📘**SDLC Phase 1: Requirement Analysis**

**🔍 Introduction**

Requirement analysis is the foundational stage of the Software Development Life Cycle (SDLC), where the project's goals, functions, and constraints are determined. This phase ensures that the application is built to meet user expectations and functional requirements. For the To-Do List Project, this phase helps outline what the app should do, who it is for, and how it should behave under different conditions.

**🧠 Purpose of Requirement Analysis**

The goal is to clearly understand:

* What users need from the To-Do List application
* How the system should respond to different actions
* Which features are critical and which are optional
* The constraints (platform, tools, time, skills)

**🎯 Objectives of the To-Do List Project**

The To-Do List app aims to:

* Help users manage daily tasks efficiently
* Provide a simple and visually appealing interface
* Allow users to add, edit, delete, and mark tasks as completed
* Display task completion statistics using progress indicators
* Offer a responsive and user-friendly experience in the browser

**👤 Target Users**

* Students managing homework and assignments
* Professionals tracking tasks and meetings
* Anyone needing a lightweight personal task manager

**💡 Functional Requirements**

These are specific features the system must support:

* **Add Tasks**: Users can input and add new tasks via a text field and a button.
* **Edit Tasks**: Users can edit task names using an inline pop-up.
* **Delete Tasks**: Tasks can be removed permanently with a delete icon.
* **Mark Tasks as Completed**: Checkbox toggles let users complete/uncomplete tasks.
* **Track Progress**: A progress bar shows the percentage of tasks completed.
* **Persistent Storage**: Tasks are saved in the browser's localStorage.
* **Interactive UI**: Includes animations and task highlighting.

**🔐 Non-Functional Requirements**

These refer to how the system behaves:

* **Usability**: Clean and intuitive interface with minimal learning curve.
* **Performance**: Fast response times for UI interactions.
* **Compatibility**: Works on modern web browsers (Chrome, Firefox, Edge).
* **Scalability**: Can handle dozens of tasks without performance drop.
* **Aesthetics**: Styled with modern CSS including a custom background, buttons, and task boxes.

**🧾 Use Case Scenarios**

1. **Adding a Task**  
   The user enters a task and clicks the add button. The task appears in the list.
2. **Editing a Task**  
   Clicking the edit icon opens a prompt where the user modifies the task text.
3. **Deleting a Task**  
   Clicking the delete icon removes the task from the list and local storage.
4. **Completing a Task**  
   Toggling the checkbox marks a task as done and updates the progress bar.
5. **Session Continuity**  
   Reloading the page shows the previous tasks using data from localStorage.

**🧱 System Requirements**

* **Frontend Technologies**: HTML, CSS, JavaScript
* **Backend**: Not required; local storage is used
* **Hosting**: Can be hosted on GitHub Pages or any static web host

**⚠️ Constraints and Assumptions**

* The app runs only in a browser (not mobile app or desktop).
* No server-side storage or login system.
* Performance is optimal for up to ~100 tasks.

**📌 Summary**

The requirement analysis phase lays the foundation for your To-Do List project. It ensures you clearly understand what the app is supposed to do, who it's for, and how it should behave. By defining both functional and non-functional requirements, you're setting yourself up for a successful and organized development process.

## 🧩 SDLC Phase 2: System Design

### 🧭 Introduction

The **System Design** phase translates the gathered requirements into a blueprint for development. This includes designing the architecture, user interface, and interactions between system components. It’s where we decide **how** the system will fulfill the needs defined during Requirement Analysis.

### 🧱 Architecture Overview

The To-Do List app is built using a **client-side architecture**, meaning everything runs inside the user’s browser.

#### 🔧 Tech Stack

* **HTML**: Structures the layout of the app
* **CSS**: Styles the elements to enhance UX
* **JavaScript**: Implements logic, dynamic features, and localStorage handling

#### 📦 Folder Structure

bash

CopyEdit

/project-root

│

├── index.html # Main HTML structure

├── main.js # Application logic

├── main.css/style.css # Styling

├── assets/ # Icons, background images

└── localStorage # (Browser feature, not a file)

### 🧑‍🎨 User Interface Design

#### Main Layout

* A centered task container with a background image
* Header with branding/icons
* Task input field and add button
* Task list section
* Progress bar
* Footer section with information links

#### UI Elements

| **Component** | **Description** |
| --- | --- |
| **Input Box** | Text field where users enter new tasks |
| **Add Button** | Adds the entered task to the list |
| **Task Item** | A card displaying a task with edit/delete icons |
| **Progress Bar** | Indicates how many tasks are completed |
| **Filters/Delete All** | Buttons to filter tasks or remove all |

### 🔗 Interaction Flow (High-Level)

text

CopyEdit

User enters task → Clicks Add → Task appears in list

→ Task stored in localStorage

→ User can edit/delete/mark as complete

→ UI updates & progress bar adjusts

### 🎨 UI Design Choices

* **Colors**: Blue tones for positivity and white for cleanliness
* **Buttons**: Rounded corners and hover effects for modern feel
* **Fonts**: ‘Roboto’ for readability and professional appearance
* **Responsive Design**: Flexbox used for adaptability on different screen sizes

### 🧮 Data Handling Design

* **Data Structure**: Each task is stored as an object like:

js

CopyEdit

{

id: 1,

name: "Complete project",

completed: false

}

* **Storage**: localStorage.setItem() and getItem() to persist data

### 📐 Design Principles Followed

* **Separation of Concerns**: HTML, CSS, and JS are modular
* **Scalability**: Can be extended to include categories, priorities, etc.
* **Maintainability**: Code is organized with clear comments and naming
* **Simplicity**: Minimalist design for ease of use

### 📌 Summary

The design phase ensures the system is both technically sound and user-friendly. It defines how the app looks, how users will interact with it, and how internal components function together. This phase sets the visual and functional tone for development.

**🧑‍💻 SDLC Phase 3: Implementation (Coding)**

**🛠 Introduction**

The **Implementation phase** (also called development or coding) is where the system design comes to life. Based on the blueprint prepared earlier, each module of the application is coded and integrated. For your To-Do List project, this involves writing HTML for structure, CSS for styling, and JavaScript for dynamic behavior and data handling.

**🧱 Project Structure Recap**

| **File** | **Purpose** |
| --- | --- |
| index.html | Defines the app’s layout and structure |
| main.css / style.css | Provides styling, layout, and responsiveness |
| main.js | Contains the core logic of the app (add/edit/delete/save tasks) |
| \*.png/.jpg | Used as icons and background images |

**🔤 HTML Implementation**

The index.html file:

* Contains the app’s title, input field, buttons, and task display area
* Organizes the UI using <div>, <input>, <button>, and <ul> tags
* Includes external links to main.css and main.js

Key Snippet:

html

CopyEdit

<input type="text" id="taskInput" placeholder="Add a new task">

<button onclick="addTask()">Add</button>

<ul id="taskList"></ul>

**🎨 CSS Implementation**

The CSS (from main.css and style.css) includes:

* Styling for buttons, inputs, and cards
* Hover animations, transitions, and colors
* A custom background image and icon styling
* Responsive layout using Flexbox and media queries

Highlights:

* .task-card { box-shadow, border-radius, padding }
* button:hover { background-color transition effect }
* Progress bar animations and smooth visual feedback

**📜 JavaScript Implementation**

The main.js handles the core functionality:

1. **Adding Tasks**
   * Reads user input
   * Creates a new task object
   * Updates DOM and localStorage
2. **Displaying Tasks**
   * Loads tasks from localStorage
   * Creates HTML for each task and appends it to the list
3. **Editing Tasks**
   * Prompts user to change the task name
   * Updates it in both the DOM and localStorage
4. **Deleting Tasks**
   * Removes the task from the DOM and from localStorage
5. **Task Completion**
   * Toggle complete/incomplete using checkboxes
   * Updates UI and triggers progress bar update
6. **Progress Tracker**
   * Calculates how many tasks are done
   * Updates a visual bar based on percentage complete

Example:

javascript

CopyEdit

function addTask() {

let taskText = document.getElementById("taskInput").value;

if (taskText) {

tasks.push({ name: taskText, completed: false });

localStorage.setItem("tasks", JSON.stringify(tasks));

displayTasks();

}

}

**🧪 Code Quality and Best Practices**

* **Code Reusability**: Functions are modular (e.g., addTask(), displayTasks())
* **Readability**: Proper naming conventions and indentation
* **Comments**: In-code comments explain logic for future maintenance
* **Error Handling**: Input checks to avoid empty tasks

**⚙️ Tools Used**

* **VS Code** for writing code
* **Browser Developer Tools** for testing and debugging
* **localStorage** for data persistence

**📌 Summary**

This phase transforms your ideas and designs into a fully functioning To-Do List application. By following modular coding practices, the codebase becomes easy to maintain, enhance, and debug. The combination of HTML, CSS, and JavaScript in this phase brings the entire user experience to life.

## 🧪 SDLC Phase 4: Testing

### 🧭 Introduction

The **Testing** phase ensures that the developed software meets all specified requirements and is free of bugs or usability issues. For your To-Do List project, testing was performed manually (since it's a front-end web app), focusing on functionality, user interface behavior, and data persistence.

### 🎯 Objectives of Testing

* Verify all features work correctly
* Ensure tasks persist after page reloads (using localStorage)
* Validate responsiveness and visual consistency
* Detect and fix any UI or logical bugs

### 🧪 Types of Testing Performed

#### ✅ 1. Functional Testing

Checks if each function works as expected:

| **Feature** | **Status** | **Test Case** |
| --- | --- | --- |
| Add a new task | ✅ Pass | Enter a task, click "Add", verify it appears |
| Edit an existing task | ✅ Pass | Click "Edit" icon, modify text, save and confirm update |
| Delete task | ✅ Pass | Click delete icon, verify task disappears |
| Mark as complete | ✅ Pass | Click checkbox, check if UI updates and progress changes |
| Save to localStorage | ✅ Pass | Refresh page, verify tasks remain |
| Clear All | ✅ Pass | Click “Clear All” and ensure all tasks are removed |

#### 📱 2. UI/UX Testing

Ensures the app looks and feels good:

* Input field and buttons align properly ✅
* Buttons respond to hover and click ✅
* Mobile responsiveness using Flexbox ✅
* Icons are visible and intuitive ✅

#### 🧪 3. Boundary & Edge Case Testing

| **Scenario** | **Expected Result** | **Status** |
| --- | --- | --- |
| Add an empty task | Show warning / ignore | ✅ |
| Delete when list is empty | No error / no action | ✅ |
| Add duplicate task names | Both appear (no validation) | ✅ |
| Extremely long task text | Scrolls or wraps | ✅ |
| Special characters in task | Renders properly | ✅ |

#### 💾 4. Data Persistence Testing

* **Check**: Add a task, reload browser, verify it still exists
* **Tested With**: Chrome, Firefox, Edge
* **Result**: Tasks persisted correctly across sessions ✅

### 🧹 Bugs Found and Fixed

| **Bug Found** | **Fix Applied** |
| --- | --- |
| Tasks not clearing from localStorage on "Clear All" | Fixed with .clear() method |
| Task text not updating after edit | Fixed by re-rendering task list after edit |
| Progress bar not accurate after delete | Fixed by re-triggering updateProgressBar() |

### 📋 Tools Used for Testing

* **Chrome DevTools** (Console, Storage tab)
* **Manual Browser Testing** (Chrome, Firefox)
* **Responsive Mode Testing** (Mobile views)

### 🔐 Limitations of Testing

* No automated unit tests (like Jest or Cypress)
* No server-side testing (since this is a static frontend project)
* Testing was manually conducted without a testing framework

### 📌 Summary

The testing phase confirmed that your To-Do List app is stable, functional, and provides a smooth user experience. Functional tests, UI checks, and storage validation ensured the app behaves correctly under different scenarios and environments.

## 🚀 SDLC Phase 5: Deployment

### 🧭 Introduction

The **Deployment phase** involves releasing the completed software to a live environment so that users can interact with it. For your To-Do List project, deployment can be done on platforms that support static websites, like **GitHub Pages**, **Netlify**, or **Vercel**.

### 🌐 Deployment Process

#### 1. **Preparing the Codebase**

* Verified all files were organized properly:
  + index.html
  + main.js
  + main.css or style.css
  + Assets folder containing images/icons
* Checked for broken links, missing files, or syntax errors

#### 2. **Choosing the Deployment Platform**

You can deploy on any of the following:

| **Platform** | **Features** |
| --- | --- |
| **GitHub Pages** | Easy for static sites, version control |
| **Netlify** | Drag-and-drop support, automatic CI/CD |
| **Vercel** | Great for frontend projects, fast global CDN |

#### 3. **Steps (GitHub Pages Example)**:

* Push project to a GitHub repository
* Go to repository → Settings → Pages
* Choose the root directory or main branch
* Wait for GitHub to build and publish your site
* Your live site will be available at: https://yourusername.github.io/todo-list/

### 🧾 Post-Deployment Checklist

| **Task** | **Status** |
| --- | --- |
| Code is minified | ✅ (Optional, but can improve load speed) |
| Assets load properly | ✅ |
| App opens without console errors | ✅ |
| Tested on multiple browsers | ✅ |
| Mobile responsiveness | ✅ |

### 🧪 Live Environment Testing

After deployment, tested the following:

* Task creation, deletion, and editing in live site
* Reloading the page to confirm localStorage still works
* Confirming all images/icons load correctly
* Responsive behavior across screen sizes

### ⚙️ Maintenance Considerations

* **Future Updates**: You can push new changes to GitHub, and the deployed version updates automatically (CI/CD)
* **Bug Fixes**: Use GitHub issues or a changelog to track and fix bugs
* **Version Control**: Git helps track history of changes for rolling back if needed

### 📌 Summary

Deployment brings your project to real users. With platforms like GitHub Pages or Netlify, the process is seamless for static sites like your To-Do List app. After deployment, it’s important to keep testing, monitor user behaviour, and make improvements where needed.

## 🔧 SDLC Phase 6: Maintenance

### 🧭 Introduction

The **Maintenance phase** begins after the application has been deployed and users start interacting with it. It focuses on ensuring that the application remains reliable, relevant, and bug-free over time. This phase is ongoing and essential for long-term success.

### 🛠 Types of Maintenance

#### 🐞 1. **Corrective Maintenance**

Fixing bugs that are discovered after deployment.

Example:

* A user reports that the "Edit" feature doesn't work on mobile.
* Investigation reveals a JavaScript event handling issue.
* Code is updated, tested, and redeployed.

#### 🧹 2. **Preventive Maintenance**

Optimizing the codebase or refactoring to improve performance or future scalability.

Example:

* Minifying JavaScript and CSS for faster loading
* Removing unused CSS classes or optimizing image sizes

#### ✨ 3. **Perfective Maintenance**

Adding new features or improving UI/UX based on user feedback.

Ideas:

* Adding task categories (e.g., Work, Personal)
* Introducing drag-and-drop task reordering
* Adding dark mode or theme toggles

#### 🔒 4. **Adaptive Maintenance**

Updating the app to stay compatible with browser updates or integrating with new technologies.

Example:

* Ensuring compatibility with future Chrome or Firefox versions
* Migrating storage from localStorage to IndexedDB for scalability

### 📌 Ongoing Tasks in Maintenance

| **Task** | **Description** |
| --- | --- |
| Monitor performance | Use browser dev tools to check load time, responsiveness |
| Collect feedback | Ask users what features or improvements they want |
| Fix bugs | Track and resolve new issues quickly |
| Update documentation | Keep README and instructions up-to-date |
| Push updates | Use version control to deploy patches and new features |

### 🧪 Tools for Maintenance

* **GitHub Issues** – Track bugs and improvements
* **Browser DevTools** – Diagnose layout or performance issues
* **Version Control (Git)** – Roll back or apply new changes easily
* **Responsive Design Checker** – Ensure compatibility on all devices

### 📊 Example Maintenance Log (Fictional)

| **Date** | **Task** | **Type** | **Status** |
| --- | --- | --- | --- |
| 2025-04-10 | Fixed edit icon misalignment | Corrective | Done |
| 2025-04-15 | Added "Priority" feature | Perfective | Planned |
| 2025-04-20 | Optimized CSS file | Preventive | Done |
| 2025-04-22 | Tested on iOS Safari | Adaptive | Done |

### 📌 Summary

The Maintenance phase ensures your To-Do List app continues to deliver value to users over time. Whether it’s fixing issues, enhancing functionality, or adapting to new environments, regular maintenance helps retain users and keeps the application future-ready.