

**MIT Art, Design and Technology University**

**MIT School of Computing, Pune**

**Department of Information Technology**

|  |
| --- |
| **Lab Manual** |

# **Practical - Web Programming**

# **Class - S.Y. (SEM-II), DA**

# **Batch - DA-I**

# **Name of the Student**

# **LOSHETTY HARSHITHA RAMESH**

ADT23SOCB1529

**A.Y. 2024 – 2025 (SEM-II)**

File Index page given in the stationary

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Web Programming**  **SEMESTER – IV** | | | | | |
| **Course Code:** | | 23IT2008 | **Course Credits:** | 02 | |
| **Teaching Hours / Week (L:T:P):** | | 0:0:4 | **CA Marks:** | 25 | |
| **Total Number of Teaching Hours:** | |  | **END-SEM Marks:** | 25 | |
| **Course Pre-requisites:** | | | | | |
| **Course Description:**  This course provides a comprehensive introduction to web technology, designed to help students develop a strong foundation in building and managing websites and web applications. The curriculum covers key topics such as HTML, CSS, and JavaScript,PHP, MySQL, which are essential for creating interactive, well-designed web pages. Students will also explore the principles of responsive design, ensuring that web applications are optimized for different devices and screen sizes.  The course dives deeper into server-side technologies, including HTTP, web servers, and databases, allowing students to understand how websites function behind the scenes. Emphasis is placed on practical learning, and students will gain hands-on experience by working on projects that showcase their ability to design, develop, and deploy websites.  By the end of the course, students will be proficient in using modern web technologies to create web applications. They will understand how to handle client-server interactions, manage user data, and implement various web technologies to enhance the functionality of their applications. | | | | | |
| **Course Learning Objectives:** This course will enable the students to:   1. Understand fundamental concepts of front-end web development. 2. Enable students to create basic web pages incorporating essential elements such as images, hyperlinks, lists, tables, and forms. 3. Teach students how to use CSS to manage fonts, lists, colors, text alignment, and background images for a cohesive and aesthetically pleasing web design. 4. Develop an understanding of JavaScript scopes to manage the visibility and lifetime of variables and functions effectively. 5. Equip students with the skills to implement and handle JavaScript events, enabling enhanced user interactions through event-driven programming. 6. Apply comprehensive knowledge of HTML, CSS, and JavaScript to develop a complete front-end application. Utilize project-based learning to showcase problem-solving skills and creativity in web development projects. 7. Configure server environments with Apache/TOMCAT. 8. Set up a PHP development environment and write basic PHP scripts. 9. Master PHP programming constructs for web development tasks. 10. Create and process HTML forms, and manage MySQL database operations. 11. Develop comprehensive back-end applications using PHP and MySQL. | | | | | |
| **Course Outcome:** After taking this course, Students will be able to :   1. Apply knowledge of HTML to create the structure of the webpage and CSS to style and layout the elements, making the application visually appealing. 2. Apply comprehensive knowledge of HTML, CSS, and JavaScript to develop a complete front-end application and utilize project-based learning to showcase problem-solving skills and creativity in web development projects. 3. Set up and configure a server environment using tools like Apache or TOMCAT and set up a PHP development environment. Write & execute simple PHP scripts, understanding PHP syntax and basic features, create HTML forms to collect user data and integrate with PHP for processing. 4. Design and develop a back-end application using PHP and MySQL, implementing CRUD operations to manage data effectively. | | | | | |
| **UNIT – I** | **Introduction to HTML and Cascading Style Sheet** | | | | **09 Hours** |
| Module 1 - Markup Language (HTML): Introduction to HTML, Formatting and Fonts, Commenting Code, Anchors, Backgrounds, Images, Hyperlinks, Lists, Tables, Frames, HTML Forms  Module 2 - CSS: Need for CSS, introduction to CSS, basic syntax and structure, Levels of style sheets, Style specification formats, BOX Model, Selector forms, Property value forms, Font properties, List properties, Color, Alignment of text, Background images | | | | | |
| **Pedagogy** | **ICT Teaching / PowerPoint Presentation and Videos:**  **Use tools like Visual Studio Code (free).**  **Videos:**  [**https://www.coursera.org/learn/html-css-javascript-for-web-developers**](https://www.coursera.org/learn/html-css-javascript-for-web-developers) | | | | |
| **Self-study / Do it yourself /:**  **Practice creating basic HTML pages and enhancing them using CSS.** | | | | |
| **Experiential Learning Topics:**  **Design a simple webpage for coffee shop website** | | | | |
| **PBL - Project Based Learning:**  **Create a multi-page website (e.g., coffee shop website) using HTML and CSS.** | | | | |
|  | | | | | |
| **UNIT – II** | **Front-End Development** | | | | **09 Hours** |
| Module 3 - Overview of JavaScript, including JS in an HTML (Embedded, External), Basic JS syntax, basic interaction with HTML  Module 4 - Core features of JavaScript: Data types, Control Structures, Arrays, Functions and Scopes | | | | | |
| **Pedagogy** | **ICT Teaching / PowerPoint Presentation and Videos:**  **Use tools like Visual Studio Code (free).**  **Videos:**  [**https://www.coursera.org/learn/javascript-basics**](https://www.coursera.org/learn/javascript-basics) | | | | |
| **Self-study / Do it yourself /:**  **Solve exercises on JavaScript syntax, control structures, and functions** | | | | |
| **Experiential Learning Topics:**  **Build a web page with interactive elements (e.g., a simple calculator).** | | | | |
| **PBL - Project Based Learning:**  **Develop an interactive webpage that uses JavaScript to validate form inputs or perform basic calculations.** | | | | |
|  | | | | | |
| **UNIT – III** | **Advanced Front-End Development** | | | | **09 Hours** |
| Module 5 - DOM: DOM levels, DOM Objects and their properties and methods, Manipulating DOM  Module 6 - JavaScript Events: JavaScript Events, Types of JavaScript Events, Objects in JS, Event Handling | | | | | |
| **Pedagogy** | **ICT Teaching / PowerPoint Presentation and Videos:**  [**https://www.coursera.org/learn/building-interactive-web-pages-using-javascript**](https://www.coursera.org/learn/building-interactive-web-pages-using-javascript)  **Use tools like Visual Studio Code (free).** | | | | |
| **Self-study / Do it yourself /:**  **Practice exercises on DOM traversal and event handling.** | | | | |
| **Experiential Learning Topics:**  **Add dynamic behavior to a webpage using DOM and events (e.g., a to-do list app).** | | | | |
| **PBL - Project Based Learning:**  **Develop a web page with dynamic content (e.g., a task manager or interactive quiz) using DOM manipulation and event handling.** | | | | |
|  | | | | | |
| **UNIT – IV** | **Server Side Scripting** | | | | **09 Hours** |
| Module 7 - Set up and configure a server environment using tools like Apache or TOMCAT, set up a PHP development environment.  Module 8 -Introduction to PHP: : Introduction to PHP, Server side scripting Vs Client side scripting, Basic Development Concepts (Mixing PHP with HTML), Creating, Writing & Running First PHP Script, PHP syntax, conditions & Loops, Functions, String manipulation, Arrays & Functions,  Module 9 - Form handling with HTML and PHP: Designing of Forms using HTML, Form Handling using GET and POST methods of Form | | | | | |
| **Pedagogy** | **ICT Teaching / PowerPoint Presentation and Videos:**  [**https://www.coursera.org/learn/web-applications-php**](https://www.coursera.org/learn/web-applications-php)  **Use tools like Visual Studio Code (free), XAMPP/WAMP for PHP server setup, and MySQL Workbench for database management** | | | | |
| **Self-study / Do it yourself /:**  **Practice exercises on form handling and server-side scripting with PHP.** | | | | |
| **Experiential Learning Topics:**  **Create a basic form for data submission and handle it using PHP (e.g., feedback form).** | | | | |
| **PBL - Project Based Learning:**  **Develop a small server-side application (e.g., a contact form with email validation and submission).** | | | | |
|  | | | | | |
| **UNIT – V** | **Working with Databases and Web Application Development** | | | | **09 Hours** |
| Module 10 - Working with databases using MySQL with PHP: MySQL database, create database, create table, primary key with AUTO\_INCREMENT setting, Insert Data Into a Database Table, Select Data From a Database Table, Open or close a Connection to the MySQL Server.  Module 11 - Web Application Development (Project): Develop the web application to handle client-server interactions, manage user data, and implement various web technologies to enhance the functionality of their applications. Example: Website for a Coffee Shop | | | | | |
| **Pedagogy** | **ICT Teaching / PowerPoint Presentation and Videos:**  **Use tools like Visual Studio Code (free), XAMPP/WAMP for PHP server setup, and MySQL Workbench for database management**  **Videos:**  [**https://www.coursera.org/learn/web-app**](https://www.coursera.org/learn/web-app) | | | | |
| **Self-study / Do it yourself /:**  **Exercises on creating and manipulating databases using PHP and MySQL.** | | | | |
| **Experiential Learning Topics:**  **Create a database and design a webpage to display its data dynamically.** | | | | |
| **PBL - Project Based Learning:**  **Develop a fully functional web application (e.g., a Coffee Shop website or e-commerce platform) that integrates database functionality for data management.** | | | | |

**Text Books:**

1. "HTML and CSS: Design and Build Websites" by Jon Duckett.
2. "Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics" by Jennifer Niederst Robbins.
3. Achyut Godbole & Atul Kahate, ‖Web Technologies: TCP/IP to Internet Application Architectures‖, McGraw Hill Education publications, ISBN, 007047298X, 9780070472983.
4. Ralph Moseley & M. T. Savaliya, ―Developing Web Applications‖, Wiley publications, ISBN 13 : 9788126538676.

**Reference Books:**

1. Eloquent JavaScript: A Modern Introduction to Programming by Marijn Haverbeke.
2. JavaScript: The Good Parts by Douglas Crockford.
3. CSS Secrets: Better Solutions to Everyday Web Design Problems by Lea Ver.
4. Web Technologies- Jeffery C. Jackson, ISBN 978-81-317-1715-8 Pearson 2015.
5. PHP Objects, Patterns, and Practice by Matt Zandstra
6. MySQL Cookbook by Paul DuBois.
7. Advanced PHP Programming - George Schlossnagle- ISBN 0-672-32561-6,2004.

**URLs (Optional) - List of Online Courses**

1. W3Schools HTML, CSS, JavaScript Tutorial: <https://www.w3schools.com/html/>
2. Mozilla Developer Network (MDN) Web Docs - HTML, CSS, JavaScript, DOM: <https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML>
3. Project-Based Learning Resources:https://developer.mozilla.org/en-US/docs/Learn

**Contents beyond Syllabus:**

1. Web Essentials
2. Using JavaScript to handle form submission and login events (e.g., onsubmit, onclick)
3. JavaScript Form validations, General Input Validation, Password Validation
4. Storing user data (like a username) temporarily using localStorage or sessionStorage
5. Dynamically updating the content of the webpage, such as displaying a welcome message
6. Redirecting users using window.location

**List of Experiments:**

In this series of assignments, you will create a coffee shop / any other website step by step. Each assignment will focus on a different aspect of the website, covering various HTML elements, CSS, JavaScript, PHP and MySQL concepts.

**Laboratory/Project Assignment Guidelines:**

1. Project Selection:
   * Each student must select a unique project topic for their laboratory assignments.
   * The chosen project topic should align with the concepts covered in the course syllabus.
   * The chosen project topic should be approved by the course coordinator/ subject teacher.
   * Students have the freedom to choose their project topics based on their interests and career aspirations.
   * Project topics may include but are not limited to:
     + E-commerce website
     + Blogging platform
     + Online booking system
     + Content management system (CMS)
     + Discussion forum
     + Social networking platform
     + Task management application
     + Portfolio website
2. Laboratory Assignments:
   * Throughout the course, students will complete laboratory assignments related to their chosen project topic.
3. Evaluation Criteria:
   * The laboratory assignments and the final project will be evaluated based on criteria such as Structure and Semantics, Content Organization, Forms and Inputs, Links and Navigation, Styling and Layout, Design Consistency, Functionality, Code Quality and adherence to project requirements.
   * Students are expected to demonstrate creativity, and a comprehensive understanding of web development principles in their projects.
   * The laboratory assignments based on chosen project topics will be assessed based on several key criteria that reflect both technical proficiency and creative application in web development. These include:

* Structure & Semantics: Proper use of HTML to create a logical, accessible structure with meaningful and semantically correct elements.
* Content Organization: Clear and intuitive organization of content, ensuring ease of navigation and logical flow throughout the site.
* Forms & User Input: Effective implementation of forms and user input elements that are functional, validated, and accessible.
* Links & Navigation: Well-structured navigation and functional links that provide a seamless user experience.
* Styling & Layout: Visually appealing and responsive design, with a well-executed layout that adapts to various screen sizes.
* Design Consistency: Uniformity in design elements, including colors, typography, and spacing, to maintain a cohesive look and feel across the site.
* Functionality: Full functionality of all interactive elements, ensuring a bug-free, smooth experience for users.
* Code Quality & Best Practices: Clean, well-organized, and efficient code that adheres to modern web development best practices and is easy to maintain.

1. Submission and Presentation:
   * The project and project report/journal must be submitted within the specified deadline and should meet the specified requirements outlined by the course coordinator/ subject teacher.

Project Problem Statement-

Design and develop a basic website for a local coffee shop using HTML, CSS, JavaScript, PHP and MySQL. This website will serve as an online presence for the coffee shop, effectively communicating the brand identity, showcasing the menu, providing essential information, and allowing customers to easily get in touch or locate the shop.

The project directory is as follows:

coffee-shop-website/

├── css/

│ └── styles.css

├── html/

│ ├── index.html

│ ├── menu.html

│ ├── about.html

│ └── contact.html

├── js/

│ └── scripts.js

└── images/

|  |  |
| --- | --- |
| 1. | Create the basic structure of the coffee shop website, including the home page layout with a header, main content area, and footer.  Prepare a common project website design and plan document for all assignments. Consider following points:   1. Brief information about the project. 2. Set the goals & deliverables. 3. Finalize the modules of the project. 4. Define the audience. 5. Describe pain points & the ideal experience (On the basis of existing systems) 6. Set the visual direction 7. Map out the Project structure. 8. Plan the content for each page. 9. Add ideas for content, images & layout. 10. Determine your site structure or Create content for your core website pages:     1. Home page     2. About page     3. Product/Service page     4. Testimonial/review page     5. Contact page     6. Starter blog posts 11. Create and collect design elements   These design elements define your brand personality and help customers feel what your brand represents through the use of:   * 1. Colors   2. Fonts and typography   3. Logos   4. Images and photos |
| 2. | HTML   1. Create a detailed home page for the coffee shop website. 2. Create a detailed menu/product page for the coffee shop website, listing all available items categorized appropriately. 3. Create a cart page that allows customers to review and manage the items they wish to purchase before proceeding to checkout. 4. Create an about us page that provides detailed information about the coffee shop’s history, mission, and team. 5. Create a contact page that allows customers to easily get in touch with the coffee shop through a form. 6. Design and implement admin/user registration form for the coffee shop website. 7. Design and implement admin/user login form for the coffee shop website. |
| 3. | CSS   1. Enhance the layout of the coffee shop website using CSS Grid for the home page. 2. Use CSS Grid to layout the menu/product items in a structured and style the menu categories with appropriate headings, spacing, separators, images, descriptions, and prices. |
| 4. | CSS   1. Enhance the cart page to make it user-friendly and visually appealing. Style the cart items with appropriate margins, paddings, and input field styles to provide a seamless shopping experience. 2. Enhance and style the about us page with appropriate margins, paddings, and input field styles. 3. Enhance and style the contact page to make it user-friendly and visually appealing. Style the contact form with appropriate margins, paddings, and input field styles. 4. Enhance and style the admin/user registration form with appropriate margins, paddings, and input field styles. 5. Enhance and style the admin/user login form with appropriate margins, paddings, and input field styles. |
| 5. | JavaScript   1. Implement user registration and login forms for the coffee shop website. These forms will allow users to create an account, log in, and access personalized features, such as saving favorite items or viewing order history.   User Registration Form will allow new customers to sign up and create an account on the website. The form will capture basic user details, including the name, email address, and password (not limited to these fields).  User Login Form will allow registered users to log into their accounts. The form will require an email address and a password to authenticate the user.   1. Provide validations for user registration and login forms to validate the input to ensure that all required fields are filled and that the email format is valid. (**Contents beyond Syllabus)** 2. Develop cart functionality to allow users to add items, update quantities, and remove items. |
| 6. | JavaScript   1. The user login form will allow registered users to log into their accounts. The form will require an email address and a password to authenticate the user. 2. If the login is successful, the user should be redirected to the homepage or their user dashboard. (**Contents beyond Syllabus)** 3. Use localStorage or sessionStorage to store authentication data, such as the user’s email and login status. This ensures that once a user is logged in, they remain authenticated even after the page reloads or when they visit the site again. (**Contents beyond Syllabus)** 4. Save the cart data to local storage when items are added, updated, or removed. Retrieve and load the cart data from local storage when the page loads. (**Contents beyond Syllabus)** |
| 7. | PHP   1. Develop a PHP script to handle user registration for the Coffee Shop website. The script should accept input from users for their name, email address, password, etc. (all required fields for registration). 2. Implement error handling to notify users of any issues during registration, such as validation errors. 3. Provide feedback to the user upon successful registration, either through a confirmation message or a redirect to a login page. |
| 8. | PHP   1. Develop a PHP script to handle user login for the Coffee Shop website. The script should accept input from users for their login credentials. (all required fields for login). 2. Provide feedback to the user upon successful login, either through a confirmation message or a redirect to a welcome page. 3. Implement error handling to notify users of login failures due to incorrect credentials or other errors. 4. Provide feedback to the user upon successful login, either through a welcome user name message or a redirect to a home page. |
| 9. | PHP and MySQL   1. Develop a PHP script that allows users to manage their shopping cart for an e-commerce website (e.g., a Coffee Shop store). The script should allow users to add items to their cart, view their cart contents, and remove items if needed. 2. Develop a PHP script to manage the shopping cart for an e-commerce website (e.g., a Coffee Shop store) using MySQL. This script should allow users to add items to their cart, view their cart contents, and remove items from the cart. The cart data should be stored in the MySQL database to allow persistence across sessions. |
| 10. | PHP and MySQL   1. Develop a PHP script to handle the checkout process for users who are ready to complete their purchase. The script should process the cart data and provide feedback to the user upon successful or failed checkout. 2. Develop a PHP script that processes the checkout process for users who are ready to complete their purchase, integrating the MySQL database for handling user and order information. The script should validate user input, process the cart data, and provide feedback upon successful or failed checkout. |

# 

## **Experiment No.1**

## 

## **Problem Statement:**

## **Weather Wise is an app designed to provide users with real-time weather data, forecasts, and personalized notifications based on their location and preferences. Many weather apps offer basic information, but Weather Wise will differentiate itself by offering a comprehensive, user-friendly interface, personalized settings, and advanced data, such as hourly forecasts, weather maps, and long-term trends.**

|  |  |
| --- | --- |
| 1. | Create the basic structure of the Weather wise website, including the home page layout with a header, main content area, and footer.  Prepare a common project website design and plan document for all assignments. Consider following points:   1. Brief information about the project. 2. Set the goals & deliverables. 3. Finalize the modules of the project. 4. Define the audience. 5. Describe pain points & the ideal experience (On the basis of existing systems) 6. Set the visual direction 7. Map out the Project structure. 8. Plan the content for each page. 9. Add ideas for content, images & layout. 10. Determine your site structure or Create content for your core website pages:     1. Home page     2. About page     3. Product/Service page     4. Testimonial/review page     5. Contact page     6. Starter blog posts 11. Create and collect design elements   These design elements define your brand personality and help customers feel what your brand represents through the use of:   * 1. Colors   2. Fonts and typography   3. Logos   4. Images and photos |

## **Objective:**

## To design the basic structure of the Weather Wise web app by planning its layout, content, and visual elements, ensuring it meets user needs for easy access to weather updates, interactive features, and a personalized experience**.**

## **Theory:**

### Project Design and Plan Document for Weather Wise Website

#### 

#### 1. Brief Information About the Project

#### Weather Wise is a web application providing users with real-time weather updates, hourly and weekly forecasts, interactive weather maps, and personalized notifications based on location and user preferences. The app aims to enhance how people interact with weather data by offering a modern, intuitive, and customizable interface.

**2. Set the Goals & Deliverables**

**Goals:**

* Provide real-time, accurate weather data.
* Deliver a personalized user experience based on preferences.
* Display weather maps and trends effectively.
* Enable login/signup functionality to save user preferences.

**Deliverables:**

* Fully responsive frontend (HTML, CSS, JavaScript).
* Backend system for user authentication (PHP & MySQL).
* APIs for real-time weather data (OpenWeatherMap or similar).
* Interactive weather maps and forecast features.

**3. Finalize the Modules of the Project**

* **Home Page (Live Weather + Search)**: Real-time weather data, city search functionality.
* **Login & Signup Module**: User authentication system for saving preferences.
* **Forecast Module (Hourly & Weekly)**: Detailed hourly and 7-day forecast with visual data.
* **Weather Map Module**: Interactive weather map with radar/temperature overlays.
* **Settings/Profile**: User preferences management for alerts and location.
* **About Page**: Information about the app’s purpose, technologies, and mission.
* **Contact Page**: Query form, email, and social media links.

**4. Define the audience**

The **Weather Wise** website is designed to cater to various types of users, each with specific needs and expectations when it comes to accessing weather information. Understanding the audience ensures the website's content, design, and features align with user preferences and provide a seamless experience. Below is a breakdown of the target audience for the Weather Wise app:

**a. Daily Commuters**

**Characteristics:**

* Regular travelers who need weather updates for daily planning.
* Typically rely on quick, reliable forecasts to plan their day or commute.

**Needs:**

* Real-time weather updates for specific locations.
* Hourly forecasts for their daily travel times.
* Notifications about sudden weather changes like rain or storms.

**b. Students and Professionals**

**Characteristics:**

* Students who need weather updates to plan outdoor activities or commute to school.
* Working professionals who need weather data for meetings, travel, or outdoor activities.

**Needs:**

* Clear and easy-to-read hourly and weekly forecasts.
* Personalization options, such as saving preferred cities or locations.
* Notifications about temperature drops, weather warnings, or significant weather changes.

**c. Outdoor Enthusiasts and Travelers**

**Characteristics:**

* Individuals who engage in outdoor activities such as hiking, camping, or sports.
* Travelers seeking weather information to plan their trips, especially in unfamiliar areas.

**Needs:**

* Detailed long-term weather forecasts, including extreme weather alerts.
* Interactive maps to visualize weather trends, precipitation, or radar.
* Specialized data for activities like snow, rain, or sun exposure.

**d. Health-Conscious Individuals and Families**

**Characteristics:**

* People who plan outdoor activities around weather patterns for health, safety, or recreational purposes (e.g., sun exposure, air quality, etc.).
* Families who need weather updates to ensure safe outdoor activities for children.

**Needs:**

* Alerts on UV levels, air quality, and pollen counts.
* Recommendations for outdoor activities based on weather conditions (e.g., the best time for a run, bike ride).
* Health-related weather notifications (e.g., high pollen days, heatwaves).

**e. Tourists and Regional Visitors**

**Characteristics:**

* Visitors or tourists looking for local weather updates when exploring new areas.
* Interested in experiencing weather-related activities or needing weather data for their travel plans.

**Needs:**

* Localized weather forecasts with specific details about tourist hotspots.
* Geolocation features to automatically provide weather data based on the user’s location.
* Interactive weather maps highlighting weather patterns in regions of interest.

**f. New Users (Unfamiliar Customers)**

**Characteristics:**

* Individuals who are not familiar with the Weather Wise app and are visiting for the first time.
* Looking for reliable and intuitive weather data but may not yet trust or know the platform well.

**Needs:**

* A professional, user-friendly interface to establish credibility.
* Clear explanations about the features and advantages of using Weather Wise.
* Simple onboarding experience for new users, explaining how to use the app efficiently.

**g. Tech-Savvy Users and Early Adopters**

**Characteristics:**

* Users who appreciate cutting-edge technology and innovative features in weather apps.
* Interested in advanced features like weather maps, alerts, and predictive trends.

**Needs:**

* Access to detailed data visualizations, interactive maps, and advanced weather models.
* Features like weather forecasting using machine learning, or data on trends over long periods.
* Seamless integration with other tech, like smart home devices (e.g., receiving weather updates through Alexa).

|  |
| --- |
| **Website Features Mapped to Audience Needs:** |
| | **Audience Segment** | **Key Features Needed** | | --- | --- | | **Daily Commuters** | Real-time weather updates, hourly forecasts, location-based alerts for weather changes, quick city search. | | **Students/Professionals** | Hourly and 7-day forecasts, location-based alerts, personalized weather data for specific cities, easy navigation. | | **Outdoor Enthusiasts/Travelers** | Interactive weather maps, detailed long-term forecasts, extreme weather alerts, location-based recommendations. | | **Health-Conscious Individuals** | UV index, air quality and pollen alerts, activity-based weather recommendations, health-related weather notifications. | | **Tourists/Regional Visitors** | Local weather updates, geolocation-based weather, regional travel tips, and recommendations. | | **New Users** | Clean and intuitive UI/UX, clear onboarding, easy-to-understand weather data and app functionalities. | | **Tech-Savvy Users** | Advanced weather visualizations, trend forecasting, integration with smart home devices, detailed maps. | |

**Why Understanding the Audience is Important:**

* **Personalized Experience:** By understanding the needs of various user segments, Weather Wise can provide a more personalized experience with location-specific alerts and tailored forecasts.
* **Enhanced User Engagement:** Designing features that meet the exact needs of each audience—whether it’s commuters needing quick updates or outdoor enthusiasts looking for detailed maps—ensures higher user engagement and satisfaction.
* **Increased Brand Loyalty:** Providing a seamless and personalized experience will help establish trust and loyalty among users. A reliable and intuitive app fosters long-term engagement and return visits.
* **Targeted Marketing Campaigns:** Understanding the audience allows for more effective marketing campaigns. For example, tailored ads for travelers, students, or health-conscious individuals can improve user acquisition and retention.

**5.Pain Points & Ideal Experience for Weather Wise Website**

1. Identifying Pain Points of Existing Weather Systems

a. Pain Point: Inaccurate or Delayed Weather Data

* Issue: Many weather websites provide outdated or inaccurate weather data, leading to user frustration when forecasts are unreliable.
* Impact: Users lose trust in the app, leading them to seek alternative sources for more accurate and timely updates.

b. Pain Point: Cluttered and Complex Interface

* Issue: Weather apps often have overloaded interfaces, making it difficult for users to quickly access key information such as the current temperature, radar, or forecasts.
* Impact: Users become overwhelmed by too much information and may abandon the app in favor of simpler alternatives.

c. Pain Point: Lack of Personalized Features

* Issue: Many weather systems don’t offer personalization, such as saving favorite locations, setting specific weather alerts, or customizing the app’s layout to suit individual preferences.
* Impact: Users feel like the app does not cater to their specific needs, leading to lower engagement.

d. Pain Point: Poor Mobile Experience

* Issue: Non-responsive designs or apps that don’t function well on mobile devices create an inferior user experience.
* Impact: Users struggle to view weather data or navigate the app on smartphones or tablets, leading to frustration and decreased usage.

e. Pain Point: Complex Navigation for Key Features

* Issue: Users often find it difficult to locate important features like hourly forecasts, weather maps, or radar images due to confusing navigation or poor menu organization.
* Impact: Users may leave the site or app in search of a more user-friendly alternative.

f. Pain Point: Insufficient Data Visualization

* Issue: Some weather systems fail to provide intuitive and interactive visualizations, such as weather maps, radar imagery, or graphical forecasts.
* Impact: Users miss out on a clearer understanding of weather patterns, which may affect their ability to plan ahead effectively.

g. Pain Point: Lack of Notifications for Severe Weather Events

* Issue: Many weather systems do not offer timely and actionable weather alerts for extreme weather conditions, such as storms, heavy rain, or snow.
* Impact: Users may not receive critical notifications about severe weather, leading to potential safety risks or missed opportunities to prepare.

2. Crafting the Ideal Experience

To address the pain points identified above, the Weather Wise website design should prioritize user-friendly features, accurate data, and smooth functionality, ensuring an ideal and enjoyable experience.

a. Real-Time, Accurate Weather Data

* Solution: Integrate reliable and up-to-date weather APIs (e.g., OpenWeatherMap) to ensure real-time, accurate weather information.
* Benefit: Users will trust the app for accurate weather updates, increasing engagement and satisfaction.

b. Clean, Minimalist Interface

* Solution: Use a simple and consistent design layout with clearly defined sections for key features like current weather, hourly forecasts, and maps.
* Benefit: Users can easily find and understand the information they need without distractions.

c. Personalization Options

* Solution: Allow users to save favorite locations, set custom weather alerts (e.g., notifications for rain, snow, or high temperatures), and customize the display for specific preferences.
* Benefit: Personalized experiences will keep users engaged and satisfied with the service, enhancing loyalty.

d. Mobile-Optimized Design

* Solution: Ensure the website and app are fully responsive, with a mobile-first approach that ensures compatibility across devices (smartphones, tablets, and desktops).
* Benefit: A mobile-friendly design guarantees that users have a seamless experience, no matter what device they use.

e. Easy Navigation for Quick Access

* Solution: Implement a simple, intuitive navigation structure with easy access to key sections such as weather data, forecasts, maps, and alerts.
* Benefit: Users can quickly find the information they need without unnecessary clicks or confusion.

f. Interactive Weather Visualizations

* Solution: Provide interactive weather maps, radar imagery, and graphical charts for hourly, daily, and weekly forecasts. Include features like precipitation forecasts, temperature overlays, and radar animations.
* Benefit: Clear, visually appealing weather data helps users understand weather patterns at a glance, improving their planning and decision-making.

g. Timely Weather Alerts

* Solution: Set up notifications for extreme weather events (e.g., thunderstorms, tornadoes, heatwaves), with the option for users to customize alert thresholds (e.g., severe rain or snow warnings).
* Benefit: Users will appreciate receiving timely, actionable alerts that help them stay safe during adverse weather conditions.

3. The Ideal User Journey for Weather Wise

Step 1: Visiting the Website

* Users land on a clean and welcoming homepage with clear navigation links to essential sections like current weather, forecasts, maps, and settings.

Step 2: Checking Current Weather

* Upon entering a location (or enabling location services), users are presented with real-time weather information, including temperature, humidity, wind speed, and weather conditions (e.g., sunny, cloudy, rainy).

Step 3: Exploring Hourly and Weekly Forecasts

* Users can view detailed hourly forecasts for the next 24 hours and a 7-day forecast with easy-to-read visualizations, such as temperature graphs and precipitation charts.

Step 4: Interacting with Weather Maps

* Users can click through to an interactive weather map showing precipitation, temperature, or radar overlays. The map should be easy to navigate and zoom into different locations for a more detailed view.

Step 5: Setting Personal Preferences

* Users can create an account to save their preferred locations, set custom weather alerts, and adjust their weather display settings.

Step 6: Receiving Alerts for Severe Weather

* Users receive timely notifications about extreme weather events relevant to their location, ensuring they can take precautions or plan accordingly.

Step 7: Engaging with Content and Updates

* Users can read blog posts or articles related to weather trends, forecasts, or safety tips. Engaging content will help build a community around the app.

Step 8: Returning to the Website Regularly

* With personalized weather data and alerts, users will continue returning to Weather Wise for daily updates and long-term forecasts.

**6. Visual Direction for Weather Wise App**

1. Visual Design Goals

The visual design of the Weather Wise app should provide a seamless, intuitive, and visually appealing experience that conveys trustworthiness, clarity, and modernity. The design should focus on the following principles:

* Clear and Readable: The app should prioritize legibility and easy navigation, enabling users to get weather information quickly.
* Modern and Dynamic: Utilize fresh, modern design elements with vibrant colors and clean layouts to make the app engaging and user-friendly.
* Brand Representation: The app's visual elements should reflect the app's purpose of providing accurate and real-time weather information while being visually attractive to users.

2. Defining the Core Visual Elements

a.ColorPalette  
A color scheme inspired by the natural environment, with cool blues and warm sun tones, will create a balanced visual experience. These colors will help users connect with the weather-related theme while ensuring readability.

| Color | Hex Code | Usage |
| --- | --- | --- |
| Sky Blue | #87CEEB | Backgrounds, icons, and headers to evoke clear skies. |
| Sunshine Yellow | #FFD700 | Highlighted buttons, alerts, and important call-to-actions. |
| Cloud White | #F0F8FF | Main background color for clarity and freshness. |
| Rainstorm Grey | #708090 | Text and accents for legibility in darker sections. |
| Stormy Navy | #1E2A47 | Footer, secondary buttons, and backgrounds for a calm effect. |
| Leaf Green | #32CD32 | Interactive elements like buttons and alerts for weather-specific features. |

b.Typography  
The typography should reflect the app’s modern and dynamic personality while maintaining readability across devices.

* Primary Font: Montserrat or Roboto (Sans-serif) – For headings and large text that emphasizes key information like weather conditions and forecasts.
* Secondary Font: Lora or Open Sans – For body text and descriptions to ensure legibility and clarity.
* Attributes: Use bold for headings, subheadings, and key data (e.g., temperature). Lighter font weights for descriptive text and forecasts.

c.Logos and Branding  
The logo should be simple, memorable, and weather-related, possibly incorporating elements like:

* A stylized cloud, sun, or raindrop to evoke weather imagery.
* The app name Weather Wise in the selected typography, ensuring the font is clean and modern.
* A monochrome version for use in headers, app icons, or smaller areas requiring a simpler design.

d. Imagery and Icons

* Photography:
  + Use high-quality images of nature (clear skies, sun, rain, and clouds) as part of background visuals or feature images for specific weather conditions.
  + Interactive or real-time imagery, such as live weather data visualizations (radar, temperature changes), will keep the app engaging.
* Icons:
  + Simple, clear icons for weather conditions (sun, cloud, rain, storm, snow), time indicators, and navigation buttons.
  + Animated Icons for weather conditions (e.g., animated raindrops, clouds moving, or the sun shining).

e. Hero Images and Graphics

* Hero images should highlight the current weather conditions with minimal text overlay, such as “Clear Sky” or “Light Rain.”
* Forecast graphics can show dynamic updates of hourly or daily forecasts with visually appealing graphics like temperature bars, wind speeds, and cloud cover maps.

3. Applying Visual Design to Pages

a. Home Page

* Banner Area: Display a large hero image or a dynamic weather graphic, with a clear text overlay showing the current weather condition and the location.
* Color Scheme: Use sky blue for background, stormy navy for footer, and sunshine yellow for call-to-action buttons like "Get Forecast" or "Set Alerts."
* Typography: Prominent, bold headings for current conditions with more subtle descriptions of temperature, humidity, and wind speed.

b. Weather Forecast Page

* Section Layout: Divide the page into clear sections for hourly forecasts, daily forecasts, and radar maps.
* Icons & Visuals: Use weather icons and graphs for hourly forecasts and a temperature slider for easy reading.
* Interactive Graphs: Display dynamic, interactive charts for temperature, wind, and precipitation over a specified period.

c. Location and Map Page

* Maps and Locations: Use a combination of animated maps for radar or precipitation patterns with interactive zoom features for detailed location-specific weather.
* Call-to-Action Buttons: Buttons like "Save Location" or "Get Directions" can be highlighted in sunshine yellow for easy interaction.

d. Alerts and Notifications Page

* Alert List: Display active weather alerts in a concise list with clear visuals (e.g., red or orange for severe warnings, yellow for general alerts).
* Engagement Features: Use leaf green buttons for enabling specific alerts and for setting custom preferences.

e. Profile and Personalization Page

* User Profile: Include a clean interface for users to save their favorite locations, customize units (Celsius or Fahrenheit), and set specific weather notifications.
* Data Presentation: Use a clean layout with clear, readable fonts, and intuitive icons to toggle preferences or view saved locations.

4. Layout and Design Hierarchy

The visual hierarchy is essential for guiding users through the app smoothly. It should focus on the most relevant data and actions, allowing for a seamless experience.

1. Headers and Banners: Make them large and engaging with immediate weather information and strong call-to-action buttons like “Get Forecast” or “View Map.”
2. Navigation Bar: Use a sticky, simple navigation bar that allows easy movement between sections, such as the home page, forecast, and profile.
3. Sections and Grids: Use card-style layouts for forecasts, and grids for hourly and daily data, ensuring easy navigation through weather data.
4. Call-to-Action Buttons: Use contrasting, vibrant colors like sunshine yellow or leaf green to highlight critical actions such as “Set Weather Alerts” or “Save Location.”

5. Expected Impact of Visual Direction

* Enhanced Engagement: A modern and visually appealing design will keep users engaged with the app and encourage them to return for timely updates.
* Stronger Brand Identity: Consistent use of colors, typography, and iconography will create a strong, recognizable brand identity for Weather Wise.
* Increased Retention: Clean layouts, intuitive navigation, and personalization options will enhance the user experience, leading to better retention and long-term usage.
* Higher Conversion Rates: Effective call-to-action placements and interactive elements will drive users to engage with key features, such as setting alerts or saving locations.

1. **Map out the Project structure**

weather\_wise\_website/

│

├── index.html # Home page (Current weather summary, location search)

├── about.html # About page (Information about the app, its features)

├── forecast.html # Detailed forecast page (Hourly, daily forecast)

├── radar.html # Weather radar map page (Interactive map for real-time weather data)

├── alerts.html # Weather alerts page (Severe weather notifications)

├── settings.html # User settings page (Location, temperature unit preferences)

├── login.html # Login page (User authentication)

├── register.html # Registration page (Account creation)

│

├── assets/

│ ├── css/

│ │ ├── style.css # Global CSS (All common styles)

│ │ ├── responsive.css # Media queries for mobile optimization

│ │

│ ├── js/

│ │ ├── main.js # Core interactive scripts (Weather data fetching, dynamic updates)

│ │ ├── forecast.js # JavaScript to handle the forecast page interactions

│ │ ├── map.js # Radar map interactions and dynamic elements

│ │ ├── alerts.js # Handling weather alerts and notifications

│ │ ├── formValidation.js # Scripts for login/registration form validation

│ │

│ ├── images/

│ │ ├── logo.png # App logo

│ │ ├── homepage\_banner.jpg # Hero banner for homepage

│ │ ├── weather\_icons/ # Weather condition icons (sun, rain, cloud, snow, etc.)

│ │ ├── maps/ # Background or map images for radar page

│ │

├── fonts/

│ ├── Montserrat/ # Primary font for headings and key UI text

│ ├── Roboto/ # Secondary font for body text

│

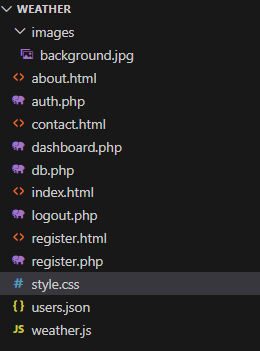
├── data/

│ ├── locations.json # List of user locations, city names for search suggestions

│ ├── weather\_data.json # Weather data (either mock or fetched from API)

│

└── README.md # Project documentation (App details, setup instructions**)**



**8. Plan the content for each page**

**1. Home Page**

**Purpose:**

* Welcome visitors and introduce Weather Wise.
* Highlight key features of the app and encourage users to explore.

**Content Plan:**

* **Header:**
  + Logo on the left.
  + Navigation menu: Home, Features, About, Blog, Contact.
  + Login/Sign-Up button on the top right.
* **Hero Section:**
  + High-quality banner image (e.g., dynamic weather scenes or map).
  + Tagline: “Weather at Your Fingertips.”
  + CTA button: “Check the Weather Now.”
* **Introduction Section:**
  + Brief description about Weather Wise and its core functionality.
  + CTA: “Learn More About Us” linking to the About page.
* **Features Section:**
  + Highlight core features like weather maps, real-time updates, and location-based forecasts.
  + CTA: “Explore Features” linking to the Features page.
* **Footer:**
  + Quick links, social media icons, contact information, and privacy policy.

**2. Features Page**

**Purpose:**

* Showcase the app's key features and benefits.

**Content Plan:**

* **Header:** (same as Home Page)
* **Main Features Sections:**
  + **Real-Time Weather Data:** Describe how users get instant updates.
  + **Hourly and Weekly Forecasts:** Explain forecast timelines.
  + **Interactive Weather Map:** Detail the map's functionality (radar, satellite).
  + **Personalized Notifications:** Let users know how they can set up custom alerts.
* **CTA Section:**
  + “Download Weather Wise Now” or “Explore More Features.”
* **Footer:** (same as Home Page)

**3. Login Page**

**Purpose:**

* Enable existing users to log into their accounts.

**Content Plan:**

* **Form:**
  + Email and Password fields.
  + Submit button.
* **Forgot Password Link:**
  + Redirects to the password recovery page.
* **CTA:**
  + Link to the Registration page: “New to Weather Wise? Sign Up Now!”

**4. Registration Page**

**Purpose:**

* Allow new users to create an account.

**Content Plan:**

* **Form Fields:**
  + Full Name.
  + Email.
  + Password and Confirm Password.
* **Form Validation:**
  + Password requirements (e.g., minimum length, special characters).
* **Submit Button:**
  + Validate data and submit.
* **Footer:** (same as Home Page)

**9. Add ideas for content, images & layout**

**1. Home Page**

**Layout Ideas:**

* **Header Section:**
  + Fixed navigation bar with a transparent background, blending with the hero image.
  + Logo on the left, navigation items (Home, Features, Blog, Contact) in the center, and Login/Register button on the top right.
* **Hero Section:**
  + Full-width background image with a weather-related theme (e.g., dynamic sky, rain, or sunshine).
  + Overlay text with a tagline: “Weather at Your Fingertips.”
  + A prominent CTA button: “Check the Weather Now” leading to the Features or Weather Page.
* **Features Section:**
  + Display 3-4 key features using cards or tiles (e.g., Real-Time Weather, Weather Maps, Notifications, Weekly Forecasts).
  + Each card will have a short description and an icon representing the feature.
* **Footer:**
  + Quick links, social media icons, contact details, and privacy policy.

**Content Ideas:**

* Short welcome message introducing the app's functionality.
* Announce seasonal features or updates, e.g., "Weather Maps Now Available!"
* Direct link to the "Features" or "Download Now" section.

**Image Ideas:**

* High-quality image of weather conditions (rain, sun, snow, etc.).
* A dynamic weather forecast screenshot showing the app in use.
* Background image with clear skies or dramatic cloud formations.

**2. Features Page**

**Layout Ideas:**

* **Categories Section:**
  + Divide the page into sections like Real-Time Weather, Maps, Forecasts, and Notifications.
  + For each category, include a clear, concise description, an icon, and a screenshot or mock-up of the feature.
* **Highlight Section:**
  + Carousel or grid layout to show customer favorites or the most popular features.
* **CTA Section:**
  + “Download Weather Wise Now” button at the bottom of the page linking to app download options.

**Content Ideas:**

* Provide a detailed breakdown of features such as hourly forecasts, radar maps, and notifications.
* Include short descriptions for each feature, and possibly a brief demo video of the app in use.

**Image Ideas:**

* Mock-ups of the app showing features like weather maps or forecast views.
* Icons for each feature to add visual appeal.
* Animated weather symbols to represent different features.

**3. Login Page**

**Layout Ideas:**

* **Minimalist Form Interface:**
  + Large, centered form fields for email and password.
  + A clear "Log In" button with contrasting color.
  + Option to log in via Google/Facebook (optional).
* **Side Panel/Banner:**
  + A banner featuring a weather-related background or the app’s logo.

**Content Ideas:**

* Fields for email and password.
* Reminder: “Forgot your password? Reset it here.”
* CTA: “New to Weather Wise? Sign Up Now!”

**Image Ideas:**

* Light, welcoming background image with clear weather elements (clouds, sun, etc.).
* Icons for login options (email, Google, etc.).

**4. Registration Page**

**Layout Ideas:**

* **Centered Form:**
  + Clean, simple form for users to enter their full name, email, password, and confirm password.
  + Include a password strength meter.
* **CTA Section:**
  + “Join Weather Wise and get personalized weather updates!” tagline.

**Content Ideas:**

* Highlight the benefits of signing up (personalized alerts, saved locations, etc.).
* Checkbox for opting into newsletters or weather notifications.

**Image Ideas:**

* Welcoming imagery, like a peaceful sky or a city skyline.
* Illustrations for form fields (e.g., a lock icon for password).

**5. Visual Design Ideas for All Pages**

**Colors:**

* Primary colors: Blue (for calm and trust), white (for clarity), and light grey (for simplicity).
* Accent color: Yellow or orange for buttons and highlights (representing warmth and alertness).

**Fonts & Typography:**

* **Headers:** A friendly, sans-serif font like “Poppins” or “Montserrat.”
* **Body Text:** Clean, easy-to-read font like “Roboto” or “Open Sans.”
* **CTA Text:** Bold font, highlighting important actions (e.g., “Sign Up Now,” “Explore Features”).

**Logos:**

* A clean logo, possibly with a weather icon like a cloud, sun, or rain drop integrated into the typography.

**Images:**

* Realistic, high-quality weather imagery.
* Use illustrations for user-friendly icons.
* Interactive maps, mock-ups, or screenshots of the app in use.

**10. Determine your site structure or Create content for your core website pages:**

1. **Home page**
2. **Login Page**
3. **Registration Page**
4. **Features Page**
5. **Overall Website Structure Map**

**1. Home Page**

**Sections and Content:**

* **Header:**
  + **Logo:** Simple, modern logo with a cloud, sun, or other weather-related imagery.
  + **Navigation Links:** Home, Features, Blog, Contact, Login/Register.
  + **Call-to-Action Button:** “Check Weather Now” or “Get Started” linking to the Weather Page or Features.
* **Hero Section:**
  + **Full-Width Background Image:** A dynamic weather-related image, such as a sunny sky or dramatic weather conditions.
  + **Text Overlay:** “Accurate Weather at Your Fingertips.”
  + **Call-to-Action Button:** “See Today’s Forecast” or “Explore Features” to guide users to the weather features or app download.
* **Features Section:**
  + **3-4 Key Features:** Use icons with short descriptions for Real-Time Weather, Maps, Weekly Forecast, and Notifications.
  + **Button:** “Explore Features” or “Learn More” linking to the Features page.
* **Social Proof Section:**
  + **Testimonial Teaser:** Display 2-3 snippets of positive customer feedback.
  + **Button:** “See More Reviews” linking to the Testimonials page.
* **Footer:**
  + **Quick Links:** About, Features, Contact, Blog.
  + **Social Media Icons:** Facebook, Twitter, Instagram.
  + **Location Info:** Address, Google Maps embed for the Weather Wise headquarters.
  + **Privacy Policy & Terms of Use Links.**

**2. Login Page**

**Sections and Content:**

* **Login Form:**
  + **Fields:** Email/Username, Password.
  + **Buttons:**
    - Submit login credentials.
    - “Forgot Password?” link for password recovery.
* **Call to Action:**
  + “New to Weather Wise? Sign Up Now!” linking to the Registration page.
* **Background Image:** Subtle weather elements like clouds, rain, or a sunrise to maintain the app’s theme.

**3. Registration Page**

**Sections and Content:**

* **Registration Form:**
  + **Fields:** Name, Email, Password, Confirm Password.
  + **Checkbox:** Option to subscribe to weather notifications, newsletters, and promotions.
  + **Submit Button:** “Create Account.”
* **Password Strength Indicator:** Display a strength meter to help users choose a secure password.
* **Call to Action:**
  + “Already have an account? Log in here” linking to the Login page.
* **Background Image:** Light weather-themed background (e.g., sunrise, clear skies).

**4. Features Page**

**Sections and Content:**

* **Categories Section:**
  + **Real-Time Weather:** Describe how the app gives accurate weather reports for your location.
  + **Weather Maps:** Explain the feature of interactive weather maps.
  + **Weekly Forecasts:** Provide details on long-term weather forecasts.
  + **Personalized Notifications:** How users can receive notifications based on weather preferences.
* **Highlight Section:**
  + **Carousel or Grid:** Showcasing screenshots or mock-ups of each feature.
* **CTA Button:** “Download Now” or “Get Started” linking to the app download or user registration page.

**5. Overall Website Structure Map**

* **Home Page:** Introduction to the app, key features, testimonials, and call to action.
* **Login Page:** User login interface for existing users.
* **Registration Page:** New user sign-up with options for weather notifications and personalized features.
* **Features Page:** Detailed breakdown of the app's features.

**11. Create and collect design elements**

1. **Colors**
2. **Fonts and typography**
3. **Logos**
4. **Images and photos**

The combination of **cool blues, vibrant natural colors, and neutral colors** will create a user experience that is both professional and energetic. The use of **simple yet functional typography** and **interactive elements** ensures the site is both easy to navigate and visually appealing, while **dynamic imagery** of weather conditions will emotionally connect with users.

**1. Colors**

The color scheme for **Weather Wise** should communicate clarity, trust, and sophistication, while emphasizing the dynamic and ever-changing nature of weather.

* **Primary Colors:**
  + **Sky Blue (#3EB9FF):** Represents calmness, trust, and clarity. Ideal for headers, call-to-action buttons, and links.
  + **Cloud White (#F0F4F8):** A soft and neutral color, used for backgrounds and sections to keep the site feeling light and airy, representing cloud cover and clear skies.

**2. Fonts and Typography**

Typography should reflect modern, clear, and approachable qualities, with readability as a priority for quick weather updates and mobile responsiveness.

* **Heading Font:**
  + **Poppins (sans-serif):** A modern, geometric font with a clean, rounded style. It’s perfect for headings and subheadings, bringing a contemporary and approachable vibe.
  + **Roboto Slab (serif):** For more serious or scientific topics like weather forecasts, research, and climate data. The slab serif provides an element of stability and trustworthiness.
* **Body Font:**
  + **Open Sans (sans-serif):** Clean, simple, and easy to read for larger text blocks like product descriptions, blogs, and weather details.
* **Font Weights:**
  + **Bold:** For headings and key information (e.g., current weather data).
  + **Regular:** For body text, ensuring a smooth reading experience.

**4. Imagery and Photos**

Imagery plays a crucial role in setting the mood and helping users visually connect with the website’s message.

* **Weather Imagery:**
  + High-quality photos showcasing different weather conditions (sunny, rainy, cloudy) to create a dynamic and immersive user experience. These images can be used on the homepage, forecast pages, and blog sections.
  + Icons of weather conditions like thunderstorms, rain, or sunny days can be used to visually represent forecast information.
* **Background Images:**
  + Consider using subtle weather patterns, such as clouds or a sky gradient, as the background for sections to maintain a clean and modern feel while visually supporting the weather theme.
* **Lifestyle Imagery:**
  + Candid photos of people using the app in various scenarios, like checking the weather on a phone or using it to plan a trip, can humanize the brand and show how the app fits into daily life.
* **Screenshots of App/Features:**
  + Use screenshots that show the app’s real-time weather data, maps, and notifications, providing users with a preview of what to expect.

**5. Interactive Elements and Buttons**

* **Navigation Buttons:**
  + **Primary Buttons (e.g., "Get Weather Now," "Sign Up"):** These buttons should be large, easy to click/tap, and should have a hover effect (e.g., color change or subtle shadow) to indicate interactivity.

Experiment No.2

## **Problem Statement:**

|  |  |
| --- | --- |
| 2. | HTML   1. Create a detailed home page for the coffee shop website. 2. Create a detailed menu/product page for the coffee shop website, listing all available items categorized appropriately. 3. Create a cart page that allows customers to review and manage the items they wish to purchase before proceeding to checkout. 4. Create an about us page that provides detailed information about the coffee shop’s history, mission, and team. 5. Create a contact page that allows customers to easily get in touch with the coffee shop through a form. 6. Design and implement admin/user registration form for the coffee shop website. 7. Design and implement admin/user login form for the coffee shop website. |

**Objective**

The main objective of the **Weather Wise** website is to provide users with a real-time, accurate, and user-friendly weather forecasting platform. It aims to enhance the user experience by offering features like current weather conditions, weekly forecasts, interactive weather maps, and location-based weather alerts. The project is designed to be responsive, visually appealing, and easy to use on both desktop and mobile devices.

**Theory**

In today's digitally driven world, instant access to weather information plays a crucial role in daily planning—whether it's for travel, agriculture, outdoor events, or health and safety precautions. Traditional weather checking methods like television news, newspaper columns, or generic mobile apps often fail to deliver personalized, real-time, and location-specific data in a user-friendly way. To address this gap, the **Weather Wise** project is proposed as a web-based platform that delivers **live weather data, weekly forecasts, and weather alerts** tailored to user needs.

The **Weather Wise** web app utilizes modern web technologies and third-party APIs to present accurate weather data to users in an intuitive and interactive format. This theoretical overview describes the tools, technologies, design methodologies, and integration mechanisms involved in building the Weather Wise system.

**Web Technologies Used**

The development of the Weather Wise app is based on **front-end and optional back-end technologies**:

**1. HTML (HyperText Markup Language)**

HTML is used to structure the pages of the website, including the home page, forecast display, login/register forms, and contact form. Each page uses semantic HTML elements for clarity and accessibility.

**2. CSS (Cascading Style Sheets)**

CSS is used to style the website with a clean, modern, and responsive layout. Responsive design ensures that the Weather Wise app functions efficiently across devices—desktops, tablets, and smartphones.

**3. JavaScript (JS)**

JavaScript enables dynamic behavior on the website. Most notably, it is used to fetch real-time data from the weather API, validate forms, and update the DOM dynamically based on user input.

**4. Weather API (e.g., OpenWeatherMap API)**

The app fetches weather information such as temperature, humidity, wind speed, weather conditions, and weekly forecasts using a public weather API. The API request is made in JavaScript and the response data is parsed and rendered in real time.

**5. PHP & MySQL (Optional - For Login/Register System)**

If the system includes user authentication, PHP handles the server-side scripting and interacts with a MySQL database to store user credentials and preferences securely. Form inputs are sanitized to prevent SQL injection and other security vulnerabilities.

**System Features**

**1. Home Page**

The homepage acts as a weather dashboard where users can enter a city or allow location access to receive current weather updates instantly. It displays temperature, weather conditions, and icons dynamically.

**2. Forecast Page**

This section provides a **7-day weather forecast** based on user-selected location. Data is shown in individual day cards for better visualization and quick understanding.

**3. Login and Registration**

Users can create accounts to save their favorite cities, set preferences, and receive personalized forecasts. Secure login ensures privacy and helps maintain custom settings across sessions.

**4. Contact and About Pages**

The Contact page enables users to reach out for queries or feedback via a form. The About page provides details about the purpose, vision, and team behind Weather Wise.

A.HOME PAGE

CODE:

<!-- home.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Weather Wise - Home</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<header>

<h1>Weather Wise</h1>

<nav>

<a href="home.html">Home</a>

<a href="forecast.html">Forecast</a>

<a href="about.html">About</a>

<a href="contact.html">Contact</a>

<a href="login.html">Login</a>

</nav>

</header>

<main>

<section class="hero">

<h2>Your Personalized Weather Dashboard</h2>

<input type="text" placeholder="Enter city" id="citySearch">

<button>Search</button>

</section>

<section class="weather-overview">

<h3>Current Weather</h3>

<div id="current-weather"></div>

</section>

</main>

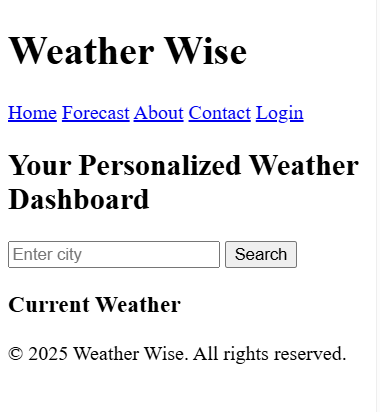
<footer>

<p>&copy; 2025 Weather Wise. All rights reserved.</p>

</footer>

</body>

</html>



B.Forecast / Product Page

CODE:

<!-- forecast.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Weekly Forecast - Weather Wise</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<header>

<h1>Weather Wise</h1>

<nav>

<a href="home.html">Home</a>

<a href="forecast.html">Forecast</a>

<a href="about.html">About</a>

<a href="contact.html">Contact</a>

</nav>

</header>

<main>

<h2>7-Day Weather Forecast</h2>

<div class="forecast-grid">

<!-- Loop or JS will populate these -->

<div class="day-card">Monday: ☀️ 28°C</div>

<div class="day-card">Tuesday: 🌧️ 24°C</div>

<!-- etc -->

</div>

</main>

<footer>

<p>&copy; 2025 Weather Wise</p>

</footer>

</body>

</html>



1. **Weather Favorites / Cart Page**

CODE:

<!-- favorites.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>My Locations - Weather Wise</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<header>

<h1>Weather Wise</h1>

<nav>

<a href="home.html">Home</a>

<a href="forecast.html">Forecast</a>

<a href="favorites.html">My Locations</a>

</nav>

</header>

<main>

<h2>Saved Locations</h2>

<div class="favorites-list">

<!-- List of favorite cities -->

<div class="location">Pune <button>Remove</button></div>

<div class="location">Mumbai <button>Remove</button></div>

</div>

</main>

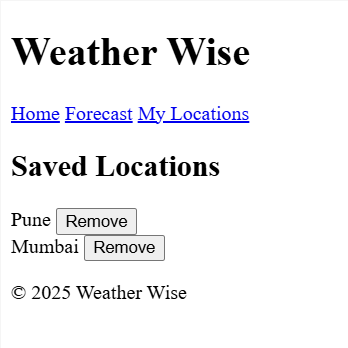
<footer>

<p>&copy; 2025 Weather Wise</p>

</footer>

</body>

</html>



**D. About Us Page**

CODE:

<!-- about.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>About Us - Weather Wise</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<header>

<h1>About Weather Wise</h1>

<nav>

<a href="home.html">Home</a>

<a href="about.html">About</a>

<a href="contact.html">Contact</a>

</nav>

</header>

<main>

<section>

<h2>Our Mission</h2>

<p>To provide reliable and accessible weather data to everyone, anywhere.</p>

</section>

<section>

<h2>Our Story</h2>

<p>Started as a college project, now aims to become a full-fledged weather platform.</p>

</section>

<section>

<h2>Team</h2>

<ul>

<li>Developer: [Your Name]</li>

<li>Designer: [Your Name]</li>

</ul>

</section>

</main>

<footer>

<p>&copy; 2025 Weather Wise</p>

</footer>

</body>

</html>  


E.Contact Page

CODE: <!-- contact.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Contact Us - Weather Wise</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<header>

<h1>Contact Weather Wise</h1>

</header>

<main>

<form>

<label for="name">Your Name:</label><br>

<input type="text" id="name" name="name"><br><br>

<label for="email">Email:</label><br>

<input type="email" id="email" name="email"><br><br>

<label for="message">Message:</label><br>

<textarea id="message" name="message" rows="5" cols="30"></textarea><br><br>

<input type="submit" value="Send Message">

</form>

</main>

<footer>

<p>&copy; 2025 Weather Wise</p>

</footer>

</body>

</html>



F.Registration Form

CODE:

<!-- register.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Register - Weather Wise</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<h2>Create Account</h2>

<form>

<label>Full Name:</label><br>

<input type="text" name="fullname"><br><br>

<label>Email:</label><br>

<input type="email" name="email"><br><br>

<label>Password:</label><br>

<input type="password" name="password"><br><br>

<label>Account Type:</label><br>

<select name="role">

<option value="user">User</option>

<option value="admin">Admin</option>

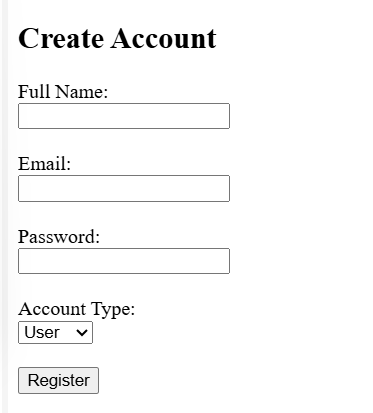
</select><br><br>

<input type="submit" value="Register">

</form>

</body>

</html>



G. Login Form

CODE:

<!-- login.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Login - Weather Wise</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<h2>Login to Weather Wise</h2>

<form>

<label>Email:</label><br>

<input type="email" name="email"><br><br>

<label>Password:</label><br>

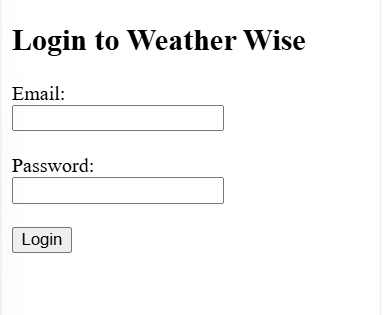
<input type="password" name="password"><br><br>

<input type="submit" value="Login">

</form>

</body>

</html>



EXPERIMENT NO :3

|  |  |
| --- | --- |
| 3. | CSS   1. Enhance the layout of the coffee shop website using CSS Grid for the home page. 2. Use CSS Grid to layout the menu/product items in a structured and style the menu categories with appropriate headings, spacing, separators, images, descriptions, and prices. |

**OBJECTIVE**

The objective of the CSS module in the **Weather Wise Website** project is to enhance the visual presentation and layout of the website using **CSS Grid**. This involves organizing the **home page** and **weather data cards** in a structured, responsive, and user-friendly manner. CSS Grid enables the creation of modern layouts that adapt to various screen sizes and improve user interaction with weather-related content like forecasts, weather maps, and city-based weather cards.

**THEORY:**

**Introduction**

The **Weather Wise Website** aims to provide real-time weather data and forecasts with an attractive, responsive, and interactive web design. HTML gives the content structure, but CSS is essential for layout, styling, and overall user experience. As user attention spans are short, a clean and intuitive layout plays a crucial role in keeping users engaged.

**CSS Grid** is a modern CSS layout system specifically designed for building two-dimensional layouts. It gives developers powerful tools to align elements vertically and horizontally, control spacing, and define areas in a grid-like structure—making it ideal for weather dashboards and forecast sections.

**CSS Grid for Home Page Layout**

The **home page** serves as the entry point of the Weather Wise website. A well-laid-out grid can highlight core sections such as:

* **City Search bar**
* **Current Weather Summary**
* **Featured Weather Cards**
* **Forecast Previews**
* **Weather Alerts/Warnings**
* **Map/Location Widgets**

CSS Grid allows defining rows and columns, using named areas or fractional units (fr) to distribute space effectively.

* + 1. Home Page Layout Using CSS Grid

CODE: .home-container {

display: grid;

grid-template-areas:

"header header"

"search search"

"current forecast"

"map map"

"footer footer";

grid-gap: 20px;

padding: 20px;

}

.header { grid-area: header; }

.search { grid-area: search; }

.current { grid-area: current; }

.forecast { grid-area: forecast; }

.map { grid-area: map; }

.footer { grid-area: footer; }

B. Menu/Product-style Weather Layout Using CSS Grid

CODE: <div class="weather-grid">

<div class="weather-card">

<h3>Pune</h3>

<img src="cloudy.png" alt="cloudy">

<p>28°C</p>

<p>Cloudy</p>

<p>Humidity: 70%</p>

</div>

<!-- more cards -->

</div>

.weather-grid {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));

gap: 1.5rem;

padding: 1rem;

}

.weather-card {

background: #f8f9fa;

padding: 15px;

border-radius: 10px;

text-align: center;

box-shadow: 0 2px 8px rgba(0,0,0,0.1);

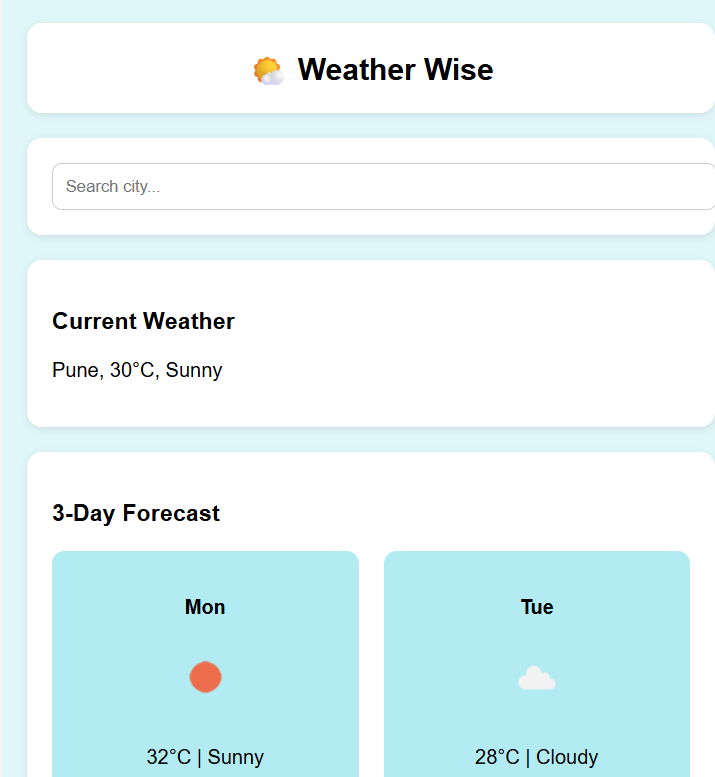
}

.weather-card img {

width: 50px;

margin-bottom: 10px;

}



EXPERIMENT NO 4

|  |  |
| --- | --- |
| 4. | CSS   1. Enhance the cart page to make it user-friendly and visually appealing. Style the cart items with appropriate margins, paddings, and input field styles to provide a seamless shopping experience. 2. Enhance and style the about us page with appropriate margins, paddings, and input field styles. 3. Enhance and style the contact page to make it user-friendly and visually appealing. Style the contact form with appropriate margins, paddings, and input field styles. 4. Enhance and style the admin/user registration form with appropriate margins, paddings, and input field styles. 5. Enhance and style the admin/user login form with appropriate margins, paddings, and input field styles. |

**OBJECTIVE**

* To improve the **user interface and experience** across key pages of the Weather Wise website.
* To apply consistent **spacing, alignment, and styling** using CSS to make pages visually appealing and easy to use.
* To style **forms and interactive elements** with intuitive input fields, buttons, and focus states.
* To ensure pages are **responsive** and maintain accessibility on all device sizes.
* To create a **professional and clean look** that matches the brand identity of Weather Wise.

**THEORY**

**1. Importance of User-Friendly Layout**

A well-designed layout with proper margins and paddings ensures content is clear and easy to scan. This prevents clutter and enhances readability, improving user satisfaction.

**2. Consistent Form Styling**

Forms are critical for login, registration, and contact pages. Consistent input sizes, padding, border styles, and focus states make form completion easier and reduce user errors.

**3. Visual Appeal Through CSS**

Using subtle shadows, rounded corners, and harmonious colors enhances the aesthetics and gives a modern feel. Buttons and interactive elements with hover effects provide clear visual feedback.

**4. Responsive and Accessible Design**

CSS should ensure elements adjust smoothly to screen size changes, making the site mobile-friendly. Input fields and buttons must be large enough for touch input and meet accessibility standards.

**5. CSS Best Practices**

* Use box-sizing: border-box for predictable element sizing.
* Separate content into logical blocks with padding and margin.
* Use flexbox/grid to align items cleanly.
* Use transitions on hover/focus for smooth interactivity

CODE:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>Weather Wise - Styled Pages</title>

<style>

/\* Global styles \*/

\*, \*::before, \*::after {

box-sizing: border-box;

}

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background: #e0f2f1;

margin: 0; padding: 20px;

color: #004d40;

}

h2 {

margin-bottom: 15px;

font-weight: 700;

color: #00796b;

text-align: center;

}

/\* Common container for pages \*/

.page-container {

background: white;

max-width: 600px;

margin: 30px auto;

padding: 25px 30px;

border-radius: 12px;

box-shadow: 0 3px 10px rgba(0, 0, 0, 0.12);

}

/\* --- CART PAGE (Saved locations/subscriptions) --- \*/

.cart-item {

display: flex;

justify-content: space-between;

align-items: center;

padding: 15px 0;

border-bottom: 1px solid #b2dfdb;

}

.cart-item:last-child {

border-bottom: none;

}

.cart-item-name {

font-size: 1.1rem;

font-weight: 600;

}

.cart-item-qty input {

width: 60px;

padding: 7px;

border-radius: 8px;

border: 1.5px solid #00796b;

text-align: center;

font-size: 1rem;

transition: border-color 0.3s ease;

}

.cart-item-qty input:focus {

outline: none;

border-color: #004d40;

box-shadow: 0 0 6px #004d4088;

}

/\* --- FORM STYLES (About Us contact form, Registration, Login) --- \*/

label {

display: block;

margin-bottom: 8px;

font-weight: 600;

color: #004d40;

}

input[type="text"],

input[type="email"],

input[type="password"],

textarea {

width: 100%;

padding: 12px 14px;

margin-bottom: 18px;

border: 1.8px solid #00796b;

border-radius: 10px;

font-size: 1rem;

transition: border-color 0.3s ease;

resize: vertical;

}

input[type="text"]:focus,

input[type="email"]:focus,

input[type="password"]:focus,

textarea:focus {

border-color: #004d40;

outline: none;

box-shadow: 0 0 8px #004d4088;

}

textarea {

min-height: 100px;

}

button {

background-color: #00796b;

color: white;

padding: 14px;

font-size: 1.1rem;

border: none;

border-radius: 10px;

cursor: pointer;

width: 100%;

transition: background-color 0.3s ease;

font-weight: 600;

}

button:hover {

background-color: #004d40;

}

/\* About Us page enhancements \*/

.about-us p {

line-height: 1.6;

margin-bottom: 18px;

color: #00695c;

font-size: 1.05rem;

}

</style>

</head>

<body>

<!-- CART PAGE -->

<section class="page-container">

<h2>Saved Locations</h2>

<div class="cart-item">

<div class="cart-item-name">Pune</div>

<div class="cart-item-qty">

<input type="number" value="1" min="1" />

</div>

</div>

<div class="cart-item">

<div class="cart-item-name">New York</div>

<div class="cart-item-qty">

<input type="number" value="2" min="1" />

</div>

</div>

</section>

<!-- ABOUT US PAGE -->

<section class="page-container about-us">

<h2>About Weather Wise</h2>

<p>

Weather Wise is dedicated to providing accurate, real-time weather information globally. Our mission is to empower users with precise forecasts and weather alerts to plan their day efficiently.

</p>

<p>

Founded in 2024, our team combines meteorological expertise with cutting-edge technology to deliver seamless weather experiences across devices.

</p>

</section>

<!-- CONTACT PAGE -->

<section class="page-container">

<h2>Contact Us</h2>

<form>

<label for="name">Name</label>

<input type="text" id="name" placeholder="Your Name" required />

<label for="email">Email</label>

<input type="email" id="email" placeholder="Your Email" required />

<label for="message">Message</label>

<textarea id="message" placeholder="Your message here..." required></textarea>

<button type="submit">Send Message</button>

</form>

</section>

<!-- REGISTRATION FORM -->

<section class="page-container">

<h2>Register</h2>

<form>

<label for="reg-username">Username</label>

<input type="text" id="reg-username" placeholder="Choose a username" required />

<label for="reg-email">Email</label>

<input type="email" id="reg-email" placeholder="Enter your email" required />

<label for="reg-password">Password</label>

<input type="password" id="reg-password" placeholder="Create a password" required />

<button type="submit">Sign Up</button>

</form>

</section>

<!-- LOGIN FORM -->

<section class="page-container">

<h2>Login</h2>

<form>

<label for="login-username">Username or Email</label>

<input type="text" id="login-username" placeholder="Enter username or email" required />

<label for="login-password">Password</label>

<input type="password" id="login-password" placeholder="Enter password" required />

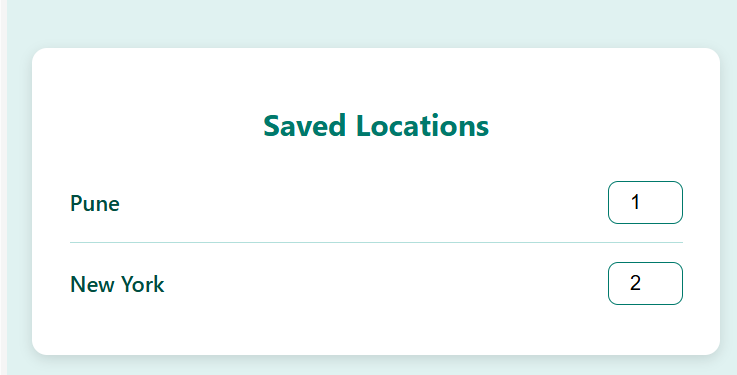
<button type="submit">Login</button>

</form>

</section>

</body>

</html>



EXPERIMENT NO 5.

|  |  |
| --- | --- |
| 5. | JavaScript   1. Implement user registration and login forms for the coffee shop website. These forms will allow users to create an account, log in, and access personalized features, such as saving favorite items or viewing order history.   User Registration Form will allow new customers to sign up and create an account on the website. The form will capture basic user details, including the name, email address, and password (not limited to these fields).  User Login Form will allow registered users to log into their accounts. The form will require an email address and a password to authenticate the user.   1. Provide validations for user registration and login forms to validate the input to ensure that all required fields are filled and that the email format is valid. (**Contents beyond Syllabus)** 2. Develop cart functionality to allow users to add items, update quantities, and remove items. |

**OBJECTIVE**

* To implement **user registration and login functionality** enabling users to create accounts and access personalized features such as saving favorite locations, setting alerts, or managing subscriptions.
* To provide **client-side validation** for registration and login forms ensuring that users enter valid and complete data before submission, thereby improving user experience and reducing errors.
* To develop **interactive cart functionality** that allows users to add weather-related items (e.g., premium subscriptions, alerts, saved locations), update quantities or preferences, and remove items from their cart seamlessly.
* To enhance user engagement by making the website interactive and responsive through JavaScript-driven dynamic content updates.

**THEORY**

**1. User Registration and Login Functionality**

User registration and login are fundamental features for personalized web applications. Registration captures user data like name, email, and password, which is stored securely (usually server-side). Login authenticates users by verifying entered credentials against stored data.

On the client side, JavaScript can collect input data, validate it before sending to the server, and provide instant feedback (e.g., invalid email format, empty fields). This reduces unnecessary server requests and improves user experience.

**2. Form Validations**

Client-side validations involve checking:

* Required fields are not empty.
* Email fields match standard email regex patterns.
* Password meets minimum strength requirements (e.g., length, complexity).
* Confirmation password matches the original password (if applicable).

Validations ensure data integrity and prevent invalid submissions. Instant validation feedback also helps users correct mistakes immediately.

**3. Cart Functionality**

A cart enables users to select multiple products or services and manage them before checkout. For Weather Wise, this might include saving favorite cities, subscribing to weather alerts, or buying premium content.

Key functionalities include:

* Adding items to the cart.
* Updating item quantities or preferences.
* Removing items from the cart.
* Calculating totals or displaying summary info dynamically.

JavaScript can update the cart UI in real-time without page reloads, using DOM manipulation and event listeners.

**4. Importance of JavaScript in Enhancing UX**

JavaScript allows the creation of interactive, real-time responsive features on the client side, providing:

* Immediate validation feedback.
* Smooth user interactions (e.g., adding/removing items).
* Enhanced navigation and control over user data inputs.

CODE:

// Simulated "database" (in-memory)

const users = [];

let loggedInUser = null;

// --- Registration ---

document.getElementById('registerForm').addEventListener('submit', function (e) {

e.preventDefault();

const name = document.getElementById('regName').value.trim();

const email = document.getElementById('regEmail').value.trim();

const password = document.getElementById('regPassword').value;

const confirmPassword = document.getElementById('regConfirmPassword').value;

const errorDiv = document.getElementById('registerError');

// Clear previous errors

errorDiv.textContent = '';

// Validation

if (!name || !email || !password || !confirmPassword) {

errorDiv.textContent = 'All fields are required.';

return;

}

if (!validateEmail(email)) {

errorDiv.textContent = 'Invalid email format.';

return;

}

if (password !== confirmPassword) {

errorDiv.textContent = 'Passwords do not match.';

return;

}

if (password.length < 6) {

errorDiv.textContent = 'Password must be at least 6 characters.';

return;

}

// Check if email already exists

if (users.some(user => user.email === email)) {

errorDiv.textContent = 'Email already registered.';

return;

}

// Save user

users.push({ name, email, password });

alert('Registration successful! You can now log in.');

this.reset();

});

// --- Login ---

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

e.preventDefault();

const email = document.getElementById('loginEmail').value.trim();

const password = document.getElementById('loginPassword').value;

const errorDiv = document.getElementById('loginError');

errorDiv.textContent = '';

if (!email || !password) {

errorDiv.textContent = 'All fields are required.';

return;

}

if (!validateEmail(email)) {

errorDiv.textContent = 'Invalid email format.';

return;

}

const user = users.find(u => u.email === email && u.password === password);

if (!user) {

errorDiv.textContent = 'Invalid email or password.';

return;

}

loggedInUser = user;

alert(`Welcome back, ${user.name}!`);

this.reset();

});

// Email validation function

function validateEmail(email) {

const re = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

return re.test(email);

}

// --- Cart Functionality ---

// Example items available (can be weather alerts, premium features)

const products = [

{ id: 1, name: "Premium Weather Alerts", price: 10 },

{ id: 2, name: "Ad-Free Experience", price: 5 },

{ id: 3, name: "Detailed Forecasts", price: 8 }

];

// User's cart

let cart = [];

// Add product to cart

function addToCart(productId) {

const product = products.find(p => p.id === productId);

if (!product) return;

const cartItem = cart.find(item => item.id === productId);

if (cartItem) {

cartItem.quantity++;

} else {

cart.push({ ...product, quantity: 1 });

}

renderCart();

}

// Remove product from cart

function removeFromCart(productId) {

cart = cart.filter(item => item.id !== productId);

renderCart();

}

// Update quantity

function updateQuantity(productId, quantity) {

const cartItem = cart.find(item => item.id === productId);

if (cartItem) {

cartItem.quantity = quantity > 0 ? quantity : 1;

}

renderCart();

}

// Render cart UI

function renderCart() {

const cartDiv = document.getElementById('cartItems');

cartDiv.innerHTML = '';

let total = 0;

cart.forEach(item => {

total += item.price \* item.quantity;

const itemDiv = document.createElement('div');

itemDiv.style.marginBottom = '10px';

itemDiv.innerHTML = `

<strong>${item.name}</strong> - $${item.price}

<input type="number" min="1" value="${item.quantity}" style="width: 50px;" onchange="updateQuantity(${item.id}, this.value)" />

<button onclick="removeFromCart(${item.id})">Remove</button>

`;

cartDiv.appendChild(itemDiv);

});

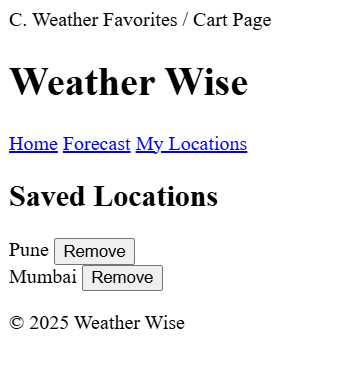
document.getElementById('cartTotal').textContent = total.toFixed(2);

}

// Example: adding some items initially

addToCart(1);

addToCart(2);



EXPERIMENT.6

|  |  |
| --- | --- |
| 6. | JavaScript   1. The user login form will allow registered users to log into their accounts. The form will require an email address and a password to authenticate the user. 2. If the login is successful, the user should be redirected to the homepage or their user dashboard. (**Contents beyond Syllabus)** 3. Use localStorage or sessionStorage to store authentication data, such as the user’s email and login status. This ensures that once a user is logged in, they remain authenticated even after the page reloads or when they visit the site again. (**Contents beyond Syllabus)** 4. Save the cart data to local storage when items are added, updated, or removed. Retrieve and load the cart data from local storage when the page loads. (**Contents beyond Syllabus)** |

**Objective**

* To implement a **user login system** that allows registered users to securely access their accounts on the Weather Wise app.
* To enable **persistent user sessions** by storing authentication data in **localStorage**, so users remain logged in even after refreshing or revisiting the site.
* To develop **cart functionality** that lets users add, update, and remove items (such as weather services or subscriptions) with a seamless and interactive user experience.
* To persist cart data across sessions by saving it in **localStorage**, ensuring users don’t lose their selections when navigating or refreshing pages.
* To provide **form validation** to improve data quality and user experience during login and cart operations.

**Theory**

**User Authentication with JavaScript and localStorage**

User authentication in web applications usually involves verifying credentials (like email and password) and maintaining session state to track logged-in users. Traditionally, this is done server-side, but for simple or demo apps, **localStorage** can be used to store authentication data securely on the client side.

* **localStorage** is a web storage API that stores data with no expiration date. Data persists even after the browser is closed and reopened.
* When a user logs in, their credentials are checked against a user database (in production, this is a server-side check).
* Upon successful login, user details such as email and login status can be stored in localStorage.
* On subsequent page loads, the app checks localStorage to confirm if the user is logged in and grants access accordingly.
* This approach improves user experience by preventing the need to log in repeatedly.

**Cart Management with localStorage**

An e-commerce or service app typically provides a shopping cart to select items before checkout. Key functionalities include:

* **Adding items:** Users can select services or products to add to their cart.
* **Updating quantities:** Users can change the quantity of each item.
* **Removing items:** Users can delete items from their cart.
* The **cart state** needs to be maintained across page reloads or navigation for seamless experience.
* Using **localStorage** for the cart data ensures persistence without requiring backend storage.
* Whenever the cart changes, the data is saved to localStorage, and upon page load, the cart is reloaded from localStorage to restore the user's selections.

**Validation and UX Enhancements**

* Forms should validate inputs such as ensuring required fields are filled, and emails are correctly formatted before submission.
* Proper validation reduces errors, enhances data integrity, and improves user satisfaction.
* Responsive feedback such as error messages for invalid login attempts improves usability.

**Limitations and Security Considerations**

* Storing sensitive data like passwords in localStorage is not secure for production apps — this approach is only suitable for demos or learning purposes.
* Real-world applications must use secure authentication with backend verification and session handling.
* localStorage data can be accessed and manipulated by users, so it should never store critical secrets or unencrypted sensitive data.

CODE:

<!-- Login Form -->

<form id="loginForm">

<h2>Login</h2>

<input type="email" id="loginEmail" placeholder="Email" required />

<input type="password" id="loginPassword" placeholder="Password" required />

<button type="submit">Login</button>

<div id="loginError" style="color:red;"></div>

</form>

<!-- Cart -->

<h2>Your Cart</h2>

<div id="cartItems"></div>

<div>Total: $<span id="cartTotal">0</span></div>

// Mock user database (in real app, validate with server)

const users = [

{ email: 'user@example.com', password: 'password123', name: 'John Doe' }

];

// Global variables

let loggedInUser = null;

let cart = [];

// Check login state on page load

window.onload = () => {

const savedUser = localStorage.getItem('weatherWiseUser');

if (savedUser) {

loggedInUser = JSON.parse(savedUser);

redirectToDashboard();

}

loadCartFromLocalStorage();

renderCart();

};

// Login form handler

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

e.preventDefault();

const email = document.getElementById('loginEmail').value.trim();

const password = document.getElementById('loginPassword').value;

const errorDiv = document.getElementById('loginError');

errorDiv.textContent = '';

if (!email || !password) {

errorDiv.textContent = 'All fields are required.';

return;

}

if (!validateEmail(email)) {

errorDiv.textContent = 'Invalid email format.';

return;

}

// Authenticate user

const user = users.find(u => u.email === email && u.password === password);

if (!user) {

errorDiv.textContent = 'Invalid email or password.';

return;

}

// Save user data to localStorage to persist login

loggedInUser = { email: user.email, name: user.name };

localStorage.setItem('weatherWiseUser', JSON.stringify(loggedInUser));

redirectToDashboard();

});

// Redirect to homepage or dashboard (mock)

function redirectToDashboard() {

alert(`Welcome back, ${loggedInUser.name}! Redirecting to dashboard...`);

// Example: window.location.href = 'dashboard.html';

// For demo, just hide login form

document.getElementById('loginForm').style.display = 'none';

}

// Email validation function

function validateEmail(email) {

const re = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

return re.test(email);

}

// CART FUNCTIONALITY

// Example products (weather services/features)

const products = [

{ id: 1, name: "Premium Weather Alerts", price: 10 },

{ id: 2, name: "Ad-Free Experience", price: 5 },

{ id: 3, name: "Detailed Forecasts", price: 8 }

];

// Add item to cart

function addToCart(productId) {

const product = products.find(p => p.id === productId);

if (!product) return;

const cartItem = cart.find(item => item.id === productId);

if (cartItem) {

cartItem.quantity++;

} else {

cart.push({ ...product, quantity: 1 });

}

saveCartToLocalStorage();

renderCart();

}

// Remove item from cart

function removeFromCart(productId) {

cart = cart.filter(item => item.id !== productId);

saveCartToLocalStorage();

renderCart();

}

// Update item quantity

function updateQuantity(productId, quantity) {

const cartItem = cart.find(item => item.id === productId);

if (cartItem) {

cartItem.quantity = quantity > 0 ? quantity : 1;

}

saveCartToLocalStorage();

renderCart();

}

// Save cart data to localStorage

function saveCartToLocalStorage() {

localStorage.setItem('weatherWiseCart', JSON.stringify(cart));

}

// Load cart data from localStorage

function loadCartFromLocalStorage() {

const savedCart = localStorage.getItem('weatherWiseCart');

if (savedCart) {

cart = JSON.parse(savedCart);

}

}

// Render cart UI

function renderCart() {

const cartDiv = document.getElementById('cartItems');

cartDiv.innerHTML = '';

let total = 0;

cart.forEach(item => {

total += item.price \* item.quantity;

const itemDiv = document.createElement('div');

itemDiv.style.marginBottom = '10px';

itemDiv.innerHTML = `

<strong>${item.name}</strong> - $${item.price}

<input type="number" min="1" value="${item.quantity}" style="width: 50px;" onchange="updateQuantity(${item.id}, this.value)" />

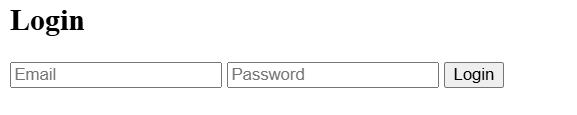
<button onclick="removeFromCart(${item.id})">Remove</button>

`;

cartDiv.appendChild(itemDiv);

});

document.getElementById('cartTotal').textContent = total.toFixed(2);

}

EXPERIMENT 7

|  |  |
| --- | --- |
| 7. | PHP   1. Develop a PHP script to handle user registration for the Coffee Shop website. The script should accept input from users for their name, email address, password, etc. (all required fields for registration). 2. Implement error handling to notify users of any issues during registration, such as validation errors. 3. Provide feedback to the user upon successful registration, either through a confirmation message or a redirect to a login page. |

**Objective**

* To create a server-side PHP script that processes user registration data securely.
* To validate the input data, ensuring all required fields (name, email, password) are submitted correctly.
* To check for common registration errors such as missing fields or duplicate email addresses.
* To store new user data in a database securely, including hashing passwords for safety.
* To provide user feedback through error messages or success confirmation and optionally redirect the user to the login page after successful registration.

**Theory**

**User Registration and PHP**

User registration involves collecting personal details (like name, email, and password) from the client and saving them to a database for authentication and personalized services later.

**Key points:**

* **Input validation:** Server-side validation ensures that all required fields are filled out correctly, which complements client-side validation and protects against malicious or incorrect data.
* **Password security:** Passwords must never be stored in plain text. PHP provides password\_hash() and password\_verify() functions to handle secure password hashing and checking.
* **Database interaction:** PHP interacts with a backend database (like MySQL) to store user information securely.
* **Error handling:** Proper error handling informs the user if something goes wrong, e.g., missing fields, invalid email format, or duplicate accounts.
* **Feedback:** After successful registration, users should be informed and often redirected to the login page.

CODE:

<?php

// Database connection parameters

$servername = "localhost";

$username = "root";

$password = ""; // Your DB password

$dbname = "weatherwise\_db";

// Create connection

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

// Check connection

if ($conn->connect\_error) {

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

}

// Initialize variables for form data and errors

$name = $email = $pass = "";

$errors = [];

if ($\_SERVER["REQUEST\_METHOD"] == "POST") {

// Sanitize input data

$name = trim($\_POST['name']);

$email = trim($\_POST['email']);

$pass = $\_POST['password'];

// Validate inputs

if (empty($name)) {

$errors[] = "Name is required.";

}

if (empty($email)) {

$errors[] = "Email is required.";

} elseif (!filter\_var($email, FILTER\_VALIDATE\_EMAIL)) {

$errors[] = "Invalid email format.";

}

if (empty($pass)) {

$errors[] = "Password is required.";

} elseif (strlen($pass) < 6) {

$errors[] = "Password must be at least 6 characters.";

}

// Check if email already exists

$stmt = $conn->prepare("SELECT id FROM users WHERE email = ?");

$stmt->bind\_param("s", $email);

$stmt->execute();

$stmt->store\_result();

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

$errors[] = "Email is already registered.";

}

$stmt->close();

// If no errors, insert new user

if (empty($errors)) {

$hashedPassword = password\_hash($pass, PASSWORD\_DEFAULT);

$stmt = $conn->prepare("INSERT INTO users (name, email, password) VALUES (?, ?, ?)");

$stmt->bind\_param("sss", $name, $email, $hashedPassword);

if ($stmt->execute()) {

// Registration successful

echo "Registration successful! <a href='login.php'>Click here to login</a>.";

} else {

echo "Error: " . $stmt->error;

}

$stmt->close();

} else {

// Display errors

foreach ($errors as $error) {

echo "<p style='color:red;'>$error</p>";

}

}

}$conn->close();

?>

EXPERIMENT 8.

|  |  |
| --- | --- |
| 8. | PHP   1. Develop a PHP script to handle user login for the Coffee Shop website. The script should accept input from users for their login credentials. (all required fields for login). 2. Provide feedback to the user upon successful login, either through a confirmation message or a redirect to a welcome page. 3. Implement error handling to notify users of login failures due to incorrect credentials or other errors. 4. Provide feedback to the user upon successful login, either through a welcome user name message or a redirect to a home page. |

**Objective**

* To create a **secure and functional login system** for the Weather Wise app using PHP and MySQL.
* To **validate user credentials** during login and provide appropriate feedback.
* To **redirect the user to the homepage or dashboard** upon successful login.
* To **implement session management** for maintaining user authentication status.

**Theory**

**User Login Using PHP**

User login systems are fundamental to web applications where users have personalized access. This system allows registered users to authenticate themselves and gain access to protected parts of the website.

**Key Concepts:**

1. **Form Handling in PHP:**  
   Capturing login input (email & password) via POST method.
2. **Input Validation:**  
   Ensuring fields are not empty and email is in correct format.
3. **Secure Authentication:**
   * Passwords should be compared using password\_verify() function.
   * SQL injection should be prevented using **prepared statements**.
4. **Session Management:**  
   PHP $\_SESSION is used to keep the user logged in during the session.
5. **Error Handling & Feedback:**  
   Display appropriate messages for login success or failure (like incorrect credentials).

**CODE: PHP Login Script for Weather Wise App**

**<?php**

**session\_start();**

**// Database connection**

**$servername = "localhost";**

**$username = "root";**

**$password = ""; // Your database password**

**$dbname = "weatherwise\_db";**

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

**// Check DB connection**

**if ($conn->connect\_error) {**

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

**}**

**// Initialize error**

**$error = "";**

**if ($\_SERVER["REQUEST\_METHOD"] == "POST") {**

**// Get form values**

**$email = trim($\_POST["email"]);**

**$password = $\_POST["password"];**

**// Check for empty fields**

**if (empty($email) || empty($password)) {**

**$error = "Please enter both email and password.";**

**} else {**

**// Prepare SQL to check user**

**$stmt = $conn->prepare("SELECT id, name, password FROM users WHERE email = ?");**

**$stmt->bind\_param("s", $email);**

**$stmt->execute();**

**$stmt->store\_result();**

**if ($stmt->num\_rows > 0) {**

**// Bind result**

**$stmt->bind\_result($id, $name, $hashed\_password);**

**$stmt->fetch();**

**if (password\_verify($password, $hashed\_password)) {**

**// Login success**

**$\_SESSION["user\_id"] = $id;**

**$\_SESSION["user\_name"] = $name;**

**$\_SESSION["email"] = $email;**

**// Redirect to homepage**

**header("Location: home.php");**

**exit;**

**} else {**

**$error = "Invalid password. Please try again.";**

**}**

**} else {**

**$error = "No user found with this email.";**

**}**

**$stmt->close();**

**}**

**}**

**$conn->close();**

**?>**

**<!-- Basic Login Form -->**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>Login - Weather Wise</title>**

**</head>**

**<body>**

**<h2>User Login - Weather Wise</h2>**

**<?php if (!empty($error)) echo "<p style='color:red;'>$error</p>"; ?>**

**<form method="POST" action="">**

**<label>Email:</label><br>**

**<input type="email" name="email" required><br><br>**

**<label>Password:</label><br>**

**<input type="password" name="password" required><br><br>**

**<input type="submit" value="Login">**

**</form>**

**</body>**

**</html>**

EXPERIMENT:9

|  |  |
| --- | --- |
| 9. | PHP and MySQL   1. Develop a PHP script that allows users to manage their shopping cart for an e-commerce website (e.g., a Coffee Shop store). The script should allow users to add items to their cart, view their cart contents, and remove items if needed. 2. Develop a PHP script to manage the shopping cart for an e-commerce website (e.g., a Coffee Shop store) using MySQL. This script should allow users to add items to their cart, view their cart contents, and remove items from the cart. The cart data should be stored in the MySQL database to allow persistence across sessions. |

**Objective**

* To create a **user-personalized experience** using PHP and MySQL, similar to a shopping cart.
* Instead of adding products to a cart, users can **save favorite cities or weather locations** to quickly access later.
* Provide functionality to **add, view, and remove** saved cities using **PHP and MySQL**.

**Theory**

In an e-commerce system, a cart allows users to temporarily store selected items for purchase. For a **weather app**, this concept can be translated into a system where users:

* Add **cities** to their favorites list.
* View the list of **saved cities** with weather information.
* Remove **unwanted cities** from the list.

**Key Concepts:**

* **PHP**: Handles requests such as adding, deleting, and viewing cities.
* **MySQL**: Stores user preferences persistently (table like favorite\_cities).
* **Session/User ID**: Used to associate favorites with a specific user.
* **CRUD operations**: Implemented via INSERT, SELECT, and DELETE queries.

CODE:

1. add\_city.php – Add city to favorites

<?php

session\_start();

include 'db\_connect.php'; // Your DB connection file

if (isset($\_POST['city'])) {

$user\_id = $\_SESSION['user\_id']; // Logged-in user

$city = trim($\_POST['city']);

$stmt = $conn->prepare("INSERT INTO favorite\_cities (user\_id, city\_name) VALUES (?, ?)");

$stmt->bind\_param("is", $user\_id, $city);

if ($stmt->execute()) {

echo "City added to favorites!";

} else {

echo "Error adding city.";

}

$stmt->close();

}

?>

1. view\_cities.php – View saved cities

<?php

session\_start();

include 'db\_connect.php';

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

$sql = "SELECT id, city\_name FROM favorite\_cities WHERE user\_id = ?";

$stmt = $conn->prepare($sql);

$stmt->bind\_param("i", $user\_id);

$stmt->execute();

$result = $stmt->get\_result();

echo "<h3>Your Saved Cities</h3>";

echo "<ul>";

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

echo "<li>" . $row['city\_name'] . " <a href='remove\_city.php?id=" . $row['id'] . "'>Remove</a></li>";

}

echo "</ul>";

$stmt->close();

?>

1. remove\_city.php – Remove city from favorites

<?php

session\_start();

include 'db\_connect.php';

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

$city\_id = $\_GET['id'];

$stmt = $conn->prepare("DELETE FROM favorite\_cities WHERE id = ? AND user\_id = ?");

$stmt->bind\_param("ii", $city\_id, $user\_id);

if ($stmt->execute()) {

echo "City removed.";

} else {

echo "Failed to remove city.";

}

$stmt->close();

?>

db\_connect.php (Database Connection)

<?php

$servername = "localhost";

$username = "root";

$password = "";

$dbname = "weatherwise\_db";

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

if ($conn->connect\_error) {

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

}

?>  
**Output / User Experience**

* A logged-in user can:
  + Add cities to their **"Favorites"**.
  + View and manage saved cities.
  + Remove cities they no longer need.
* This system works like a cart, but adapted to **weather-tracking use cases**.

EXPERIMENT 10

|  |  |
| --- | --- |
| 10. | PHP and MySQL   1. Develop a PHP script to handle the checkout process for users who are ready to complete their purchase. The script should process the cart data and provide feedback to the user upon successful or failed checkout. 2. Develop a PHP script that processes the checkout process for users who are ready to complete their purchase, integrating the MySQL database for handling user and order information. The script should validate user input, process the cart data, and provide feedback upon successful or failed checkout. |

**Objective**

* To allow users to **save their favorite cities** for quick weather updates.
* To create a PHP + MySQL system that allows:
  + Adding a city to a user’s favorites (similar to adding to cart).
  + Viewing saved cities (similar to viewing cart contents).
  + Removing saved cities (similar to removing cart items).
* Handle **"checkout"** as a **subscription confirmation** for selected weather alerts for saved cities.

**THEORY**

**PHP & MySQL for Weather Wise – Adapted Use Case:**

* **MySQL Database Tables:**
  + users – Stores user information.
  + saved\_locations – Stores user's favorite cities.
  + subscriptions – Stores user-selected preferences for weather alerts.

**Features Equivalent to E-Commerce:**

| **E-Commerce Feature** | **Weather Wise Equivalent** |
| --- | --- |
| Add to Cart | Save Favorite City |
| View Cart | View Saved Cities |
| Remove from Cart | Remove a City from Favorites |
| Checkout | Subscribe to Weather Alerts |

A. PHP Script – Save Favorite Cities (Add/View/Remove)  
<?php

session\_start();

$user\_id = $\_SESSION['user\_id']; // Assume login system is working

$conn = new mysqli("localhost", "root", "", "weatherwise\_db");

// Add a city

if (isset($\_POST['add\_city'])) {

$city = $\_POST['city\_name'];

$stmt = $conn->prepare("INSERT INTO saved\_locations (user\_id, city\_name) VALUES (?, ?)");

$stmt->bind\_param("is", $user\_id, $city);

$stmt->execute();

$stmt->close();

}

// Remove a city

if (isset($\_GET['remove'])) {

$city\_id = $\_GET['remove'];

$stmt = $conn->prepare("DELETE FROM saved\_locations WHERE id = ? AND user\_id = ?");

$stmt->bind\_param("ii", $city\_id, $user\_id);

$stmt->execute();

$stmt->close();

}

// View saved cities

$result = $conn->query("SELECT id, city\_name FROM saved\_locations WHERE user\_id = $user\_id");

?>

<h2>Your Favorite Cities</h2>

<form method="POST">

<input type="text" name="city\_name" placeholder="Enter city name" required>

<input type="submit" name="add\_city" value="Add to Favorites">

</form>

<ul>

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

<li>

<?= htmlspecialchars($row['city\_name']) ?>

<a href="?remove=<?= $row['id'] ?>">Remove</a>

</li>

<?php } ?>

</ul>

B.PHP Script – Checkout Process as Subscription

<?php

session\_start();

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

$conn = new mysqli("localhost", "root", "", "weatherwise\_db");

if ($\_SERVER['REQUEST\_METHOD'] == 'POST') {

$frequency = $\_POST['frequency']; // daily, hourly, etc.

// Store user subscription preference

$stmt = $conn->prepare("INSERT INTO subscriptions (user\_id, frequency) VALUES (?, ?)");

$stmt->bind\_param("is", $user\_id, $frequency);

if ($stmt->execute()) {

echo "<p>✅ Subscribed to $frequency weather alerts!</p>";

} else {

echo "<p>❌ Subscription failed. Please try again.</p>";

}

$stmt->close();

}

?>

<h2>Subscribe to Weather Alerts</h2>

<form method="POST">

<label>Choose Alert Frequency:</label><br>

<select name="frequency" required>

<option value="hourly">Hourly</option>

<option value="daily">Daily</option>

<option value="severe">Severe Weather Only</option>

</select><br><br>

<input type="submit" value="Subscribe">

</form>

Database Schema Summary

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(50),

email VARCHAR(100),

password VARCHAR(255)

);

CREATE TABLE saved\_locations (

id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT,

city\_name VARCHAR(100),

FOREIGN KEY (user\_id) REFERENCES users(id)

);

CREATE TABLE subscriptions (

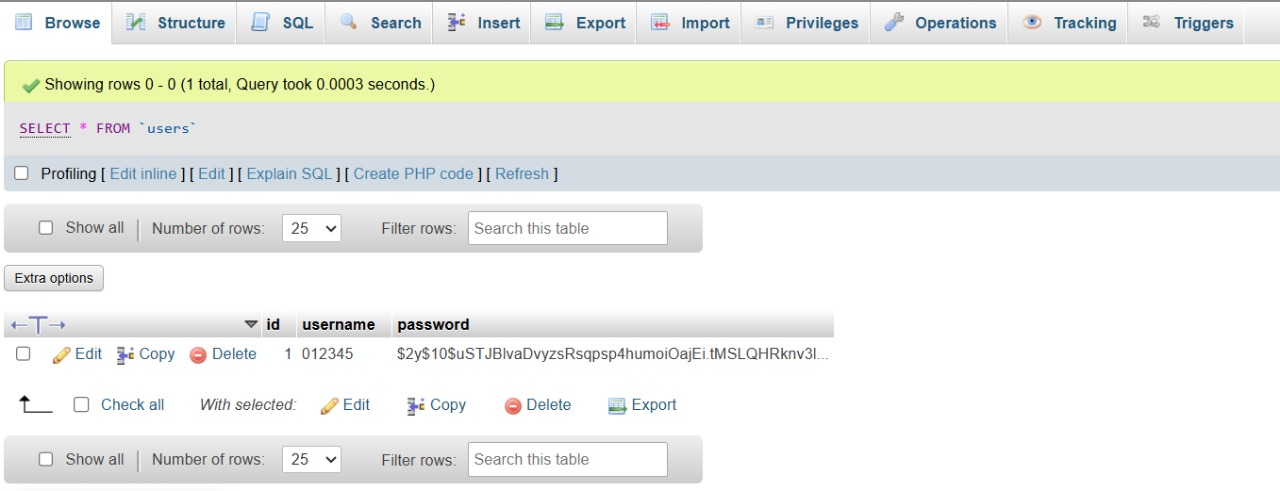
id INT AUTO\_INCREMENT PRIMARY KEY,

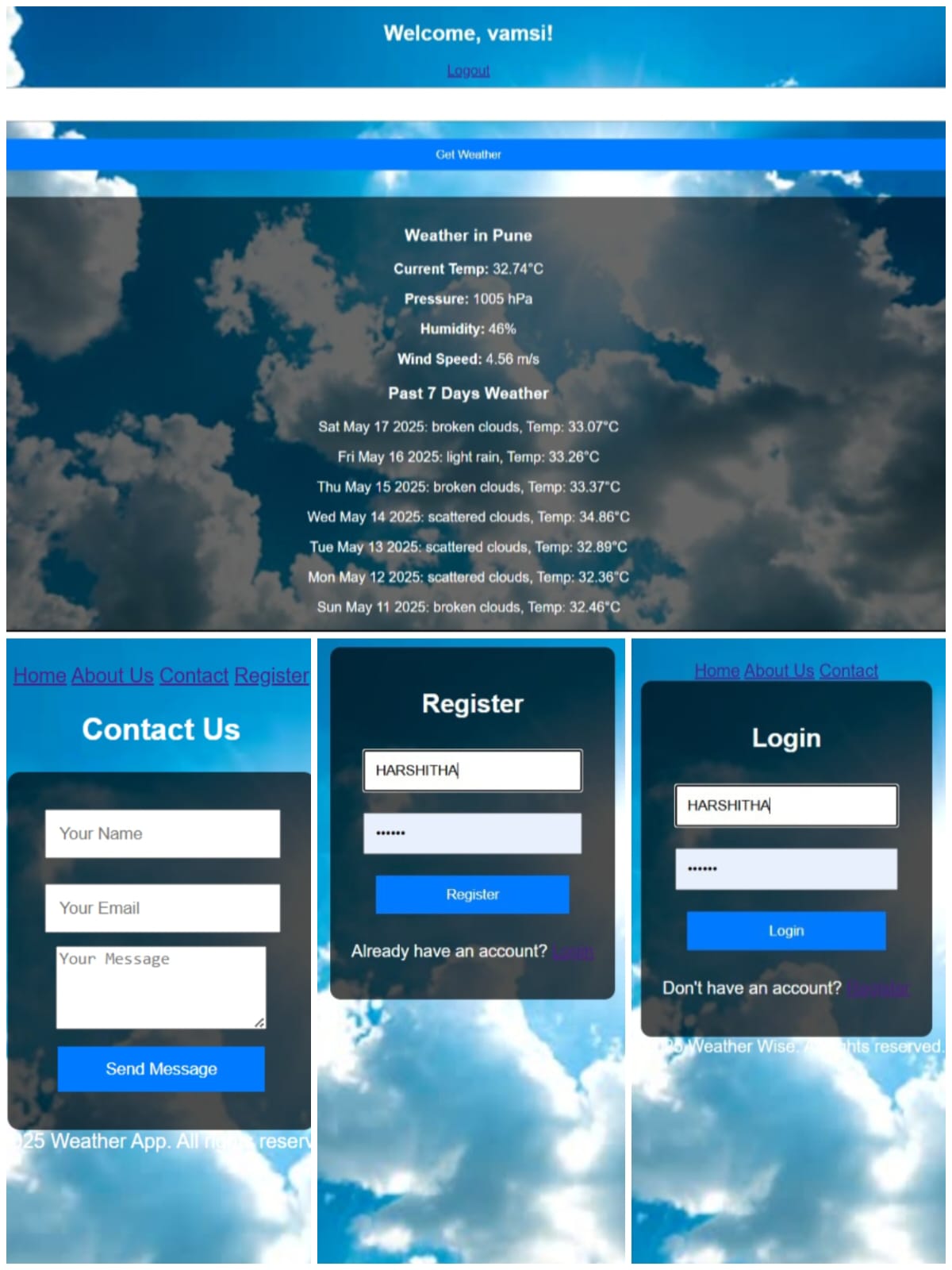
user\_id INT,

frequency VARCHAR(50),

subscribed\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES users(id)

);  


Website output  


Conclusion:

The Weather Wise Website project serves as a dynamic exercise in applying web development best practices to deliver a practical, user-friendly platform. By focusing on key elements like functionality, content organization, and visual appeal, this project ensures that users can easily access real-time weather data and forecasts, enhancing their overall experience.

Through the creation of essential pages such as the Home, Forecast, and Login/Registration, the website provides a seamless and intuitive navigation structure. Key features like dynamic weather maps, personalized notifications, and interactive widgets offer visitors a comprehensive weather experience, fostering engagement and trust.

The design direction of the Weather Wise website—leveraging a harmonious color palette, carefully selected typography, and responsive images—effectively reinforces the brand's commitment to clarity, reliability, and accessibility. With a focus on user experience, the website’s design elements help to establish a strong connection with users, making them feel confident in using the platform for daily weather updates and planning.

This assignment underscores the importance of a user-centered approach to web design—starting with defining goals and structure, followed by a thoughtful design and development process. By considering the needs of the audience and providing interactive, well-organized content, the project ensures that Weather Wise is not just a weather website, but a valuable, user-friendly tool for individuals seeking up-to-date and personalized weather information.