International University

School of Computer Science and Engineering

Web Application Development

Laboratory

IT093IU

Lab #3_2

Submitted by

Nguyễn Hồng Ngọc Hân - ITCSIU22229

Task 2.1: Weather Dashboard

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Weather Dashboard</title>
    <style>
      * {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
      }
      body {
        font-family: Arial, sans-serif;
        background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
        min-height: 100vh;
        padding: 20px;
      }
      .container {
        max-width: 900px;
        margin: 0 auto;
      }
      .search-box {
        background: white;
        padding: 20px;
        border-radius: 10px;
        margin-bottom: 20px;
        display: flex;
        flex-direction: column;
        gap: 10px;
      }
      .search-box input {
        padding: 12px;
        border: 2px solid #ddd;
        border-radius: 4px;
        font-size: 16px;
      }
      .search-box button {
        width: 28%;
        padding: 12px;
        background: #667eea;
        color: white;
        border: none;
        border-radius: 4px;
```

```
      cursor: pointer;
      font-size: 16px;
    }
    .weather-card {
      background: white;
      padding: 30px;
      border-radius: 10px;
      box-shadow: 0 10px 30px rgba(0, 0, 0, 0.2);
      margin-bottom: 20px;
    }
    .current-weather {
      display: flex;
      justify-content: space-between;
      align-items: center;
    }
    .temp-display {
      font-size: 48px;
      font-weight: bold;
    }
    .forecast-grid {
      display: grid;
      grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));
      gap: 15px;
      margin-top: 20px;
    }
    .forecast-item {
      background: #f8f9fa;
      padding: 15px;
      border-radius: 8px;
      text-align: center;
    }
    .loading {
      text-align: center;
      padding: 40px;
      font-size: 18px;
      color: white;
    }
    .error {
      background: #dc3545;
```

```
      color: white;
      padding: 15px;
      border-radius: 8px;
      margin-bottom: 20px;
    }
    .recent-searches {
      display: flex;
      gap: 10px;
      margin-top: 10px;
      flex-wrap: wrap;
    }
    .recent-city {
      padding: 5px 15px;
      background: #e9ecef;
      border-radius: 20px;
      cursor: pointer;
      font-size: 14px;
    }
    .recent-city:hover {
      background: #dee2e6;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1 style="color: white; text-align: center; margin-bottom: 30px">
      Weather Dashboard
    </h1>

    <div class="search-box">
      <input
        type="text"
        id="cityInput"
        placeholder="Enter city name..."
        onkeypress="if(event.key==='Enter') searchWeather()"
      />
      <button onclick="searchWeather()">Search</button>
      <div class="recent-searches" id="recentSearches"></div>
    </div>
```

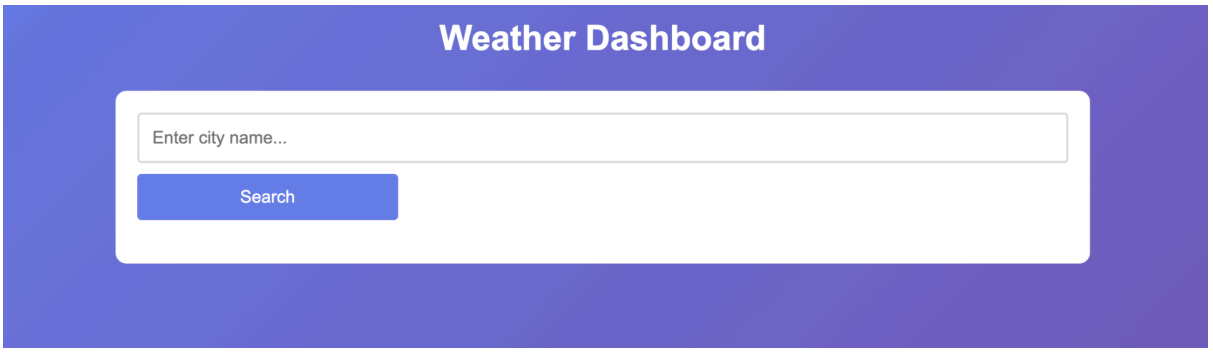
```
    <div id="errorMessage"></div>
    <div id="weatherDisplay"></div>
    <div id="forecastDisplay"></div>
  </div>

  <script>
    const API_KEY = "32d0b938a15dd55ba902c3ec35725355";
    const weatherDisplay = document.getElementById("weatherDisplay");
    const forecastDisplay = document.getElementById("forecastDisplay");
    const errorMessage = document.getElementById("errorMessage");
    const recentSearches = document.getElementById("recentSearches");

    const weatherEmojis = {
      Clear: "☀️",
      Clouds: "☁️",
      Rain: "🌧️",
      Drizzle: "🌦️",
      Thunderstorm: "⛈️",
      Snow: "❄️",
      Mist: "🌫️",
      Smoke: "🌫️",
      Haze: "🌫️",
      Dust: "🌫️",
      Fog: "🌫️",
    };

    async function fetchWeather(city) {
      const res = await fetch(
        `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${API_KEY}&units=metric`
      );
      if (!res.ok) throw new Error("City not found");
      return res.json();
    }
```

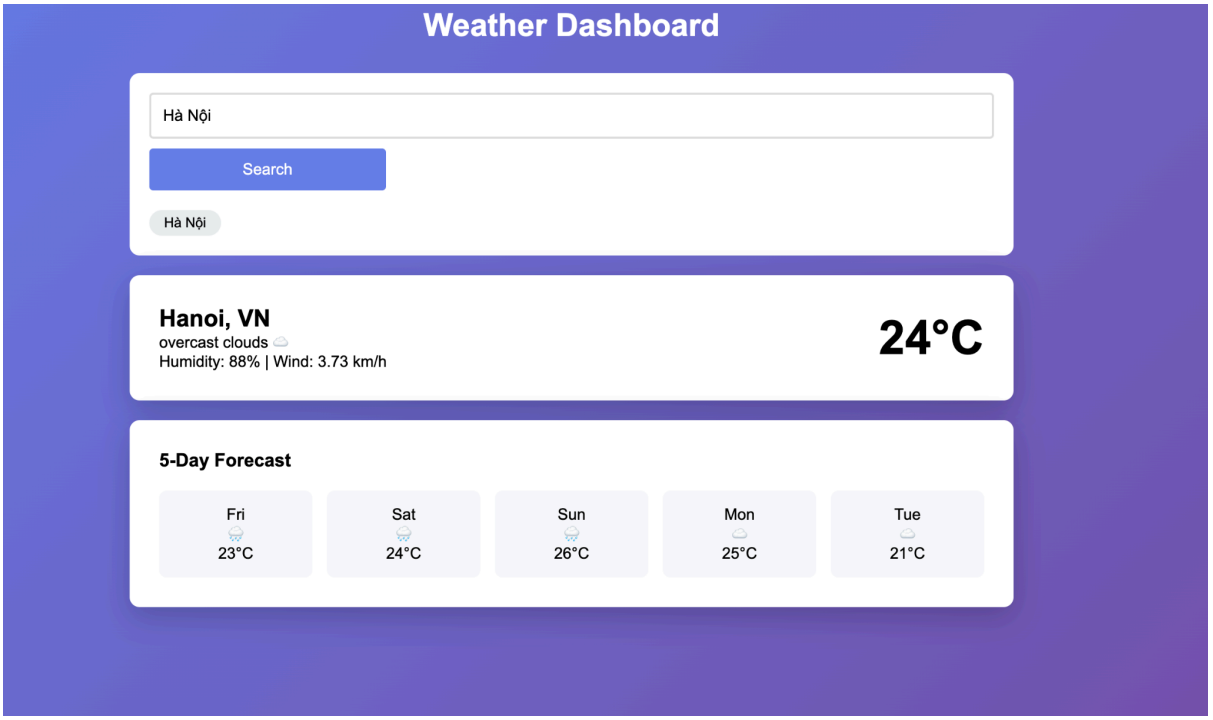
```
157       async function fetchForecast(city) {
158         const res = await fetch(
159           `https://api.openweathermap.org/data/2.5/forecast?q=${city}&appid=${API_KEY}&units=metric`
160         );
161         if (!res.ok) throw new Error("Forecast not found");
162         return res.json();
163       }
164
165       function displayWeather(data) {
166         const weather = data.weather[0];
167         const icon = weatherEmojis[weather.main] || "🌈";
168
169         weatherDisplay.innerHTML = `
170         <div class="weather-card">
171           <div class="current-weather">
172             <div>
173               <h2>${data.name}, ${data.sys.country}</h2>
174               <p>${weather.description} ${icon}</p>
175               <p>Humidity: ${data.main.humidity}% | Wind: ${
176           data.wind.speed
177         } km/h</p>
178             </div>
179             <div class="temp-display">${Math.round(data.main.temp)}°C</div>
180           </div>
181         </div>
182       `;
183       }
184
185       function displayForecast(data) {
186         const filtered = data.list.filter((item) =>
187           item.dt_txt.includes("12:00:00")
188         );
189         let forecastHTML = `<div class="weather-card"><h3>5-Day Forecast</h3>
190         <div class="forecast-grid">`;
```

```
      filtered.forEach((item) => {
        const date = new Date(item.dt_txt);
        const day = date.toLocaleDateString("en-US", { weekday: "short" });
        const weather = item.weather[0];
        const icon = weatherEmojis[weather.main] || "🌈";

        forecastHTML += `
        <div class="forecast-item">
          <p>${day}</p>
          <p>${icon}</p>
          <p>${Math.round(item.main.temp)}°C</p>
        </div>
      `;
      });

      forecastHTML += `</div></div>`;
      forecastDisplay.innerHTML = forecastHTML;
    }

    async function searchWeather() {
      const city = document.getElementById("cityInput").value.trim();
      if (!city) return showError("Please enter a city name.");
      clearError();
      weatherDisplay.innerHTML = `<div class="loading">Loading...</div>`;
      forecastDisplay.innerHTML = "";

      try {
        const weatherData = await fetchWeather(city);
        const forecastData = await fetchForecast(city);
        displayWeather(weatherData);
        displayForecast(forecastData);
        saveRecentSearch(city);
      } catch (err) {
        showError(err.message);
        weatherDisplay.innerHTML = "";
        forecastDisplay.innerHTML = "";
      }
    }
```

```
    function saveRecentSearch(city) {
      let searches = JSON.parse(localStorage.getItem("recentCities")) || [];
      searches = [
        city,
        ...searches.filter((c) => c.toLowerCase() !== city.toLowerCase()),
      ];
      if (searches.length > 5) searches.pop();
      localStorage.setItem("recentCities", JSON.stringify(searches));
      loadRecentSearches();
    }

    function loadRecentSearches() {
      const searches = JSON.parse(localStorage.getItem("recentCities")) || [];
      recentSearches.innerHTML = searches
        .map(
          (c) =>
            `<div class="recent-city" onclick="searchCity('${c}')">${c}</div>`
        )
        .join("");
    }

    function searchCity(city) {
      document.getElementById("cityInput").value = city;
      searchWeather();
    }

    function showError(msg) {
      errorMessage.innerHTML = `<div class="error">${msg}</div>`;
    }

    function clearError() {
      errorMessage.innerHTML = "";
    }

    // Initialize
    loadRecentSearches();
  </script>
</body>
</html>
```

Output:





Explain:

1. fetchWeather(city)

This function fetches current weather data for a given city using the OpenWeatherMap API. It calls the /weather endpoint with the city name and your API key. If the response is invalid (for example, the city doesn't exist), it throws an error "City not found." If successful, it returns the data as a JSON object containing temperature, humidity, wind speed, and weather conditions.

2. fetchForecast(city)

This function works similarly to fetchWeather() but retrieves the 5-day forecast (in 3-hour intervals) from the /forecast endpoint. If the request

fails, it throws "Forecast not found." The returned JSON includes a list of weather predictions for upcoming days.

3. displayWeather(data)
This function takes the current weather data and displays it on the page. It extracts details such as the city name, country, description, humidity, wind speed, and temperature, then inserts this information into the weatherDisplay element as HTML. It also adds an emoji (☀️, ☁️, 🌧️, etc.) corresponding to the weather type.

4. displayForecast(data)
This function displays the 5-day forecast. It filters the forecast list to show only entries around noon (12:00:00) each day, then creates a grid of forecast cards with the weekday name, emoji icon, and temperature. Finally, it updates the forecastDisplay section with this HTML content.

5. searchWeather()
This is the main control function.
When the user clicks “Search” or presses Enter:
It gets the city name from the input box. If empty, it shows an error message. Otherwise, it displays a “Loading…” message and clears any old results. Then it calls both fetchWeather() and fetchForecast() to get the data. If successful, it calls displayWeather() and displayForecast() to show results. It also calls saveRecentSearch() to store the city name for later quick access. If any error occurs, it displays the error message using showError().

6. saveRecentSearch(city)
This function saves the recently searched city into the browser’s localStorage. It ensures there are no duplicates, limits the list to 5 cities, and then updates the “recent searches” section by calling loadRecentSearches().

7. loadRecentSearches()
This function retrieves the stored cities from localStorage and displays them as clickable buttons. When clicked, each city triggers a new search via searchCity().

8. searchCity(city)
This function allows users to quickly re-search a city by clicking it from the recent searches list. It fills the input box with that city name and calls searchWeather() again.

9. showError(msg)
Displays an error message (for example, “City not found”) inside the #errorMessage element, styled in red for visibility.

10. clearError()
Removes any previous error messages from the screen before starting a new search.

11. loadRecentSearches(); (Initialization)
This line runs automatically when the page loads, ensuring previously saved cities appear in the “Recent Searches” section right away.

Task 2.2: GitHub Repository Finder

Lab03_2 > task2_2.html > ...

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>GitHub Repo Finder</title>
    <style>
      * {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
      }
      body {
        font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif;
        background: #0d1117;
        color: #c9d1d9;
        padding: 20px;
      }
      .container {
        max-width: 1000px;
        margin: 0 auto;
      }
      .header {
        text-align: center;
        margin-bottom: 40px;
      }
      .header h1 {
        color: #58a6ff;
        margin-bottom: 10px;
      }
      .search-container {
        background: #161b22;
        padding: 20px;
        border-radius: 8px;
        margin-bottom: 30px;
      }
      .search-box {
        display: flex;
        gap: 10px;
        margin-bottom: 15px;
      }
      .search-box input {
        flex: 1;
        padding: 12px;
```

```
    background: #0d1117;
    border: 1px solid #30363d;
    border-radius: 6px;
    color: #c9d1d9;
    font-size: 16px;
  }
  .search-box button {
    padding: 12px 30px;
    background: #238636;
    color: white;
    border: none;
    border-radius: 6px;
    cursor: pointer;
    font-size: 16px;
  }
  .search-box button:hover {
    background: #2ea043;
  }
  .filters {
    display: flex;
    gap: 10px;
    align-items: center;
  }
  .filters label {
    color: #8b949e;
  }
  .filters select {
    padding: 8px 12px;
    background: #0d1117;
    border: 1px solid #30363d;
    border-radius: 6px;
    color: #c9d1d9;
  }
  .repo-card {
    background: #161b22;
    border: 1px solid #30363d;
    border-radius: 8px;
    padding: 20px;
    margin-bottom: 15px;
```

```
    transition: transform 0.2s;
  }
  .repo-card:hover {
    transform: translateY(-2px);
    border-color: #58a6ff;
  }
  .repo-name {
    color: #58a6ff;
    font-size: 20px;
    font-weight: bold;
    text-decoration: none;
  }
  .repo-name:hover {
    text-decoration: underline;
  }
  .repo-description {
    color: #8b949e;
    margin: 10px 0;
  }
  .repo-meta {
    display: flex;
    gap: 20px;
    font-size: 14px;
    color: #8b949e;
    flex-wrap: wrap;
  }
  .language-badge {
    display: inline-block;
    padding: 4px 10px;
    background: #1f6feb;
    border-radius: 12px;
    font-size: 12px;
    color: white;
  }
  .loading {
    text-align: center;
    padding: 40px;
    color: #8b949e;
  }
```

```
122       .error {
123         background: #da3633;
124         color: white;
125         padding: 15px;
126         border-radius: 6px;
127         margin-bottom: 20px;
128       }
129       .load-more {
130         text-align: center;
131         margin: 30px 0;
132       }
133       .load-more button {
134         padding: 12px 40px;
135         background: #21262d;
136         border: 1px solid #30363d;
137         color: #c9d1d9;
138         border-radius: 6px;
139         cursor: pointer;
140         font-size: 16px;
141       }
142       .load-more button:hover {
143         background: #30363d;
144       }
145     </style>
146   </head>
147   <body>
148     <div class="container">
149       <div class="header">
150         <h1>🔍 GitHub Repository Finder</h1>
151         <p style="color: #8b949e">Search and explore GitHub repositories</p>
152       </div>
153
154       <div class="search-container">
155         <div class="search-box">
156           <input
157             id="searchInput"
158             placeholder="Search repositories..."
159             onkeypress="if(event.key==='Enter')performSearch()"
160           />
161           <button onclick="performSearch()">Search</button>
```

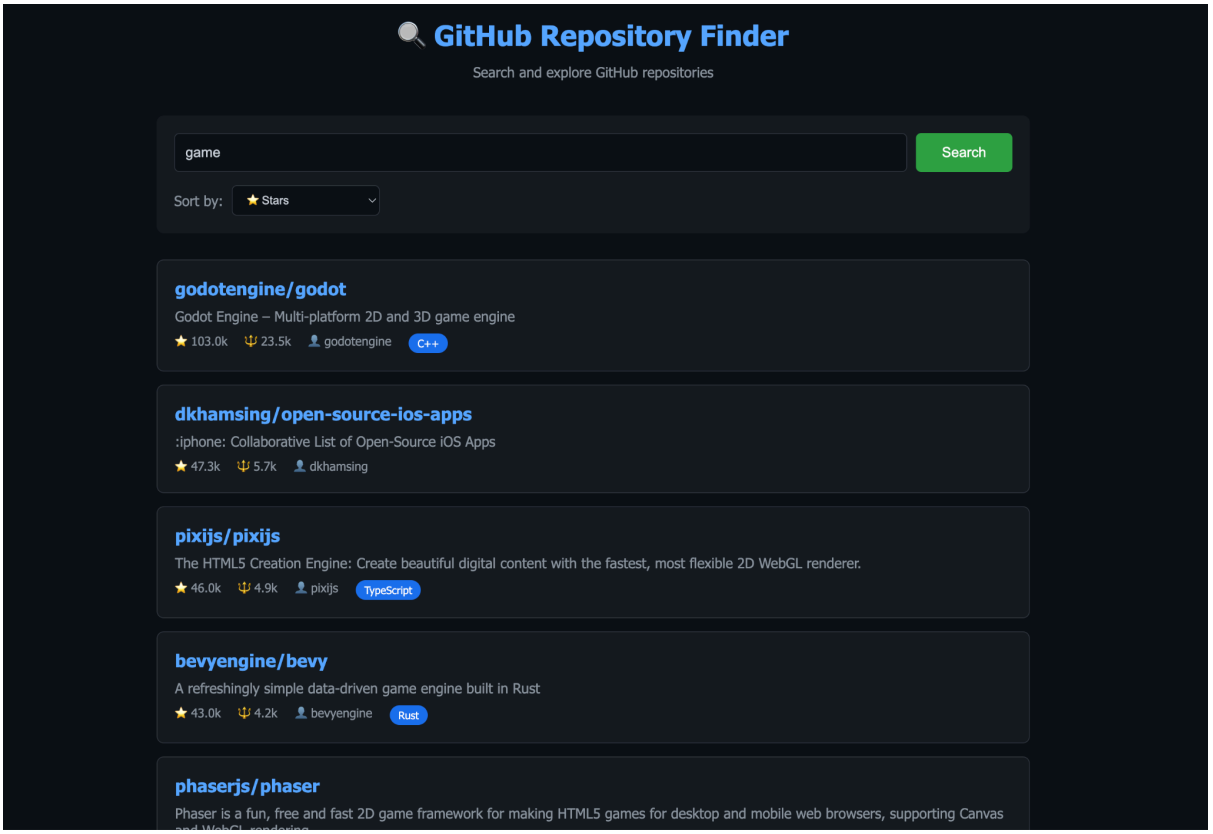
```
      </div>
      <div class="filters">
        <label>Sort by:</label>
        <select id="sortSelect" onchange="performSearch()">
          <option value="stars">⭐ Stars</option>
          <option value="forks">🔱 Forks</option>
          <option value="updated">🕒 Recently Updated</option>
        </select>
      </div>
    </div>

    <div id="errorMessage"></div>
    <div id="repoList"></div>
    <div id="loadMoreContainer"></div>
  </div>

  <script>
    let currentPage = 1;
    let currentQuery = "";
    let currentSort = "stars";
    let totalResults = 0;

    async function searchRepositories(query, sort = "stars", page = 1) {
      const url = `https://api.github.com/search/repositories?q=${encodeURIComponent(
        query
      )}&sort=${sort}&page=${page}&per_page=10`;
      const res = await fetch(url);
      if (res.status === 403)
        throw new Error("Rate limit exceeded (max 60/hr)");
      if (!res.ok) throw new Error("Failed to fetch repositories");
      return res.json();
    }

    function createRepoCard(repo) {
      return `
    <div class="repo-card">
      <a href="${repo.html_url}" class="repo-name" target="_blank">${
        repo.full_name
```

```
      }</a>
      <p class="repo-description">${
        repo.description || "No description available."
      }</p>
      <div class="repo-meta">
        <span>⭐ ${formatNumber(repo.stargazers_count)}</span>
        <span>🔱 ${formatNumber(repo.forks_count)}</span>
        <span>👤 ${repo.owner.login}</span>
        ${
          repo.language
            ? `<span class="language-badge">${repo.language}</span>`
            : ""
        }
      </div>
    </div>`;
    }

    function displayRepositories(repos, append = false) {
      const container = document.getElementById("repoList");
      const html = repos.map(createRepoCard).join("");
      container.innerHTML = append ? container.innerHTML + html : html;
    }

    async function performSearch() {
      const query = document.getElementById("searchInput").value.trim();
      const sort = document.getElementById("sortSelect").value;
      if (!query) return;
      clearError();
      currentPage = 1;
      currentQuery = query;
      currentSort = sort;
      const repoList = document.getElementById("repoList");
      repoList.innerHTML = '<div class="loading">Loading...</div>';
      document.getElementById("loadMoreContainer").innerHTML = "";
      try {
        const data = await searchRepositories(query, sort, currentPage);
        totalResults = data.total_count;
        displayRepositories(data.items);
        if (totalResults > 10) showLoadMore();
```

```
    } catch (err) {
      showError(err.message);
    }
  }

  async function loadMore() {
    currentPage++;
    try {
      const data = await searchRepositories(
        currentQuery,
        currentSort,
        currentPage
      );
      displayRepositories(data.items, true);
      if (currentPage * 10 >= totalResults)
        document.getElementById("loadMoreContainer").innerHTML = "";
    } catch (err) {
      showError(err.message);
    }
  }

  function showLoadMore() {
    document.getElementById(
      "loadMoreContainer"
    ).innerHTML = `<div class="load-more"><button onclick="loadMore()">Load More Results</button></div>`;
  }

  function showError(msg) {
    document.getElementById(
      "errorMessage"
    ).innerHTML = `<div class="error">${msg}</div>`;
  }
  function clearError() {
    document.getElementById("errorMessage").innerHTML = "";
  }

  function formatNumber(num) {
    if (num >= 1_000_000) return (num / 1_000_000).toFixed(1) + "M";
    if (num >= 1000) return (num / 1000).toFixed(1) + "k";
```

```
      return num;
    }
  </script>
 </body>
</html>
```

Output:



Explain:

1. searchRepositories(query, sort = "stars", page = 1)

This asynchronous function sends a request to the GitHub Search API to find repositories matching the given query. It includes parameters for sorting (stars, forks, or updated) and pagination (page). If GitHub's rate limit (60 requests/hour) is exceeded, it throws an error. Otherwise, it returns the response data as JSON, which includes a list of repositories and total results.

2. createRepoCard(repo)

This function builds an HTML card for a single repository using its data (name, description, stars, forks, owner, and language).

It uses template literals to create a clickable layout that links directly to the repository's GitHub page. If the repo has no description or language, it displays default text or omits that field.

3. displayRepositories(repos, append = false)
This function displays a list of repositories on the page. It maps each repository to an HTML card using createRepoCard() and then inserts the result into the #repoList container. If append is true, new results are added below the existing ones (used for pagination); otherwise, it replaces the current content.

4. performSearch()
This is the main control function triggered when the user clicks "Search" or presses Enter. It reads the search keyword and selected sorting option, resets pagination, and shows a loading message. Then it calls searchRepositories() to fetch results, displays them using displayRepositories(), and adds a "Load More" button if there are more than 10 results. If an error occurs (e.g., API limit or network issue), it shows an error message using showError().

5. loadMore()
This function enables infinite scrolling/pagination. When the "Load More" button is clicked, it increments the currentPage, fetches the next set of repositories, and appends them to the existing list. If all results have been loaded, it removes the "Load More" button.

6. showLoadMore()
Displays the "Load More Results" button at the bottom of the list by dynamically inserting a button into #loadMoreContainer.
When clicked, it calls the loadMore() function.

7. showError(msg)
Displays an error alert box (red background) inside the #errorMessage section with a custom message passed in as msg.

8. clearError()

Clears any existing error message before starting a new search or updating results.

9. formatNumber(num)
Formats large numbers for readability:

- Converts millions to M (e.g., 2,300,000 → “2.3M”)
- Converts thousands to k (e.g., 4,500 → “4.5k”)
- Returns smaller numbers unchanged.

This function is used to format the star and fork counts displayed in each repo card.