EXPERIMENT 1

Aim

Build responsive and interactive UIs using **Tailwind CSS**

Theory

Objective

The objective of this project is to design and implement a multi-page, fully responsive, and interactive Habit Tracker web application using **Tailwind CSS**. The focus is to demonstrate how Tailwind's utility-first approach can be used to build modern user interfaces without the need for custom CSS. The project aims to explore core concepts like layout control, dark mode toggling, responsive breakpoints, and interactive elements, all styled using Tailwind's utility classes. By building a clean and functional user experience, the goal is to showcase how Tailwind CSS simplifies UI development while maintaining visual consistency and responsiveness across different devices and screen sizes.

Introduction

In the current era of front-end web development, creating responsive, accessible, and visually appealing user interfaces is a core requirement. Traditional CSS and component-based frameworks often lead to bloated stylesheets and repetitive code.

Tailwind CSS addresses this by introducing a utility-first approach, where design is handled by composing small utility classes directly in the HTML markup. This drastically reduces custom styling and speeds up development. Tailwind also supports responsive design, pseudo-classes (like hover: and focus:), and even dark mode with minimal setup.

This project — a **Habit Tracker** application — serves as a hands-on implementation of these features. It consists of three main pages: a **Dashboard** (index.html) with a welcoming UI and navigation; a **Habits** page (habits.html) where users can add and track their habits through interactive modals and checkboxes; and a **Progress** page (progress.html) which visually summarizes the completion status of habits.

The interface is designed to adapt seamlessly to both light and dark themes and remains mobile-friendly throughout. LocalStorage is used for data persistence, allowing habits to be remembered across sessions and shared among the three pages. By combining structured HTML, Tailwind utility classes, and vanilla JavaScript, the project
demonstrates the effectiveness of Tailwind CSS in developing responsive, interactive, and well-organized front-end applications.

Experiment Overview

The experiment involves building a fully responsive and interactive Habit Tracker web application using Tailwind CSS as the primary styling framework. The goal is to create a clean and modern user interface that adapts to both light and dark themes and remains consistent across devices of various screen sizes.
The project consists of three main HTML pages:

- Dashboard (index.html) – a landing page with a welcome message and navigation.
- Habits (habits.html) – a page to add and manage habits using modals and interactive checkboxes.
- Progress (progress.html) – a page that displays progress statistics and visual feedback based on habit completion.

JavaScript is used to handle interactivity and store data in the browser's localStorage. Tailwind CSS utility classes are used for layout, color schemes, spacing, responsive design, and hover effects. The result is a front-end project that visually represents habit tracking with no need for external CSS or backend support.

Tools and Technologies Used

Tool/Technology	Purpose
HTML5	Structure and content of the web pages.
Tailwind CSS	Utility-first CSS framework for styling and responsive design.
JavaScript (ES6)	Adds interactivity, manages habit data using localStorage.
LocalStorage	Browser-based storage to persist user habits across sessions and pages.
VSCode (optional)	Code editor used for writing and organizing HTML, CSS, and JS.
Live Server	Tool to preview pages locally during development (optional).

Key Outcomes

This experiment resulted in the successful development of a responsive multi-page web application using HTML, Tailwind CSS, and JavaScript. A working dark mode toggle was implemented with Tailwind's utility classes, and a dynamic modal form was created for adding habits interactively. JavaScript and localStorage were used to store and manage

habits across all pages without the need for backend support. Progress statistics and completion feedback were displayed dynamically, ensuring a smooth and user-friendly experience. The application maintained consistent design and responsiveness across different screen sizes including mobile, tablet, and desktop.

Learning Outcomes

Through this experiment, practical experience was gained in using Tailwind CSS to style real-world web interfaces without writing custom CSS. The project helped in learning how to structure multi-page applications with shared state using localStorage and understanding the implementation of dark mode using Tailwind's class-based approach. Skills were developed in building modals, dynamic lists, and interactive elements with vanilla JavaScript, along with the proper use of semantic HTML5 elements for clean code organization. Overall, this experiment built confidence in creating modern user interfaces that are responsive, accessible, and visually consistent.

1. Utility Classes:

In Tailwind CSS, **utility classes** are single-purpose, predefined CSS classes that apply a specific style directly to an element in HTML.

Each utility class is responsible for **only one styling task** (e.g., setting text color, margin, padding, background, etc.), and multiple utilities can be combined to create complex designs without writing custom CSS.

- **Key Features**

1. **Single Responsibility** – One class for one style (e.g., text-center only centers text).
2. **Composable** – Multiple classes can be applied to the same element to achieve the desired look
3. **Predefined & Consistent** – Follows Tailwind's design system and spacing scale.
4. **State Variants** – Built-in pseudo-classes like hover:

- **Example**

<button onclick="toggleTheme()" class="ml-3 px-3 py-1 bg-gray-200 dark:bg-gray-600 rounded">Toggle</button>

- Breakdown:

1. **ml-3** → Margin-left of 0.75rem (12px)
2. **px-3** → Padding left & right = 0.75rem (12px)
3. **py-1** → Padding top & bottom = 0.25rem (4px)
4. **bg-gray-200** → Light gray background (shade 200) in light mode 5. **dark:bg-gray-600** → Dark gray background (shade 600) in dark mode
6. **rounded** → Slightly rounded corners

2.Responsive Design (Breakpoints):

Responsive design means creating web layouts that automatically adjust to different screen sizes and devices.
In Tailwind CSS, this is achieved using **breakpoints** — predefined screen width ranges that trigger different styles for mobile, tablet, laptop, or desktop.

- Example:

<h2 class="text-2xl md:text-4xl font-bold mb-4">
Welcome to Your Habit Tracker
</h2>

- Breakpoints:
- text-2xl → Smaller heading on mobile.

- md:text-4xl → Bigger heading from **768px and above** (tablets & desktop

Code and explanation

1. Dark Mode Toggle Button

The **Dashboard** of the Habit Tracker greets users with a clean and minimal welcome screen, encouraging them to start tracking their habits. It features a “View Habits” button for navigation and a **"Toggle" button** that switches between light and dark themes using the toggleTheme() JavaScript function. Tailwind CSS classes dynamically change the page appearance, enhancing user comfort in different lighting conditions.

```
<script>
function toggleTheme() {
document.documentElement.classList.toggle('dark');  }
</script>
<button onclick="toggleTheme()" class="ml-3 px-3 py-1 bg-gray-200
dark:bg-gray-600 rounded">Toggle</button>
```

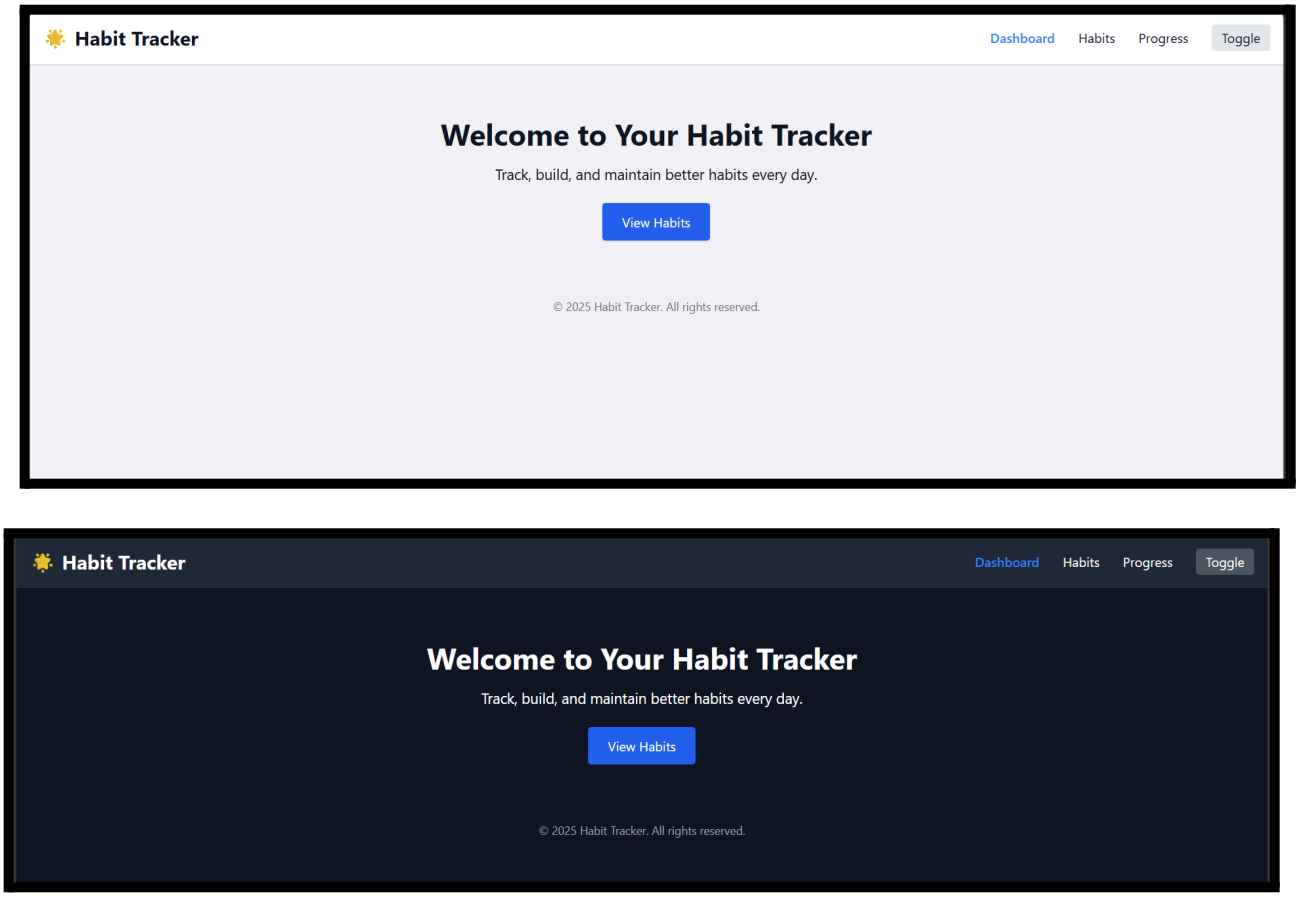


Fig 1: Home page/ Dashboard

2. Modal for Adding Habits

The image displays a **dark-themed Habit Tracker web app** with a modal popup for adding a new habit. The modal includes a text input for the habit name, a dropdown to select the frequency (Daily or Weekly), and two buttons **Add**. The modal is styled with Tailwind CSS, featuring rounded edges, dark mode support, and a clean, centered layout for better user experience.

```
 <div id="habitModal" class="fixed inset-0 flex items-center
justify-center bg-black bg-opacity-50 hidden z-50">
 <div class="bg-white dark:bg-gray-800 p-6 rounded shadow-lg w-full
max-w-md">
 <h3 class="text-xl font-semibold mb-4">Add New Habit</h3>  <input
type="text" id="habitName" placeholder="Habit name..."
class="w-full mb-3 px-3 py-2 rounded border dark:bg-gray-700
dark:border-gray-600" />
```

```
 <select id="habitFrequency" class="w-full mb-4 px-3 py-2 rounded
border dark:bg-gray-700 dark:border-gray-600">
 <option value="Daily">Daily</option>
 <option value="Weekly">Weekly</option>
 </select>
 <div class="flex justify-end space-x-2">
 <button onclick="closeModal()" class="px-4 py-2 bg-gray-300
dark:bg-gray-600 rounded">Cancel</button>
 <button onclick="addHabit()" class="px-4 py-2 bg-blue-600
text-white rounded hover:bg-blue-700">Add</button>
 </div>
 </div>  </div>
```

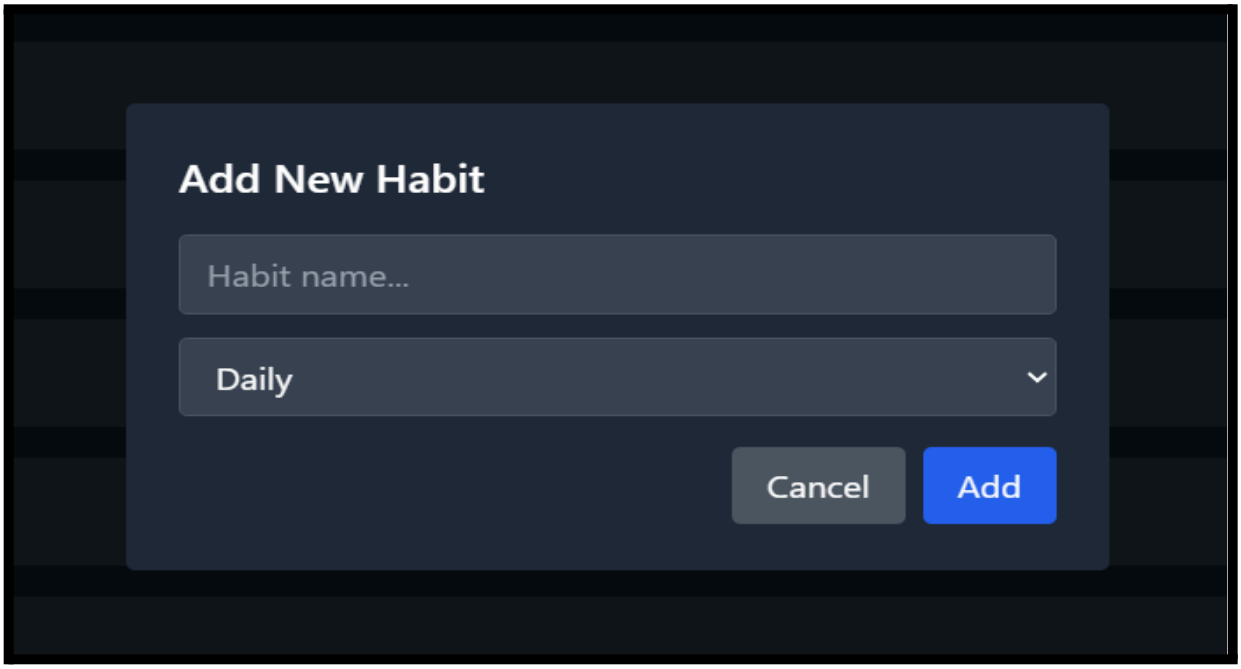


Fig 2.1: Adding habits

3. Rendering Habits on the Habits Page

The updated Habit Tracker interface displays a **list of user habits** with a checkbox for tracking completion. Each habit shows its name and frequency (e.g., “Drink Water - Daily” or “Walk - Weekly”). The renderHabits() function dynamically creates and styles each habit entry using JavaScript, with visual feedback (✅ or ⬜) and checkboxes that reflect and update the completion status. The layout is styled using Tailwind CSS, ensuring a clean and modern dark

theme.

```
function renderHabits() {
 const list = document.getElementById("habitList");
list.innerHTML = "";
 habits.forEach((habit, index) => {
 const li = document.createElement("li");
 li.className = "bg-white dark:bg-gray-800 p-4 rounded shadow flex
items-center justify-between";
 li.innerHTML = `
 <span>${habit.completed ? '✅' : '⬜'} ${habit.name} -
${habit.frequency}</span>
 <input type="checkbox" ${habit.completed ? "checked" : ""}
onchange="toggleComplete(${index})" />
 `;
 list.appendChild(li);
                                        });
```

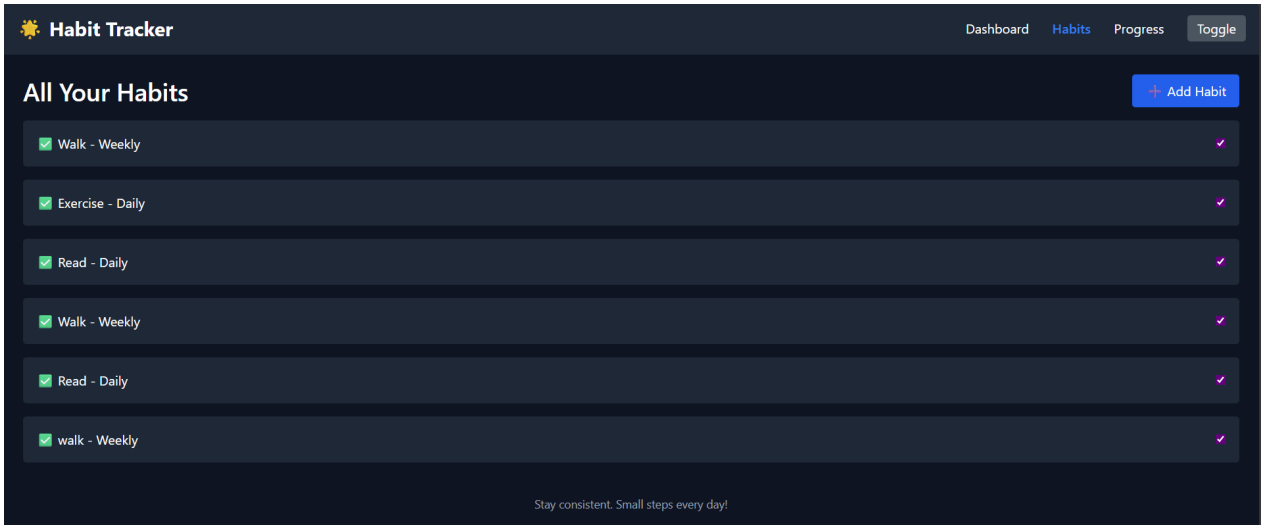


Fig 2.2: All Habits

4. Progress Calculation Logic

The **Progress** page of the Habit Tracker visually shows the percentage of completed habits. The code calculates progress by dividing the number of completed habits by the total number, then displays the result dynamically. If all habits are completed, a celebratory message like "🎉 All Habits Completed!" appears. Styled with Tailwind CSS, this section provides users with motivation and visual feedback to track their consistency in real-time

```
. let completed = habits.filter(h => h.completed).length;
```

```
 let total = habits.length;

 if (total === 0) {
 progressText.textContent = "No habits yet. Add some to start
tracking!";
 } else {
 let percentage = Math.round((completed / total) * 100);
progressText.textContent = "You have completed ${percentage}% of your
habits ��";
```

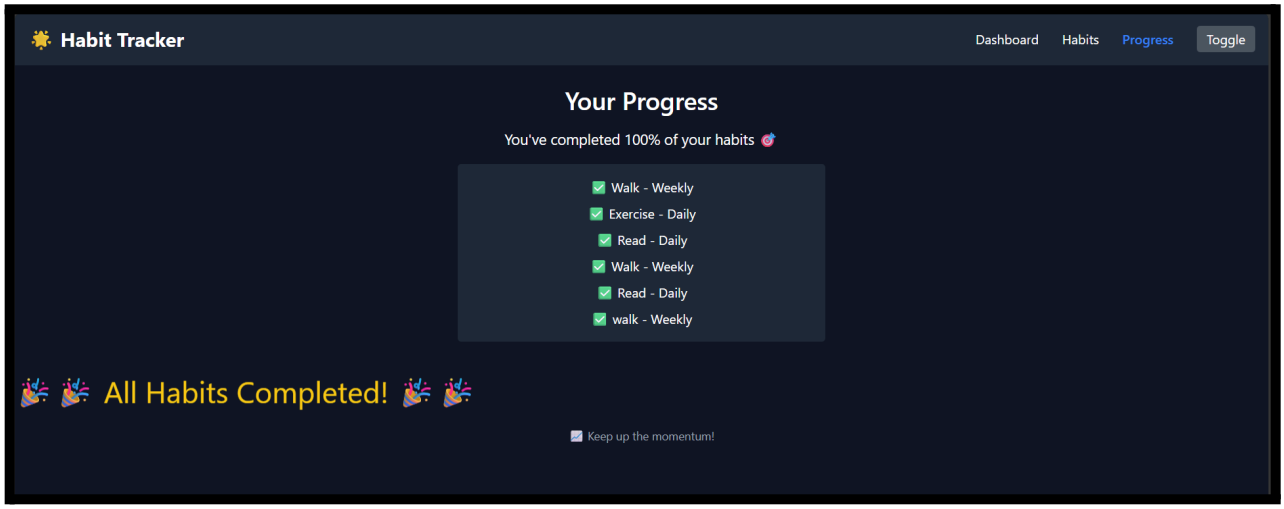
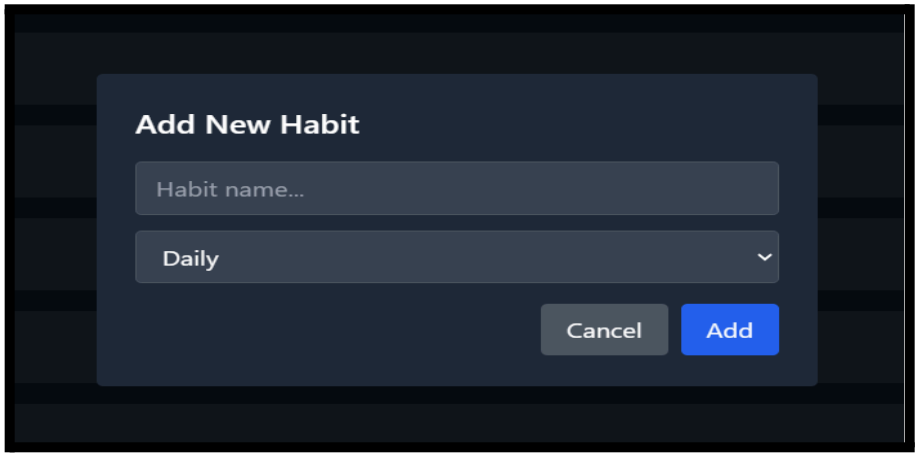


Fig 3: Progress Page

5. Adding Habits with JavaScript

```
function addHabit() {
 const name =
document.getElementById('habitName').value.trim();
 const frequency =
document.getElementById('habitFrequency').value;
 if (name !== '') {
 habits.push({ name, frequency, completed: false });
localStorage.setItem("habits", JSON.stringify(habits));
closeModal();
```

```
renderHabits();
}
}
```



- **Git pages(30% extra)**

  Definition
  GitHub Pages is a free web hosting service provided by GitHub that allows developers to host static websites (HTML, CSS, JavaScript) directly from their GitHub repository. It is widely used by students, professionals, and open-source contributors to deploy projects, portfolios, and documentation.

  Key Features

  1. Free Hosting – No cost for hosting websites.
  2. Easy Setup – Just push code to GitHub and enable GitHub Pages.
  3. Custom Domain Support – Developers can add their own domain name.
  4. Automatic Updates – Any new commit pushed to the selected branch is instantly reflected live.
  5. Secure & Reliable – Hosted with HTTPS and powered by GitHub's global servers.

  How it Works

  - A repository is created on GitHub and project files are uploaded.
  - From Repository → Settings → Pages, the branch (e.g., `main`) and folder (`root` or `/docs`) is selected.

- GitHub generates a public URL (like `https://username.github.io/projectname`).
- Users can directly open this URL to view the live website.

Example in this Experiment

In this Habit Tracker project, after development was completed, the code was pushed to a GitHub repository and deployed using GitHub Pages. Now, the application is live and can be accessed by anyone without running it locally.

Live Link of the Project:

https://leenahinduja.github.io/HabitTracker/

Conclusion:

This experiment successfully demonstrated the development of a responsive and interactive Habit Tracker application using Tailwind CSS and JavaScript. The project included features such as dark mode, dynamic modals, habit tracking, and progress visualization, all implemented without the need for external CSS or backend support. By deploying the application on GitHub Pages, the project became accessible online with a live link, ensuring ease of access, real-time usability, and global availability. Overall, the experiment highlighted how Tailwind CSS simplifies UI development and how GitHub Pages provides a simple, free, and reliable platform to host and share web projects.