS code within the HTML document.

h1 and p: These selectors target the <h1> and <p> elements, respectively. They are used to set the font family, font size, and font color for the headings and paragraphs on the page.

### CODE:

### HTML: <!DOCTYPE html>

### <html lang="en">

### <head>

### <meta charset="UTF-8">

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

### <title>Library Management System</title>

### *<!-- Google Fonts -->*

### <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@600&family=Roboto:wght@400&display=swap" rel="stylesheet">

### *<!-- Bootstrap CSS CDN -->*

### <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">

### <link rel="stylesheet" href="Assignment1.css">

### </head>

### <body>

### <header>

### <h1>Library Management System</h1>

### </header>

### *<!-- Bootstrap Container -->*

### <div class="container mt-5">

### <div class="container-custom">

### <h2 class="title">Books Available</h2>

### *<!-- Bootstrap Responsive Row -->*

### <div class="row text-center fw-bold">

### <div class="col-4">Title</div>

### <div class="col-4">Author</div>

### <div class="col-4">Issued Date</div>

### </div>

### <hr>

### <div class="row text-center">

### <div class="col-4">Book Title 1</div>

### <div class="col-4 author">Author 1</div>

### <div class="col-4 issued-date">12-05-2024</div>

### </div>

### <hr>

### <div class="row text-center">

### <div class="col-4">Book Title 2</div>

### <div class="col-4 author">Author 2</div>

### <div class="col-4 issued-date">02-12-2024</div>

### </div>

### </div>

### </div>

### *<!-- Bootstrap JavaScript Bundle (for interactive components, if needed) -->*

### <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>

### </body>

### </html>

### Css: header {

### background-color: #5b40c7;

### text-align: center;

### padding: 15px 0;

### }

### header h1 {

### color: beige;

### font-family: "Poppins", sans-serif;

### font-weight: 500;

### margin: 0;

### font-size: 34px;

### }

### body {

### background-color: #faf7f7;

### margin: 0;

### padding: 0;

### font-family: "Roboto", sans-serif;

### }

### */\* Container Styling \*/*

### .container-custom {

### max-width: 600px;

### margin: 50px auto;

### padding: 20px;

### border: 1px solid #ccc;

### border-radius: 8px;

### background-color: #f9f9f9;

### box-shadow: 5px 5px 15px rgba(0, 0, 0, 0.3);

### }

### .title {

### text-align: center;

### margin-bottom: 25px;

### color: #a492eb;

### font-family: "Poppins", sans-serif;

### font-weight: 600;

### }

### */\* Book List Styling \*/*

### .book {

### display: flex;

### justify-content: space-between;

### align-items: center;

### padding: 10px 0;

### font-weight: bold;

### }

### .book span {

### flex: 1;

### text-align: center;

### }

### .author, .issued-date {

### color: #555;

### font-weight: normal;

### }

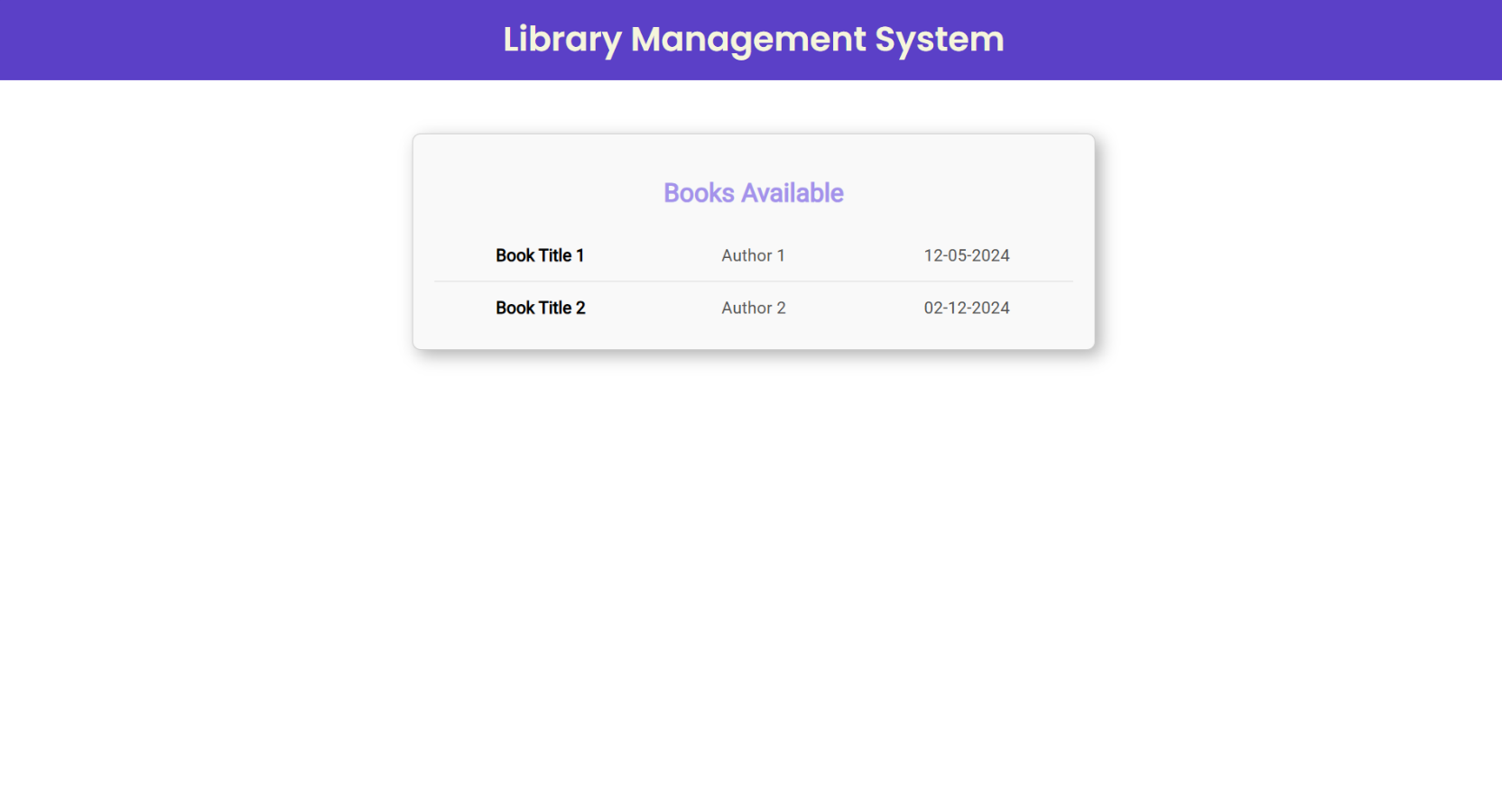
### hr {

### border: none;

### border-top: 1px solid #434242;

### margin: 5px 0;

### }

**OUTPUT:**

**CONCLUSION: In this assignment developed basic html page and demonstrated how to Implement, run the program.**

# Assignment No - 2

**PROBLEM STATEMENT:** “Create a user registration form for Library Management System in HTML and write a JavaScript Program to apply form validation and to get user registration details. Note: Add minimum 10 input form elements”

### OBJECTIVES:

* To design and develop registration form.
* To apply simple formatting styles.
* To implement and use Javascript programing.
* To validate each Form input elements using Javascript.

### SYSTEM REQUIREMENTS:

**Minimum Hardware Requirements**

MEMORY (RAM): 2GB

Hard Disk Capacity: 40GB

Processor Speed: 1.6 GHz or faster processor

**Minimum Software Requirements**

Operating System: Windows / Linux

Editor: Notepad/sublime/Visula Studio Code/ Atom

Browser: Chrome/ Microsoft Edge / Mozila Firefox / Safari /Opera

### THEORY:

**HTML Forms:**

Form Structure:

The <form> element serves as the container for various form elements and controls user input.

**Attributes include:**

**action**: Specifies the URL where the form data will be sent upon submission. method: Determines the HTTP method used for sending data (e.g., "post" or "get").

<form action="/submit" method="post">

<!-- Form elements go here -->

</form>

**Input Elements:**

**Text Input:**

Created using <input type="text">. Suitable for single-line text input. Example:

<label for="username">Username:</label>

<input type="text" id="username" name="username" required>

**Password Input:**

Created with <input type="password">. Conceals entered text for sensitive information. Example:

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

**Radio Buttons:**

Generated using <input type="radio">.

Allows users to select a single option from a group. Example:

<input type="radio" id="male" name="gender" value="male">

<label for="male">Male</label>

**Checkboxes:**

Created with <input type="checkbox">. Enables users to select multiple options. Example:

<input type="checkbox" id="subscribe" name="subscribe" value="yes">

<label for="subscribe">Subscribe to newsletter</label>

**Dropdown List:**

Utilizes <select> and <option> elements.

Provides a list of options for the user to choose from. Example:

<label for="country">Country:</label>

<select id="country" name="country">

<option value="usa">United States</option>

<option value="canada">Canada</option>

<option value="uk">United Kingdom</option>

</select>

**Form Validation:**

The *required* attribute ensures that a field must be filled out before submitting the form. JavaScript can be used for more complex validation as follows.

1. **Required Fields:**

JavaScript Operation:

var username = document.forms["myForm"]["username"].value; if (username == "") {

alert("Username must be filled out"); return false;

}

*Explanation:*

The document.forms property provides access to all the forms within the document. ["myForm"] is used to access a specific form with the name "myForm." ["username"].value extracts the value entered in the "username" input field. The if statement checks if the username is an empty string (""), and if so, it displays an alert and returns false to prevent form submission.

1. **Length Validation:**

JavaScript Operation:

var password = document.forms["myForm"]["password"].value; if (password.length < 8 || password.length > 20) {

alert("Password must be between 8 and 20 characters"); return false;

}

*Explanation:*

Similar to the first example, it retrieves the password value. The if statement checks if the password length is less than 8 or greater than 20 characters. If the condition is met, an alert is displayed, and false is returned to prevent form submission.

1. **Numeric Values:**

JavaScript Operation:

var age = document.forms["myForm"]["age"].value; if (isNaN(age)) {

alert("Age must be a number"); return false;

}

*Explanation:*

Retrieves the value entered in the "age" input field. The isNaN() function checks if the value is not a valid number. If the condition is true, an alert is displayed, and false is returned to prevent form submission.

1. **Email Validation:**

JavaScript Operation:

var email = document.forms["myForm"]["email"].value; var emailRegex = /\S+@\S+\.\S+/;

if (!emailRegex.test(email)) { alert("Invalid email address"); return false;

}

*Explanation:*

Retrieves the value entered in the "email" input field. /\S+@\S+\.\S+/ is a regular expression to validate email format. emailRegex.test(email) checks if the email value matches the regex pattern. If the condition is false, an alert is displayed, and false is returned to prevent form submission.

**HTML Tables:**

**Table Structure:**

The <table> element is employed to create tabular data structures.

Tables comprise three primary sections: <thead>, <tbody>, and optionally <tfoot>.

<table border="1">

<thead>

<!-- Table header content -->

</thead>

<tbody>

<!-- Table body content -->

</tbody>

<tfoot>

<!-- Table footer content (optional) -->

</tfoot>

</table>

**Table Headers and Cells:**

<th> elements inside <tr> tags in the <thead> section define table headers.

<td> elements within <tr> tags in the <tbody> section contain the actual data.

<thead>

<tr>

<th>Name</th>

<th>Age</th>

<th>City</th>

</tr>

</thead>

<tbody>

<tr>

<td>John Doe</td>

<td>25</td>

<td>New York</td>

</tr>

<!-- Additional rows go here -->

</tbody>

**Border and Styling:**

The border attribute in the <table> tag sets the border width.

CSS can be applied for additional styling, including background colors, borders, and spacing.

<table border="1" style="width:100%; border-collapse: collapse;">

<!-- Table content -->

</table> **CODE: (Sample Code) OUTPUT:**

**CONCLUSION: In this assignment developed basic html page and demonstrated how to Implement, run the program.**

# Assignment No - 3

**PROBLEM STATEMENT:** “Manipulate and design the user registration form using CSS3 selector with jQuery functions.”

### OBJECTIVES:

* To design and develop registration form.
* To apply simple formatting styles.
* To implement and use JQuery programing.
* To validate each Form input elements using jQuery.

### SYSTEM REQUIREMENTS:

**Minimum Hardware Requirements**

MEMORY (RAM): 2GB

Hard Disk Capacity: 40GB

Processor Speed: 1.6 GHz or faster processor

**Minimum Software Requirements**

Operating System: Windows / Linux

Editor: Notepad/sublime/Visula Studio Code/ Atom

Browser: Chrome/ Microsoft Edge / Mozila Firefox / Safari /Opera

### THEORY:

jQuery is a fast, small, and feature-rich JavaScript library. It simplifies HTML document traversal and manipulation, event handling, animation, and AJAX interactions. It was created by John Resig and released in 2006, aiming to address cross-browser compatibility issues and provide a concise syntax for common tasks.

**DOM Manipulation:**

jQuery simplifies DOM (Document Object Model) manipulation by providing a consistent and easy-to-use API. It allows developers to select elements, manipulate their content, attributes, and styles easily.

Example:

***$('#example').text('Hello, jQuery!');***

**Event Handling:**

jQuery provides a convenient way to handle events, such as clicks, keypresses, and mouse movements. Event handling is simplified, and it helps manage cross-browser inconsistencies.

Example:

***$('#myButton').on('click', function() { alert('Button clicked!');***

***});***

**AJAX (Asynchronous JavaScript and XML):**

jQuery simplifies asynchronous communication with the server using AJAX. It allows developers to make HTTP requests, handle responses, and update the DOM without a page reload.

Example:

// Make an AJAX request to retrieve data from the server

***$.ajax({***

***url: 'example.php', method: 'GET',***

***success: function(data) { console.log('Data received:', data);***

***}***

***});***

**Animations:**

jQuery provides a range of animation effects and methods to create visually appealing transitions on web pages. Animations can be applied to show, hide, or modify elements over time.

Example:

***// Hide an element with id 'myElement' with a fade out animation***

***$('#myElement').fadeOut('slow');***

**Chaining:**

jQuery allows method chaining, which means multiple methods can be called on the same set of elements in a single line. This results in concise and readable code.

Example:

***// Chain multiple jQuery methods on the same element***

***$('#myElement').css('color', 'red').addClass('highlight').fadeOut('slow');***

**Advantages of Using jQuery:**

**Cross-Browser Compatibility:**

jQuery abstracts away the differences in how various browsers handle JavaScript, providing a consistent API that works across different browsers.

**Simplified Syntax:**

jQuery offers a concise and easy-to-understand syntax, reducing the amount of code needed for common tasks compared to raw JavaScript.

**Rich Set of Plugins:**

jQuery has a vast ecosystem of plugins that extend its functionality. These plugins cover a wide range of features, from UI components to advanced animation effects.

**Community Support:**

jQuery has a large and active community, providing forums, tutorials, and documentation. This makes it easier for developers to find solutions to common issues and stay updated on best

practices.

**Fast Development:**

jQuery accelerates development by simplifying complex tasks and providing shortcuts for common operations, allowing developers to focus on building functionality rather than dealing with browser quirks.

**Form Validation using jQuery:**

**Required Field Validation:**

***$('#username').rules('add', { required: true,***

***messages: {***

***required: 'Please enter your username.'***

***}***

***});***

Explanation:

$('#username'): Selects the element with the ID username.

rules('add', { required: true, ... }): Adds a validation rule to the selected element, specifying that it is required.

messages: Allows you to customize the error message displayed if the validation fails.

**Length Validation:**

***$('#password').rules('add', { minlength: 8,***

***maxlength: 20, messages: {***

***minlength: 'Password must be at least 8 characters.', maxlength: 'Password cannot exceed 20 characters.'***

***}***

***});***

Explanation:

minlength and maxlength: Specify the minimum and maximum lengths, respectively. Custom error messages are provided for better user feedback.

**Numeric Values:**

***$('#age').rules('add', { number: true, messages: {***

***number: 'Please enter a valid numeric age.'***

***}***

***});***

number: true: Specifies that the input must be a number.

**Email Validation:**

***$('#email').rules('add', { email: true,***

***messages: {***

***email: 'Please enter a valid email address.'***

***}***

***});***

email: true: Validates that the input is a valid email address.

**Phone Number Validation:**

***$.validator.addMethod('phoneUS', function(phoneNumber, element) { phoneNumber = phoneNumber.replace(/\s+/g, '');***

***return this.optional(element) || phoneNumber.length === 10 && phoneNumber.match(/^\d{10}$/);***

***}, 'Please enter a valid phone number.');***

***$('#phone').rules('add', { phoneUS: true, messages: {***

***phoneUS: 'Please enter a valid phone number.'***

***}***

***});***

**Explanation:**

addMethod: Creates a custom validation method named phoneUS.

The phoneUS method checks if the phone number has exactly 10 digits.

**CODE: (Sample Code) OUTPUT: CONCLUSION:**

# Assignment No - 4

**PROBLEM STATEMENT:** “Create a CRUD MySQL Database Operations Application using AJAX.”

### OBJECTIVES:

* To design and develop am application to retrieve the data from database.
* To apply simple formatting styles.
* To implement and use AJAX programing.
* To develop Create, Read, Update, Delete operations on database.

### SYSTEM REQUIREMENTS:

**Minimum Hardware Requirements**

MEMORY (RAM): 2GB

Hard Disk Capacity: 40GB

Processor Speed: 1.6 GHz or faster processor **Minimum Software Requirements** Operating System: Windows / Linux

Editor: Notepad/sublime/Visula Studio Code/ Atom Server: XAMPP with PHP, MySQL

Browser: Chrome/ Microsoft Edge / Mozila Firefox / Safari /Opera

**THEORY:**

**MySQL:**

**Database:** A database is a structured collection of data organized in a way that a computer program can quickly select and retrieve specific pieces of data. In MySQL, a database is a container for tables.

**Table:** A table is a collection of related data organized in rows and columns. Each table has a name and consists of fields (columns) and records (rows).

**Column (Field):** Columns, also known as fields, represent the attributes of a table. Each column has a data type, such as INT (integer), VARCHAR (variable character), or DATE, specifying the kind of data it can store.

**Row (Record):** A row, also known as a record, represents a single set of related data in a table. Each row contains values for each column defined in the table.

**Primary Key:** A primary key is a column or a set of columns in a table that uniquely identifies each row in that table. It must contain unique values and cannot have NULL values.

**Foreign Key:** A foreign key is a column or a set of columns in a table that refers to the primary key in another table. It establishes a link between the two tables, creating a relationship.

**Index:** An index is a data structure that improves the speed of data retrieval operations on a database table. It is created on one or more columns of a table, and it allows the database engine to quickly locate and access the rows that match a particular value.

**SQL (Structured Query Language):** SQL is the language used to interact with relational databases. MySQL uses SQL for creating, updating, querying, and managing databases.

**Normalization:** Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. It involves dividing a database into tables and defining relationships between the tables.

**Transaction:** A transaction is a sequence of one or more SQL statements that are executed as a single unit of work. Transactions ensure the consistency and integrity of the database by either succeeding completely or failing completely.

**Stored Procedure:** A stored procedure is a set of SQL statements with a specific name, which is stored in the database and can be called and executed by other programs or procedures.

**Trigger:** A trigger is a set of instructions that are automatically executed or fired in response to certain events on a particular table or view in a database. Triggers are used to enforce business rules, perform complex calculations, or update other tables.

**Login to MySQL:** *mysql -u username –p*

This command is used to log in to the MySQL server. Replace "username" with your MySQL username. After entering this command, you will be prompted to enter your password.

**Show Databases:** SHOW DATABASES;

Displays a list of all available databases on the MySQL server.

**Create Database:** CREATE DATABASE database\_name; Creates a new database with the specified name.

**Use Database:** USE database\_name;

Switches to the specified database, making it the current working database for subsequent queries.

**Show Tables:** SHOW TABLES;

Displays a list of tables in the currently selected database.

**Describe Table:** DESCRIBE table\_name;

Provides information about the columns (fields) in a specific table, including data types and constraints.

**Create Table:**

CREATE TABLE table\_name ( column1 datatype, column2 datatype,..); Creates a new table with specified columns and data types.

**Insert Data:**

INSERT INTO table\_name (column1, column2, ...) VALUES (value1, value2, ...);

Inserts new records into a table. Specify the column names and corresponding values for each record.

**Select Data:**

SELECT column1, column2, ... FROM table\_name WHERE condition; Retrieves data from a table based on specified columns and conditions.

**Update Data:**

UPDATE table\_name SET column1 = value1, column2 = value2 WHERE condition; Modifies existing records in a table based on specified conditions.

**Delete Data:**

DELETE FROM table\_name WHERE condition;

Deletes records from a table based on specified conditions.

**Primary Key:**

CREATE TABLE table\_name ( id INT PRIMARY KEY, ... );

Defines a primary key for a table. The primary key uniquely identifies each record in the table.

**Foreign Key:**

CREATE TABLE table\_name ( id INT PRIMARY KEY, other\_table\_id INT, FOREIGN KEY (other\_table\_id) REFERENCES other\_table(id) );

Establishes a foreign key relationship between two tables. The foreign key references the primary key of another table.

**Create Index:**

CREATE INDEX index\_name ON table\_name (column1, column2, ...); Creates an index on specified columns, improving query performance.

**Transaction:**

START TRANSACTION;

-- SQL statements -- COMMIT;

Initiates a transaction. SQL statements within the transaction are executed as a single unit, and changes are only committed to the database if the entire transaction is successful.

**AJAX:**

AJAX (Asynchronous JavaScript and XML) is a set of web development techniques that allow web pages to send and receive data asynchronously (in the background) with the server. This enables the updating of parts of a web page without requiring a full page reload. While the name suggests XML, modern AJAX implementations often use JSON (JavaScript Object Notation) for data interchange due to its simplicity and efficiency.

Here are some key concepts and components related to AJAX:

1. **Asynchronous Request:**

AJAX allows web pages to make asynchronous requests to the server. This means that the

request is sent to the server, but the browser does not wait for the response before continuing to render the page.

1. **XMLHttpRequest Object:**

The `XMLHttpRequest` object is a fundamental part of AJAX. It is a browser API that provides functionality for making HTTP requests from the browser. While the name includes "XML," it can be used to handle various data formats, including XML, JSON, plain text, etc.

Example of creating an `XMLHttpRequest` object:

javascript

var xhr = new XMLHttpRequest();

1. **HTTP Methods:**

AJAX supports different HTTP methods, primarily `GET` and `POST`. The `GET` method is used for retrieving data from the server, while the `POST` method is used for sending data to the server.

1. **Event Handlers:**

`XMLHttpRequest` provides event handlers that can be used to define functions that will be executed at different stages of the AJAX request-response cycle. Common event handlers include `onreadystatechange`, `onload`, `onerror`, etc.

Example of using an event handler: javascript

xhr.onreadystatechange = function() {

if (xhr.readyState == 4 && xhr.status == 200) {

// Process the response console.log(xhr.responseText);

}

};

1. **AJAX with Fetch API:**

The Fetch API is a more modern and powerful alternative to `XMLHttpRequest`. It provides a more flexible and promise-based approach for making HTTP requests. It is widely used in modern web development.

Example using Fetch API:

javascript fetch('https://api.example.com/data')

.then(response => response.json())

.then(data => console.log(data))

.catch(error => console.error('Error:', error));

1. **Cross-Origin Resource Sharing (CORS):**

AJAX requests are subject to the same-origin policy, meaning they can only make requests to the same domain unless the server explicitly allows requests from other domains using CORS headers.

1. **AJAX Libraries:**

While vanilla JavaScript can be used for AJAX, several libraries simplify the process and provide additional features. Popular AJAX libraries include jQuery, Axios, and the Fetch API.

AJAX is a crucial technology for building dynamic and responsive web applications. It allows developers to create seamless user experiences by updating content on a page without requiring a full page reload.

**AJAX in jQuery: Theory with Example**

jQuery is a popular JavaScript library that simplifies many common tasks, including AJAX. jQuery provides a convenient set of methods to make AJAX requests and handle the responses. Here is a brief overview of AJAX in jQuery with an example:

**AJAX Functions in jQuery:**

1. **`$.ajax()` Function:**

The `$.ajax()` function is a versatile method for making AJAX requests in jQuery. It provides a wide range of options for customization.

Example:

javascript

$.ajax({

url: 'https://api.example.com/data', method: 'GET',

dataType: 'json', success: function(data) {

console.log('Success:', data);

},

error: function(error) { console.error('Error:', error);

}

});

In this example:

* `url`: The URL to send the request.
* `method`: The HTTP method (`GET` in this case).
* `dataType`: The expected data type of the response (`json` in this case).
* `success`: A callback function to handle a successful response.
* `error`: A callback function to handle errors.

1. **`$.get()` and `$.post()` Methods:**

jQuery provides shorthand methods for common HTTP methods: `$.get()` for `GET` requests and `$.post()` for `POST` requests.

Example (using `$.get()`):

javascript

$.get('https://api.example.com/data', function(data) { console.log('Success:', data);

});

Example (using `$.post()`):

javascript

$.post('https://api.example.com/save', { key: 'value' }, function(response) { console.log('Success:', response);

});

In these examples, the second parameter is a data object sent in the request.

1. **$.getJSON() Method:**

Specifically designed for making `GET` requests expecting JSON responses, `$.getJSON()` is a shorthand method.

Example:

javascript

$.getJSON('https://api.example.com/data', function(data) { console.log('Success:', data);

});

**Handling Responses:**

1. **Success Callback:**

The `success` callback is called when the AJAX request is successful. It typically takes the response data as its argument.

1. **Error Callback:**

The `error` callback is called when there is an error during the AJAX request. It takes the error information as its argument.

**AJAX Deferred Objects and Promises:**

1. **Deferred Object:**

The `$.ajax()` function returns a Deferred object, which provides methods like `.done()`, `.fail()`, and `.always()`.

Example:

javascript

$.ajax({

}).done(function(data) {

console.log('Done:', data);

}).fail(function(error) { console.error('Failed:', error);

}).always(function() { console.log('Always executed');

});

1. **Promises with `$.when()`:**

`$.when()` allows you to combine multiple Deferred objects into a single promise.

Example:

javascript

$.when(

$.ajax('/api/data1'),

$.ajax('/api/data2')

).done(function(result1, result2) { console.log('Success:', result1[0], result2[0]);

}).fail(function(error) { console.error('Failed:', error);

});

**CODE: (Sample Code) OUTPUT: CONCLUSION:**

# Assignment No - 5

**PROBLEM STATEMENT:** “Develop an application for a Library Management System using Angular js and MongoDB/MySQL. The application able to perform the Create, Delete, Modify the data in database.”

### OBJECTIVES:

* To understand and write the Basic angular js programming.
* To develop the Angular Application to connect with Databases.
* To perform the Database operations using Angular JS.
* To develop PHP script to connect databases.

### SYSTEM REQUIREMENTS:

**Minimum Hardware Requirements**

MEMORY (RAM): 2GB

Hard Disk Capacity: 40GB

Processor Speed: 1.6 GHz or faster processor

**Minimum Software Requirements**

Operating System: Windows / Linux

Editor: Notepad/sublime/Visula Studio Code/ Atom Server: XAMPP with PHP, MySQL, MongoDB

Browser: Chrome/ Microsoft Edge / Mozila Firefox / Safari /Opera

### THEORY:

1. **Angular JS Basics:**

**HTML Structure:**

The HTML structure is typical. However, there are a few AngularJS-specific attributes present: **ng-app="myApp":** This attribute defines the root element of the AngularJS application. myApp is the name of the AngularJS module that the application depends on.

**ng-controller="myController":** This attribute tells AngularJS which controller to use for a particular section of the HTML. Here, it's myController.

**Script Tags:** The script tags include the AngularJS library hosted on Google's CDN. This library is necessary for AngularJS to work.

**AngularJS Module:**

var app = angular.module('myApp', []);

This line defines an AngularJS module named myApp. Modules are containers for various parts of your application, such as controllers, services, directives, etc. The empty array [] is for dependencies. Here, there are no dependencies.

**Controller:**

app.controller('myController', function($scope) {

$scope.name = 'World';

});

This code defines a controller named myController and associates it with the myApp module. The controller function takes $scope as a parameter. $scope is an object that binds the view (HTML) with the controller. Here, we're attaching a variable name to the $scope object with the initial value 'World'.

**Data Binding:**

<input type="text" ng-model="name">

<p>Hello, {{name}}!</p>

**ng-model="name":** This directive binds the input field with the variable name defined in the controller's scope. So, any changes made in the input field will automatically update the name variable.

**{{name}}:** This is an expression that AngularJS evaluates. It displays the value of the name variable defined in the controller.

Example:

<!DOCTYPE html>

<html lang="en" ng-app="myApp">

<head>

<meta charset="UTF-8">

<title>AngularJS Example</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body>

<div ng-controller="myController">

<input type="text" ng-model="name">

<p>Hello, {{name}}!</p>

</div>

<script>

var app = angular.module('myApp', []); app.controller('myController', function($scope) {

$scope.name = 'World';

});

</script>

</body>

</html>

1. **Components required to Connect Database:**

**Set Up Back-End Server:**

* + Choose a back-end technology like Node.js, Python with Flask or Django, Java with

Spring Boot, etc.

* + Install necessary dependencies for your chosen technology. For example, if you're using Node.js with Express and MongoDB, you would install express and mongoose.
  + Create a server file (e.g., server.js for Node.js) to handle HTTP requests and database operations.
  + Connect to your database (e.g., MongoDB) using a connection string. You typically do this using a library like Mongoose for MongoDB.
  + Define routes to handle CRUD operations (Create, Read, Update, Delete) for your data.
  + Start the server and ensure it's listening for incoming requests.

**Create RESTful APIs:**

* + Define endpoints (URL paths) for each CRUD operation.
  + Handle incoming HTTP requests (POST, GET, PUT, DELETE) on these endpoints.
  + Write logic to interact with the database (e.g., saving data, fetching data, updating data, deleting data) within these route handlers.
  + Send appropriate HTTP responses (success or error) back to the client.

**AngularJS Front-End Setup:**

* + Create an AngularJS application module using angular.module.
  + Set up AngularJS controllers to handle different parts of your application logic.
  + Use AngularJS services like $http or $resource to make HTTP requests to your back-end server.
  + Define functions in your controllers to handle responses from the server and update the UI accordingly.

**Database Connection from AngularJS:**

* + Use AngularJS services like $http or $resource to make HTTP requests to your back-end server's endpoints.
  + Configure these services to send requests to the appropriate URL paths on your server.
  + Handle responses from the server within your AngularJS controllers or services.
  + Update the UI based on the data received from the server.

1. **HTTP Request:**

Handling incoming HTTP requests in a back-end server involves defining routes for different types of requests (GET, POST, PUT, DELETE), processing those requests, interacting with the database as needed, and sending appropriate responses back to the client. Let's break down each HTTP method:

**GET Requests:**

**Purpose:** GET requests are used to retrieve data from the server.

**Route Definition:** Define a route handler for the desired endpoint, typically using Express.js.

**Processing:** Inside the route handler, retrieve data from the database or another data source.

**Response:** Send the retrieved data back to the client as the HTTP response. Example (Express.js):

app.get('/api/products', async (req, res) => { try {

const products = await Product.find(); res.json(products);

} catch (error) {

console.error('Error fetching products:', error); res.status(500).send('Internal Server Error');

}

});

**POST Requests:**

**Purpose:** POST requests are used to send data to the server to create new resources.

**Route Definition:** Define a route handler for the POST endpoint.

**Processing:** Extract the data from the request body and use it to create a new resource in the database.

**Response:** Send back a success response along with any relevant data about the newly created resource.

Example (Express.js):

app.post('/api/products', async (req, res) => { try {

const newProduct = await Product.create(req.body); res.status(201).json(newProduct);

} catch (error) {

console.error('Error creating product:', error); res.status(500).send('Internal Server Error');

}

});

**PUT Requests:**

**Purpose:** PUT requests are used to update existing resources on the server.

**Route Definition:** Define a route handler for the PUT endpoint.

**Processing:** Extract the updated data from the request body and use it to update the corresponding resource in the database.

**Response:** Send back a success response along with any relevant data about the updated resource.

Example (Express.js):

app.put('/api/products/:id', async (req, res) => { try {

const updatedProduct = await Product.findByIdAndUpdate(req.params.id, req.body,

{ new: true });

res.json(updatedProduct);

} catch (error) {

console.error('Error updating product:', error); res.status(500).send('Internal Server Error');

}

});

**DELETE Requests:**

**Purpose:** DELETE requests are used to delete existing resources from the server.

**Route Definition:** Define a route handler for the DELETE endpoint.

**Processing:** Extract the identifier of the resource to be deleted from the request parameters and delete it from the database.

**Response:** Send back a success response indicating that the resource has been deleted. Example (Express.js):

app.delete('/api/products/:id', async (req, res) => { try {

await Product.findByIdAndDelete(req.params.id); res.status(204).send();

} catch (error) {

console.error('Error deleting product:', error); res.status(500).send('Internal Server Error');

}

});

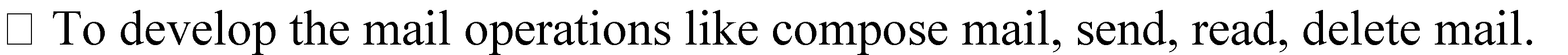
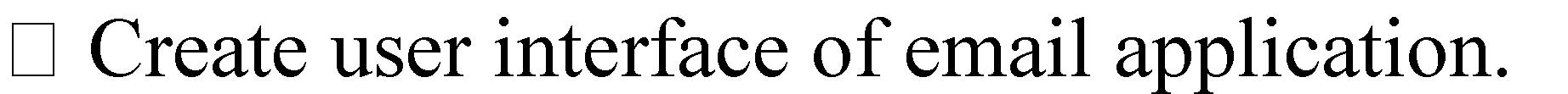
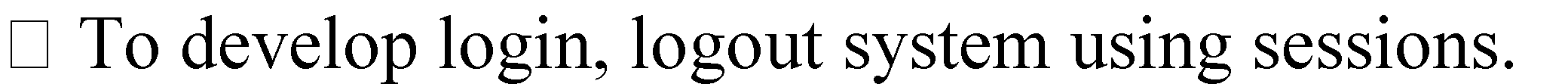
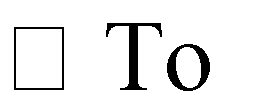
### CODE:

**Output:**

**Conclusion:**

# Assignment No - 6

PROBLEM STATEMENT: “Develop a Mail applications using PHP and SQL.” OBJECTIVES:



design and develop an application to retrieve the data from database.

### SYSTEM REQUIREMENTS:

Minimum Hardware Requirements MEMORY (RAM): 2GB

Hard Disk Capacity: 40GB

Processor Speed: 1.6 GHz or faster processor Minimum Software Requirements

Operating System: Windows / Linux

Editor: Notepad/sublime/Visula Studio Code/ Atom

Server: XAMPP with PHP, MySQL, Google SMTP server/ Any mail server Browser: Chrome/ Microsoft Edge / Mozila Firefox / Safari /Opera

### THEORY:

1. PHPMailer Package (to compose and send a new mail):
   1. Initialization:

When you initialize PHPMailer, you're essentially creating an instance of the PHPMailer class, which allows you to configure and send emails.

Syntax:

$mail = new PHPMailer(true);

PHPMailer(true): This instantiates a new PHPMailer object. Passing true enables exceptions, meaning PHPMailer will throw exceptions if there's an error, allowing you to catch and handle them.

* 1. Setting up SMTP:

SMTP (Simple Mail Transfer Protocol) is the standard protocol for sending emails. PHPMailer allows you to configure SMTP settings to send emails through an SMTP server.

Basic Syntax:

$mail->isSMTP();

$mail->Host = 'smtp.example.com';

$mail->SMTPAuth = true;

$mail->Username = ['your\_email@example.com';](mailto:%27your_email@example.com)

$mail->Password = 'your\_password';

$mail->SMTPSecure = 'tls';

$mail->Port = 587;

isSMTP(): This tells PHPMailer to use SMTP.

Host: Specifies the SMTP server hostname or IP address. SMTPAuth: Indicates whether SMTP authentication is required. Username and Password: Your SMTP server login credentials. SMTPSecure: Specifies the encryption method (tls or ssl).

Port: The port number to connect to the SMTP server.

* 1. Setting Email Parameters:

These parameters include sender, recipient, subject, and message body. Syntax:

$mail->[setFrom('your\_email@example.com',](mailto:setFrom(%27your_email@example.com) 'Your Name');

$mail->[addAddress('recipient@example.com',](mailto:addAddress(%27recipient@example.com) 'Recipient Name');

$mail->[addReplyTo('replyto@example.com',](mailto:addReplyTo(%27replyto@example.com) 'Reply To Name');

$mail->isHTML(true);

$mail->Subject = 'Subject of the Email';

$mail->Body = 'HTML message body';

$mail->AltBody = 'Plain text message body if HTML is not supported';

Sender: The setFrom() method sets the sender's email address and name. It's essential to provide accurate sender information to ensure proper email delivery and avoid being marked as spam.

Recipient: Use the addAddress() method to specify the email address of the recipient. You can add multiple recipients by calling this method multiple times.

Reply-To: The addReplyTo() method allows you to set the reply-to address. When the recipient replies to the email, their message will be directed to this address.

Format: By calling isHTML(true), you indicate that the email body is formatted as HTML. This enables you to create visually appealing emails with HTML markup.

Subject and Body: Set the subject and body of the email using the Subject and Body properties. You can provide both HTML and plain text versions of the message body using the AltBody property.

* 1. **Adding Attachments:**

**You can attach files to the email using the addAttachment() method. Syntax:**

**$mail->addAttachment('/path/to/file.pdf', 'optional\_filename.pdf'); addAttachment(): Adds an attachment to the email. The first parameter is the path to the file, and the second (optional) parameter is the name of the attachment.**

* 1. **Sending the Email:**

**Once you've configured the email, you can send it using the send() method. Syntax and Basic Logic:**

**if (!$mail->send()) {**

**echo 'Message could not be sent.';**

**echo 'Mailer Error: ' . $mail->ErrorInfo;**

**} else {**

**echo 'Message has been sent';**

**}**

**send(): Sends the email. Returns true on success and false on failure. ErrorInfo: Provides detailed error information if the email sending fails.**

* 1. **Additional Configuration Options:**

**SMTP Debugging: Enable verbose debug output to troubleshoot SMTP-related issues.**

**Custom Headers: Add custom headers to the email.**

**Using Sendmail: Use the server's Sendmail program to send emails. Using mail() Function: Use PHP's built-in mail() function to send emails.**

1. **Fetch Inbox Mails:**
   1. **Enable IMAP Extension:**

**Ensure that the IMAP extension is enabled in your PHP configuration. You can enable it by uncommenting or adding the following line in your php.ini file:**

**extension=imap**

* 1. **Connect to the Mail Server:**

Use the imap\_open() function to establish a connection to the mail server. You need to provide the server address, username, and password.

**Syntax:**

**$server = '{mail.example.com:993/imap/ssl}INBOX';**

**$username =** [**'your\_email@example.com';**](mailto:%27your_email@example.com)

**$password = 'your\_password';**

**$mailbox = imap\_open($server, $username, $password); if (!$mailbox) {**

**die('Could not connect to the mail server.');**

**}**

* 1. **Fetch Inbox Emails:**

**Once connected, you can fetch emails from the inbox using the imap\_search() and imap\_fetch\_overview() functions.**

**Syntax with Basic Code:**

**$mails = imap\_search($mailbox, 'ALL'); if (!$mails) {**

**die('No emails found in the inbox.');**

**}**

**foreach ($mails as $mailId) {**

**$overview = imap\_fetch\_overview($mailbox, $mailId, 0);**

**$subject = $overview[0]->subject;**

**$from = $overview[0]->from;**

**$date = $overview[0]->date; echo "Subject: $subject<br>"; echo "From: $from<br>"; echo "Date: $date<br><br>";**

**}**

* 1. **Read Email Content:**

**To read the content of a specific email, you can use the imap\_fetchbody() function. Syntax with basic code:**

**$emailNumber = 1; // Example: reading the first email**

**$emailBody = imap\_fetchbody($mailbox, $emailNumber, 1);**

**// Decode the email body if it's encoded**

**if ($overview[0]->encoding == 3) { // 3 is base64 encoding**

**$emailBody = base64\_decode($emailBody);**

} elseif ($overview[0]->encoding == 4) { // 4 is quoted-printable encoding

**$emailBody = quoted\_printable\_decode($emailBody);**

**}**

**echo "Email Body:<br>$emailBody";**

* 1. **Close the Connection:**

**After fetching and reading emails, it's important to close the connection to the mail server using the imap\_close() function.**

**Syntax:**

**imap\_close($mailbox);**

1. **Session Management:**

**A session is a mechanism to preserve data across subsequent HTTP requests. When**

**a user visits a website, the server assigns a unique identifier known as a session ID to that user. This session ID is typically stored in a cookie on the client side, although it can also be appended to URLs. The server then uses this session ID to associate the user's data with their session.**

* 1. **Starting a Session:**

**In PHP, you start a session using the session\_start() function. This function initializes or resumes a session, making session variables available for use. It's usually called at the beginning of every PHP script that needs to access session data. Syntax:**

**session\_start();**

* 1. **Storing Data in a Session:**

**Once a session is started, you can store data in the $\_SESSION superglobal array. This array behaves like a regular PHP associative array, allowing you to store key- value pairs of data.**

**Syntax:**

**$\_SESSION['username'] = 'john\_doe';**

**$\_SESSION['user\_id'] = 123;**

* 1. **Retrieving Data from a Session:**

**You can retrieve session data by accessing elements of the $\_SESSION array. Syntax:**

**$username = $\_SESSION['username'];**

**$user\_id = $\_SESSION['user\_id'];**

* 1. **Destroying a Session:**

To end a session and clear all session data, you can use the session\_destroy() function. This **function deletes the session data on the server and removes the session cookie from the client.**

**Syntax:**

**session\_destroy();**

* 1. **Session Configuration:**

**PHP provides various configuration options for session management, which can be set in the php.ini file or using the ini\_set() function. Some common session configuration options include:**

**Session Lifetime: Determines how long a session remains active before it's considered expired.**

**Session Cookie Parameters: Configures the behavior of the session cookie, such as its name, path, domain, and security settings.**

**Session Save Path: Specifies the directory where session data is stored on the server.**

* 1. **Session Security:**

**Session security is critical to prevent unauthorized access and data tampering. Some best practices for session security include:**

**Using HTTPS: Encrypting communication between the client and server with HTTPS ensures that session data is transmitted securely.**

**Regenerating Session IDs: Regenerate session IDs periodically or after a user logs in to prevent session fixation attacks.**

**Validating Session Data: Always validate session data on the server side to prevent injection attacks and data manipulation.**

Securing Session Storage: Ensure that session data stored on the server is inaccessible to unauthorized users.

**Code:**

**Output:**

**Conclusion:**

Mini Project Report on

# “Title of Your Mini Project”

Submitted to

Vishwakarma Institute of Information Technology, Pune

(An Autonomous institute Affiliated to Savitribai Phule Pune University) In partial fulfillment of requirements for

**We Technology (CA22233 ) Term Work**

By

## Student Name

**(GR NO: )**

Under the guidance of

**Prof. P. Mehta**

****

**Department of Computer Science and Engineering (AI)**

**Academic Year: 2024-25 Sem I**

**Vishwakarma Institute of Information Technology, Pune –37**

(An Autonomous Institute Affiliated to Savitribai Phule Pune University)

## ACKNOWLEDGEMENT

I wish to express my gratitude to Prof**. Dr. Vivek Deshpande Director, VIIT Pune,** for providing the facilities of the Institute and for his encouragement during this work. I also express my deep gratitude to **Prof. (Dr.) Nilesh Sable**, the Head of the Department of Computer Science and Engineering (AI), VIIT Pune for his guidance and support.

I would also 1ike to gratefully acknowledge the enthusiastic supervision of my mini project guide**, Prof. Pradnya Mehta** for his continuous, valuable guidance, patience, constant care, and kind encouragement throughout the mini project that made me present this mini project report in an efficient manner.

Final1y, I wish to thank my family members and my friends who have always been very supportive and encouraging.

**Student Names (DATE: )**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Chapter No** | **Table of contents** | **Pg. no** |
| 1 | Abstract |  |
| 2 | Introduction |  |
| 3 | System Requirements |  |
| 4 | Project Problem Statement |  |
| 5 | Our Modules |  |
| 6 | Technology Stack |  |
| 7 | Sample Code |  |
| 8 | Output Screenshots |  |
| 9 | Conclusion |  |
| 10 | References |  |

**Abstract**

**Introduction**

**System Requirements**

**Project Problem Statement**

**Our Modules**

**Technology Stack**

**Sample Code**

**Output Screenshots**

**Conclusion**

**Chapter 10 References**