TaskFlow

Smart Todo Manager

Architecture & Developer Documentation

Version 2.0 | February 2026

A complete guide to understanding, maintaining, and extending the TaskFlow application.

HTML5 | CSS3 | JavaScript ES6+ | Chart.js | LocalStorage API

Table of Contents

1. Project Overview

TaskFlow is a modern, full-featured task management web application built entirely with vanilla HTML, CSS, and JavaScript. It provides a professional-grade user experience with a dashboard layout, real-time analytics, data visualization, and a fully responsive design that works across all device sizes from desktop monitors to mobile phones.

The application follows a client-side architecture where all data is persisted in the browser's localStorage API, requiring zero backend infrastructure. This makes it instantly deployable as a static site on any hosting platform.

Key Features

- CRUD operations: Create, read, update (toggle), and delete tasks
- Priority system: Three levels (Low, Medium, High) with color-coded badges
- Circular SVG progress bar: Animated ring showing completion percentage
- Weekly activity bar chart: Tasks completed per day over the last 7 days (Chart.js)
- Task distribution doughnut chart: Visual split of completed vs pending tasks
- Real-time search: Instant text filtering as you type
- Multi-filter toolbar: Filter by status (All/Active/Completed) and priority
- Dark mode: Full theme toggle with localStorage persistence
- Responsive design: 4 breakpoints (desktop, tablet, mobile, small mobile)
- Data migration: Automatic upgrade from legacy localStorage format
- XSS protection: All user input is escaped before rendering
- Smooth animations: slideIn, slideOut, shake validation, hover micro-interactions
- Batch operations: Clear all completed tasks at once with animated removal

2. Technology Stack

TaskFlow is intentionally built with zero build tools and no npm dependencies. This makes it easy to clone, open, and run instantly in any browser.

Technology	Version	Purpose
HTML5	Living Standard	Semantic structure, SVG graphics, ARIA attrs
CSS3	Living Standard	Custom Properties, Grid, Flexbox, animations
JavaScript	ES6+	Application logic, DOM manipulation, events
Chart.js	v4.x (CDN)	Bar chart and doughnut chart rendering
Google Fonts	Inter	Typography (weights 300-800)
localStorage	Web API	Client-side data persistence
SVG	1.1	Circular progress bar, inline icons

> Note: Chart.js is loaded from CDN (cdn.jsdelivr.net/npm/chart.js). If it fails to load, the app degrades gracefully - all features work except the two charts.

3. Project Structure

Directory Tree

```
taskflow-project/
|-- index.html              # Single-page application entry point
|-- css/
|   +-- style.css           # Complete stylesheet (1077 lines)
|-- js/
|   +-- app.js              # Application logic (512 lines)
|-- assets/
|   +-- images/
|       +-- check.png       # Legacy asset (unused, kept for history)
|-- docs/
|   +-- TaskFlow_Architecture.pdf   # This document
|-- .gitignore
|-- README.md
```

The project follows a classic separation of concerns with three core files:

- index.html - Structure and content (349 lines)
- css/style.css - Presentation and responsive layout (1077 lines)
- js/app.js - Behavior, state management, and charts (512 lines)

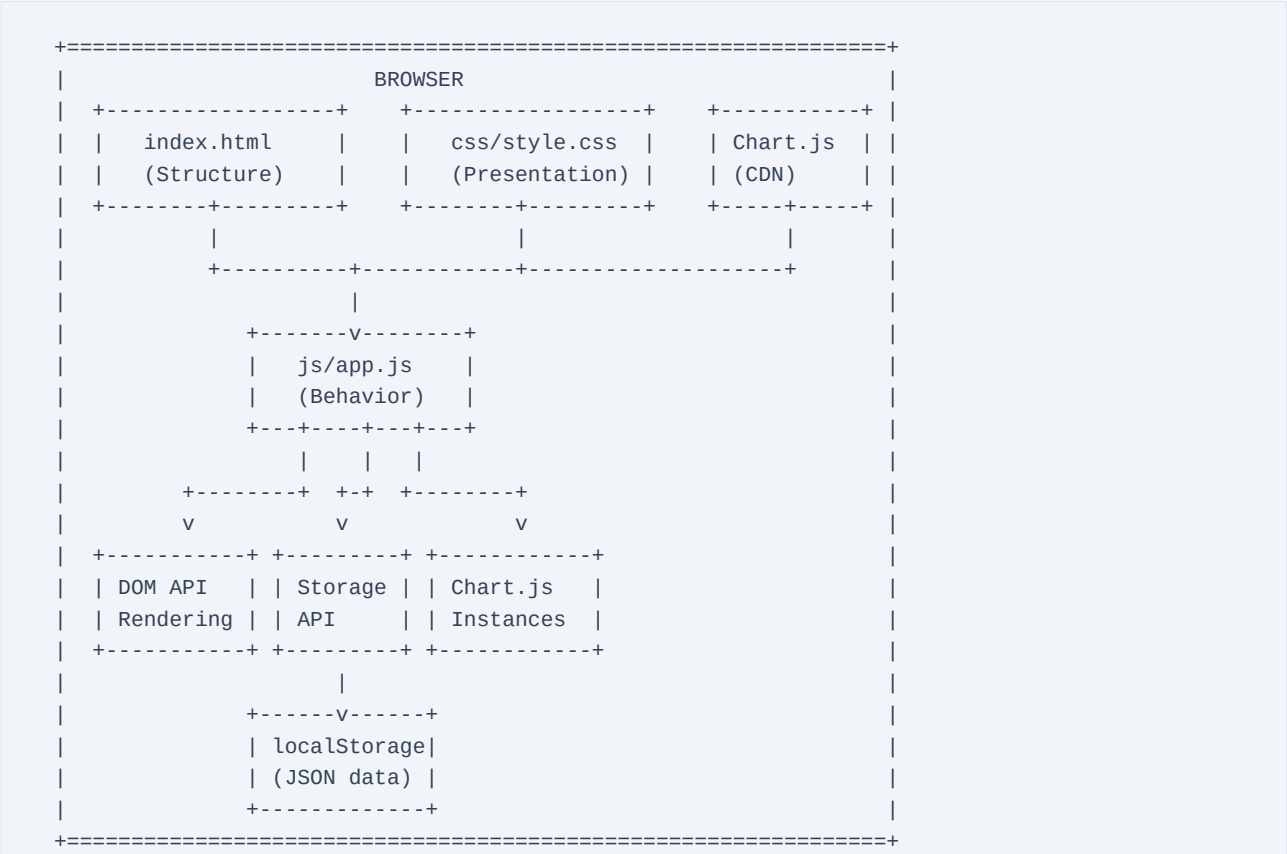
Total codebase size is approximately 1,940 lines of hand-written code. There is no build step, no bundler, and no transpilation. The code runs directly in the browser as-is.

4. Architecture Overview

TaskFlow uses a straightforward Model-View-Controller-like pattern implemented in vanilla JavaScript. The architecture is designed for simplicity and maintainability.

4.1 High-Level Architecture Diagram

System Architecture



4.2 Data Flow

Unidirectional Data Flow

```
User Action (click/type/submit)
        |
        v
Event Handler (handleAddTask, handleTaskClick, etc.)
        |
        v
State Mutation (todos array modified)
        |
        +----> saveTodos() ----> localStorage.setItem()
        |
        v
render() function called
        |
        +----> renderTasks()      (rebuilds task list DOM)
        +----> updateStats()      (updates circle + stat cards)
        +----> updateCharts()     (refreshes Chart.js instances)
```

Every user interaction follows the same unidirectional flow: Event -> State Mutation -> Save -> Re-render. This predictable pattern makes debugging straightforward: if the UI is wrong, check the state; if the state is wrong, check the event handler.

4.3 Module Organization

Although contained in a single file (app.js), the code is organized into clearly separated sections using comment headers:

Section	Responsibility
Constants	Storage keys, configuration values
State	Global todos array, chart instance references
DOM References	Cached querySelector results in 'el' object
Utilities	generateId, escapeHtml, capitalize, formatDate
Local Storage	loadTodos (with migration), saveTodos
Todo CRUD	addTodo, toggleTodo, deleteTodo, clearCompleted
Filtering	getFilteredTodos with search + status + priority
Rendering	createTaskElement, renderTasks, updateStats, render
Charts	Chart.js init, update, weekly data, color helpers
Theme	initTheme, toggleTheme with chart color sync
Event Handlers	handleAddTask, handleTaskClick
Initialization	init() - wires everything on DOMContentLoaded

5. HTML Layer (index.html)

The HTML file is a single-page application with semantic elements. It defines the complete UI structure and relies on CSS for layout and JS for dynamic behavior.

5.1 Document Head

The <head> section includes viewport meta for responsive design, Google Fonts preconnect for faster font loading, the Inter typeface (weights 300-800), and the main stylesheet.

5.2 Layout Structure

DOM Tree Overview

```
<body>
  <header class="app-header">         <!-- Sticky gradient header -->
    +-- .brand                         <!-- Logo + title + date -->
    +-- .theme-btn                     <!-- Dark mode toggle -->
  </header>

  <div class="container app-layout">   <!-- CSS Grid: sidebar + main -->
    <aside class="sidebar">            <!-- Dashboard panel -->
      +-- .circle-card                 <!-- SVG circular progress -->
      +-- .stats-row                   <!-- 3x stat cards grid -->
      +-- .chart-card (weekly)         <!-- Bar chart canvas -->
      +-- .chart-card (status)         <!-- Doughnut chart canvas -->
    </aside>

    <main class="main-content">        <!-- Task management area -->
      +-- .add-form                    <!-- Input + priority selector -->
      +-- .toolbar                     <!-- Search + filters + clear -->
      +-- .tasks-section               <!-- Task list + empty state -->
    </main>
  </div>

  <script src="chart.js (CDN)">
  <script src="js/app.js">
</body>
```

5.3 SVG Circular Progress Bar (HTML)

The circular progress bar is built with pure SVG. Two <circle> elements share the same center (cx=90, cy=90) and radius (r=75). The background circle uses a muted stroke, while the foreground circle uses an SVG linearGradient and the stroke-dasharray/stroke-dashoffset technique for animation.

SVG Markup

```
<svg class="circle-svg" viewBox="0 0 180 180">
  <defs>
    <linearGradient id="progress-gradient" x1="0%" y1="0%" x2="100%" y2="100%">
      <stop offset="0%" stop-color="#6366f1"/>    <!-- Indigo -->
      <stop offset="100%" stop-color="#06b6d4"/>   <!-- Cyan -->
    </linearGradient>
  </defs>
  <circle class="circle-bg" cx="90" cy="90" r="75"/>
  <circle class="circle-fg" cx="90" cy="90" r="75" id="circle-progress"/>
</svg>
```

5.4 Priority Selector Pattern

The priority selector uses hidden <input type='radio'> elements inside <label> wrappers. CSS uses the adjacent sibling selector (input:checked + .priority-tag) to style the active state. This is a pure CSS solution requiring zero JavaScript for visual feedback.

5.5 Accessibility

- All interactive elements have aria-label attributes
- SVG icons are decorative (no alt text needed, labels on parent buttons)
- Form inputs have proper labels and autocomplete attributes
- Semantic HTML: <header>, <main>, <aside>, <section>, <nav> elements
- The theme toggle button has aria-label='Toggle theme'

6. CSS Architecture (style.css)

The stylesheet is 1,077 lines organized into clearly commented sections. It uses CSS Custom Properties (variables) extensively for theming, and CSS Grid + Flexbox for all layout needs.

6.1 Design Token System (CSS Custom Properties)

All visual values are centralized in :root as CSS Custom Properties. This enables instant theme switching by overriding variables in [data-theme='dark'].

Design Tokens (Light Theme)

```
:root {
  /* Color Palette */
  --primary: #6366f1;        /* Indigo - brand color */
  --primary-dark: #4f46e5;   /* Hover state */
  --primary-light: #818cf8;  /* Focus rings */
  --primary-subtle: #eef2ff; /* Backgrounds */
  --accent: #06b6d4;         /* Cyan - progress gradient end */
  --success: #10b981;        /* Green - completed state */
  --warning: #f59e0b;        /* Amber - pending/medium priority */
  --danger: #ef4444;         /* Red - delete/high priority */

  /* Semantic Surface Colors */
  --bg: #f1f5f9;             /* Page background */
  --surface: #ffffff;        /* Card backgrounds */
  --text: #0f172a;           /* Primary text */
  --text-secondary: #475569; /* Secondary text */
  --text-muted: #94a3b8;     /* Muted/disabled text */
  --border: #e2e8f0;         /* Border color */

  /* Elevation (Shadow Scale) */
  --shadow-sm through --shadow-lg

  /* Spacing & Shape */
  --radius-sm: 8px;  --radius: 12px;  --radius-lg: 16px;

  /* Motion */
  --transition: 200ms cubic-bezier(0.4, 0, 0.2, 1);
  --transition-slow: 400ms cubic-bezier(0.4, 0, 0.2, 1);

  /* Layout Constants */
  --header-height: 72px;
  --sidebar-width: 340px;
}
```

6.2 Dark Theme Override

Dark mode overrides every surface, text, border, and shadow variable. The subtle color variants use rgba() with low opacity for a natural dark-mode feel.

Dark Theme Overrides

```
[data-theme="dark"] {
  --bg: #0f172a;                            /* Slate-900 */
  --surface: #1e293b;                       /* Slate-800 */
  --text: #f1f5f9;                          /* Slate-100 */
  --text-secondary: #94a3b8;                /* Slate-400 */
  --border: #334155;                        /* Slate-700 */
  --primary-subtle: rgba(99, 102, 241, 0.15); /* Translucent */
  --success-subtle: rgba(16, 185, 129, 0.15);
  /* ... shadows get heavier opacity ... */
}
```

6.3 Layout System

Desktop Layout (> 1024px)

Desktop Grid

```
.app-layout {
  display: grid;
  grid-template-columns: 340px 1fr;   /* Fixed sidebar + fluid main */
  gap: 28px;
  align-items: start;                  /* Sidebar sticks to top */
}

.sidebar {
  position: sticky;
  top: calc(var(--header-height) + 28px);  /* Below sticky header */
}
```

Tablet Layout (768px - 1024px)

Tablet Breakpoint

```
@media (max-width: 1024px) {
  .app-layout { grid-template-columns: 1fr; }   /* Single column */
  .sidebar {
    position: static;                            /* Not sticky */
    display: grid;
    grid-template-columns: 1fr 1fr;              /* Charts side by side */
  }
  .circle-card, .stats-row { grid-column: 1/-1; } /* Full width */
}
```

Mobile Layout (< 768px)

On mobile, everything stacks vertically. The add button becomes icon-only, the toolbar stacks, delete buttons are always visible (no hover on touch), and the circular progress bar shrinks to 150px.

6.4 Circular Progress Bar (CSS)

SVG Circle Technique

```
.circle-svg {
  width: 180px; height: 180px;
  transform: rotate(-90deg);          /* Start from 12 o'clock */
}

.circle-fg {
  fill: none;
  stroke: url(#progress-gradient);     /* SVG gradient reference */
  stroke-width: 8;
  stroke-linecap: round;               /* Rounded endpoints */
  stroke-dasharray: 471.24;            /* Circumference = 2 * PI * 75 */
  stroke-dashoffset: 471.24;           /* 100% hidden initially */
  transition: stroke-dashoffset 1s cubic-bezier(0.4, 0, 0.2, 1);
}
```

Math: circumference = 2 * PI * radius = 2 * 3.14159 * 75 = 471.24. To show N% progress: stroke-dashoffset = 471.24 * (1 - N/100). At 0%: offset = 471.24 (fully hidden). At 100%: offset = 0 (fully visible).

7. JavaScript Architecture (app.js)

The JavaScript layer is 512 lines of vanilla ES6+ code organized into 12 logical sections. It manages application state, DOM rendering, chart integration, theme switching, and all user interactions.

7.1 State Management

State & DOM Cache

```
// Global state
let todos = [];              // Array of todo objects (source of truth)
let weeklyChart = null;      // Chart.js bar chart instance
let statusChart = null;      // Chart.js doughnut chart instance

// DOM element cache (queried once at load time)
const el = {
  addForm: $("#add-form"),
  taskInput: $("#task-input"),
  tasksList: $("#tasks-list"),
  circleProgress: $("#circle-progress"),
  // ... 14 more cached references
};
```

The 'todos' array is the single source of truth. The UI is always derived from it. DOM references are cached in the 'el' object at load time to avoid repeated querySelector calls during renders.

7.2 Todo Data Model

Todo Object Schema

```
{
  id: "m2abc1234xyz",           // Unique ID (timestamp + random base36)
  text: "Buy groceries",        // Task description (user input, escaped)
  completed: false,             // Boolean completion state
  priority: "medium",           // "low" | "medium" | "high"
  createdAt: "2026-02-08T...",  // ISO 8601 creation timestamp
  completedAt: null             // ISO 8601 completion timestamp (or null)
}
```

The completedAt timestamp is critical for the weekly activity chart. When a task is toggled complete, the current timestamp is stored. When uncompleted, it's set back to null. This enables accurate historical tracking of completion activity.

7.3 CRUD Operations

addTodo(text, priority)

Creates a new todo object with a generated ID and current timestamps, prepends it to the todos array (unshift for newest-first ordering), saves to localStorage, and triggers a full re-render.

toggleTodo(id)

Finds the todo by ID, flips the completed boolean, sets or clears the completedAt timestamp, saves, and re-renders. The completedAt timestamp feeds the weekly chart.

deleteTodo(id)

Filters the todo out of the array and saves. The re-render is handled by the caller after the slide-out animation completes (via animationend event).

clearCompleted()

Adds the 'slide-out' CSS class to all completed task DOM elements, waits 350ms for the animation, then bulk-removes completed todos from state and re-renders.

7.4 Rendering Pipeline

Render Pipeline

```
function render() {
  renderTasks();     // 1. Rebuild task list from filtered state
  updateStats();     // 2. Update stat cards + circle progress
  updateCharts();    // 3. Push new data to Chart.js instances
}

function renderTasks() {
  const filtered = getFilteredTodos();  // Apply search + filters
  el.tasksList.innerHTML = "";          // Clear current DOM
  filtered.forEach(todo => {
    el.tasksList.appendChild(createTaskElement(todo));
  });
  el.emptyState.classList.toggle("visible", filtered.length === 0);
}

function updateStats() {
  // Calculate totals
  const total = todos.length;
  const done = todos.filter(t => t.completed).length;
  const percent = total ? Math.round((done / total) * 100) : 0;

  // Update DOM text
  el.statTotal.textContent = total;
  el.statDone.textContent = done;
  el.statPending.textContent = total - done;

  // Update circular progress bar
  const circumference = 2 * Math.PI * 75;
  const offset = circumference * (1 - percent / 100);
  el.circleProgress.style.strokeDashoffset = offset;
  el.circlePercent.textContent = percent + "%";
}
```

7.5 Filtering System

Multi-criteria Filtering

```
function getFilteredTodos() {
  const search = el.searchInput.value.toLowerCase().trim();
  const status = el.filterStatus.value;     // "all"|"active"|"completed"
  const priority = el.filterPriority.value; // "all"|"low"|"medium"|"high"

  return todos.filter(todo => {
    const matchSearch = !search || todo.text.toLowerCase().includes(search);
    const matchStatus = status === "all" ||
      (status === "active" && !todo.completed) ||
      (status === "completed" && todo.completed);
    const matchPriority = priority === "all" || todo.priority === priority;
    return matchSearch && matchStatus && matchPriority;
  });
}
```

Filters compose cleanly: each criterion returns true if 'all' is selected, or checks the specific match. The search input triggers renderTasks on every keystroke (input event) for real-time feedback. The dropdown filters trigger on the change event.

7.6 Event Delegation

Instead of attaching event listeners to every task button, a single listener on the task list container uses event delegation with Element.closest() to determine what was clicked. This is more performant and automatically handles dynamically added tasks.

Event Delegation Pattern

```
function handleTaskClick(e) {
  const deleteBtn = e.target.closest(".task-delete");
  const checkBtn = e.target.closest(".task-check");
  const taskItem = e.target.closest(".task-item");

  if (!taskItem) return;

  if (deleteBtn) {
    taskItem.classList.add("slide-out");       // Trigger CSS animation
    taskItem.addEventListener("animationend", () => {
      deleteTodo(taskItem.dataset.id);         // Remove from state
      render();                                 // Re-render
    });
  } else if (checkBtn) {
    toggleTodo(taskItem.dataset.id);           // Toggle + re-render
  }
}

// Single listener for all tasks
el.tasksList.addEventListener("click", handleTaskClick);
```

8. Data Model & Storage

8.1 localStorage Schema

localStorage Keys

```
Key: "taskflow_todos"
Value: JSON array of todo objects

Example:
[
  {
    "id": "m2abc1234xyz",
    "text": "Finish quarterly report",
    "completed": true,
    "priority": "high",
    "createdAt": "2026-02-07T09:30:00.000Z",
    "completedAt": "2026-02-08T14:22:00.000Z"
  },
  {
    "id": "m2def5678abc",
    "text": "Buy milk",
    "completed": false,
    "priority": "low",
    "createdAt": "2026-02-08T08:00:00.000Z",
    "completedAt": null
  }
]

Key: "taskflow_theme"
Value: "light" or "dark"
```

8.2 Legacy Data Migration

The previous version of the app stored todos as a simple array of strings under the key 'todos'. The loadTodos() function automatically detects and migrates this format:

Migration Logic

```
function loadTodos() {
  // 1. Try new format first
  const data = localStorage.getItem(STORAGE_KEY);
  if (data) return JSON.parse(data);

  // 2. Check for legacy format
  const legacy = localStorage.getItem("todos");
  if (legacy) {
    const items = JSON.parse(legacy);
    const migrated = items.map(text => ({
      id: generateId(),
      text: typeof text === "string" ? text : String(text),
      completed: false,
      priority: "medium",
      createdAt: new Date().toISOString(),
      completedAt: null,
    }));
    saveTodos(migrated);               // Save in new format
    localStorage.removeItem("todos");  // Clean up legacy key
    return migrated;
  }

  return [];  // Fresh start
}
```

> The migration is transparent to the user. On first load after upgrade, legacy tasks appear with 'Medium' priority and today's date. The old 'todos' key is removed.

8.3 ID Generation

Unique ID Strategy

```
function generateId() {
  return Date.now().toString(36) + Math.random().toString(36).substr(2, 9);
}
// Example output: "m2k7f3a1x" + "abc123def" = "m2k7f3a1xabc123def"
```

IDs combine a base-36 timestamp with random characters, ensuring uniqueness even if multiple tasks are created in the same millisecond.

9. Circular Progress Bar

The circular progress bar is one of the most visually prominent features. It's built entirely with SVG and CSS - no canvas, no library.

9.1 How It Works

The technique uses three SVG/CSS properties working together:

- stroke-dasharray: Sets the total length of the dash pattern (= circumference)
- stroke-dashoffset: Controls how much of the stroke is hidden
- transform: rotate(-90deg): Rotates the circle so 0% starts at 12 o'clock

The math is simple:

Offset Calculation

```
Circumference = 2 * PI * radius = 2 * 3.14159 * 75 = 471.24

For a given completion percentage:
  offset = circumference * (1 - percent / 100)

Examples:
  0% complete   -> offset = 471.24 * (1 - 0)    = 471.24 (circle hidden)
  25% complete  -> offset = 471.24 * (1 - 0.25)  = 353.43
  50% complete  -> offset = 471.24 * (1 - 0.50)  = 235.62
  75% complete  -> offset = 471.24 * (1 - 0.75)  = 117.81
  100% complete -> offset = 471.24 * (1 - 1.0)   = 0      (full circle)
```

9.2 Gradient Effect

The stroke uses an SVG <linearGradient> that transitions from Indigo (#6366f1) to Cyan (#06b6d4). This gradient is defined in the <defs> section of the SVG and referenced via stroke: url(#progress-gradient) in CSS.

9.3 Animation

The CSS transition property on stroke-dashoffset provides smooth animation whenever the value changes. The 1-second cubic-bezier easing creates a satisfying deceleration effect as the progress ring fills or empties.

10. Chart.js Integration

10.1 Weekly Activity Bar Chart

The bar chart shows the number of tasks completed per day over the last 7 days. Data is computed from the completedAt timestamps in the todos array.

Weekly Data Computation

```
function getWeeklyData() {
  const labels = [], data = [];
  for (let i = 6; i >= 0; i--) {
    const date = new Date();
    date.setHours(0, 0, 0, 0);
    date.setDate(date.getDate() - i);

    labels.push(date.toLocaleDateString("en-US", { weekday: "short" }));

    const nextDay = new Date(date);
    nextDay.setDate(nextDay.getDate() + 1);

    const count = todos.filter(t => {
      if (!t.completedAt) return false;
      const d = new Date(t.completedAt);
      return d >= date && d < nextDay;    // Day boundary check
    }).length;
    data.push(count);
  }
  return { labels, data };
}
```

Chart configuration: borderRadius: 8 for rounded bar tops, maxBarThickness: 32px, no legend, custom tooltip with task count. Y-axis uses stepSize: 1 for integer ticks.

10.2 Task Distribution Doughnut Chart

The doughnut chart shows the split between completed and pending tasks. Colors: green (#10b981) for completed, amber (#f59e0b) for pending. The cutout is 72% creating a thin ring. When there are no tasks, a single gray segment displays with a 'No tasks' label.

10.3 Theme-Aware Chart Updates

When the user toggles dark mode, chart colors must update. The toggleTheme() function recalculates grid colors, text colors, and legend colors, then calls chart.update() to repaint.

Dynamic Chart Colors

```
function getChartColors() {
  const isDark = document.documentElement.dataset.theme === "dark";
  return {
    gridColor: isDark ? "rgba(148,163,184,0.08)" : "rgba(0,0,0,0.06)",
    textColor: isDark ? "#94a3b8" : "#64748b",
  };
}
```

10.4 Graceful Degradation

Chart initialization is wrapped in a try/catch block and checks for typeof Chart === 'undefined'. If Chart.js fails to load from CDN, the rest of the app continues to work normally. Only the two chart canvases remain empty.

11. Theme System (Dark Mode)

The theme system uses HTML data attributes and CSS Custom Property overrides for a zero-flicker, JavaScript-controlled dark mode.

11.1 How It Works

Theme Toggle Flow

```
1. <html data-theme="light">      (default)

2. CSS variables defined in :root   (light values)
   CSS overrides in [data-theme="dark"]  (dark values)

3. User clicks theme toggle button

4. JS toggles: document.documentElement.dataset.theme = "dark"

5. ALL CSS variables instantly update  (no class toggling per element)

6. CSS transition on body smooths the color change:
   transition: background 400ms, color 400ms

7. Preference saved: localStorage.setItem("taskflow_theme", "dark")

8. On next page load, initTheme() reads and applies saved preference
```

11.2 Icon Toggle

The theme button contains both a sun and moon SVG icon. CSS controls visibility:

Icon Toggle CSS

```
[data-theme="light"] .icon-sun  { display: none; }
[data-theme="light"] .icon-moon { display: block; }
[data-theme="dark"]  .icon-sun  { display: block; }
[data-theme="dark"]  .icon-moon { display: none; }
```

12. Responsive Design System

The application supports four layout tiers with media query breakpoints. The approach is desktop-first: the default styles target wide screens, and max-width media queries progressively adapt for smaller devices.

Breakpoint	Target	Layout	Key Changes
> 1024px	Desktop	Sidebar + Main grid	Sticky sidebar, full features
768-1024	Tablet	Single column	Sidebar becomes 2-col grid
481-768	Mobile	Single column	Icon-only btn, stacked toolbar
< 480px	Sm. Mobile	Single column	Compact spacing, 130px circle

12.1 Desktop (> 1024px)

- Two-column CSS Grid: 340px sidebar + fluid main content
- Sidebar is position: sticky, scrolls with content but stays visible
- Circular progress bar at 180x180px
- Full 'Add Task' button with text label
- Delete buttons hidden until hover

12.2 Tablet (768px - 1024px)

- Grid collapses to single column
- Sidebar becomes a 2-column grid (charts side by side)
- Circle card and stats row span full width
- Sidebar is no longer sticky

12.3 Mobile (< 768px)

- Header height reduces from 72px to 64px
- Container padding reduces from 24px to 16px
- Sidebar becomes fully stacked (single column)
- Circular progress bar shrinks to 150x150px
- Add button becomes icon-only (text hidden)
- Toolbar stacks vertically, filters stretch full width
- Delete buttons always visible (no hover on touch devices)

12.4 Small Mobile (< 480px)

- Container padding further reduces to 12px

- Brand text shrinks to 1.1rem
- Circular progress bar shrinks to 130x130px, percentage to 1.6rem
- Stat cards get tighter padding
- Chart heights reduce for better viewport usage

13. Animation System

TaskFlow uses CSS keyframe animations and transitions for all motion. No JavaScript animation libraries are used.

Animation	Trigger	Description
slideIn	Task added	Fades in + slides down from -10px (0.35s)
slideOut	Task deleted	Fades out + slides right 50px, collapses (0.35s)
shake	Empty input	Horizontal shake -6/+6px (0.4s)
Stroke offset	Stats change	Circle ring fills/empties smoothly (1s)
Hover lift	Mouse enter	translateY(-1px) + shadow on task items
Check pop	Toggle done	Checkbox scale 0.3 -> 1.0 with opacity
Theme fade	Mode toggle	Background/color transitions over 400ms
Button press	Click	scale(0.97) on active state

All animations use cubic-bezier(0.4, 0, 0.2, 1) easing, which is the Material Design 'standard' curve. This provides a natural deceleration feel.

14. Security Considerations

- XSS Prevention: All user-entered task text is escaped via escapeHtml() before being inserted into innerHTML. This function creates a temporary DOM text node to safely convert special characters to HTML entities.
- No eval() or Function(): The codebase never uses eval or dynamic code execution.
- No external data ingestion: The app only reads from localStorage (same-origin). There are no API calls, fetch requests, or external data sources.
- CDN integrity: Chart.js is loaded from jsdelivr CDN. For production, consider adding Subresource Integrity (SRI) hash attributes to the script tag.
- localStorage limits: ~5MB per origin. For a todo app, this is more than sufficient (thousands of tasks). No sensitive data is stored.

15. Performance Notes

- DOM caching: All querySelector calls happen once at init time and are stored in the 'el' object. Subsequent accesses are O(1) property lookups.
- Event delegation: A single click listener on the task list handles all task interactions, regardless of how many tasks exist.
- Chart.js update('none'): Charts are updated with animation disabled during normal data changes to prevent janky transitions on every keystroke.
- Font preconnect: Google Fonts uses <link rel='preconnect'> to establish early connections, reducing font load latency.
- Minimal reflows: The render() function clears and rebuilds the task list in a single innerHTML = '' + appendChild loop, minimizing layout thrashing.
- CSS containment: Cards use overflow: hidden, which allows the browser to optimize paint operations.
- No framework overhead: Zero library code for DOM management. The entire JS payload is ~512 lines (~12KB unminified).

16. Developer Guide

16.1 Getting Started

Quick Start

```
# Clone the repository
git clone <repo-url>
cd taskflow-project

# Open in browser (no build step needed)
# Option A: Double-click index.html
# Option B: Use a local server
npx serve .
# or
python -m http.server 8000
```

16.2 Adding a New Feature

Follow these steps to add a new feature (example: due dates):

- 1. Extend the todo schema in addTodo() with a new field (e.g., dueDate: null)
- 2. Add UI elements to the add-form in index.html (e.g., date picker input)
- 3. Update createTaskElement() to render the new field in the task item

> 4. Add CSS styles in style.css for the new UI elements
> 5. Update getFilteredTodos() if the new field should be filterable
> 6. Test the legacy migration - old todos without the field should work

16.3 Adding a New Chart

To add a new chart (example: priority distribution):

> 1. Add a <canvas> element inside a .chart-card in the sidebar HTML
> 2. Add a DOM reference in the 'el' object
> 3. Create a data computation function (e.g., getPriorityData())
> 4. Initialize the chart in initCharts()
> 5. Update it in updateCharts()
> 6. Handle theme colors in toggleTheme()

16.4 Modifying the Color Scheme

All colors are defined as CSS Custom Properties in :root (light) and [data-theme='dark'] (dark). To change the brand color from indigo to blue:

Color Scheme Change

```
/* Change in :root */
--primary: #3b82f6;       /* Blue-500 instead of Indigo-500 */
--primary-dark: #2563eb;  /* Blue-600 */
--primary-light: #60a5fa; /* Blue-400 */
--primary-subtle: #eff6ff; /* Blue-50 */

/* Also update the SVG gradient in index.html */
<stop offset="0%" stop-color="#3b82f6"/>
```

16.5 Key Files Quick Reference

What to change	Where to look
App name/branding	index.html line 36, TaskFlow
Color palette	css/style.css :root variables (lines 10-55)
Dark mode colors	css/style.css [data-theme='dark'] (lines 58-83)
Layout breakpoints	css/style.css @media queries (lines 874+)
Todo data schema	js/app.js addTodo() function (line 93)
Chart config	js/app.js initCharts() function (line 253)
Storage key	js/app.js STORAGE_KEY constant (line 2)
Progress bar math	js/app.js updateStats() (line 204)
Priority options	index.html priority-group (line 238)

TaskFlow

Architecture Documentation v2.0

Generated on February 08, 2026

Built with HTML5, CSS3, JavaScript ES6+, and Chart.js

Zero dependencies. Zero build tools. Pure web standards.