```
<!DOCTYPE html>
<html lang="en" class="dark">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Momentum Dashboard</title>
    <script src="https://cdn.tailwindcss.com"></script>
    <style>
        .completed {
            text-decoration: line-through;
            opacity: 0.5;
            transition: opacity 0.3s ease-in-out;
        }
        html.dark {
            background-color: #1d1d1d;
            color: #f0f0f0;
        }
        .task-card {
            transition: all 0.3s ease-in-out;
        }
        .task-card:hover {
            transform: translateY(-2px);
            box-shadow: 0 4px 8px rgba(0,0,0,0.3);
        }
    </style>
    <!-- Chosen Palette: Dark Neutral with Red (Urgent), Blue (Focus), and Grey (Subtle) -->
    <!-- Application Structure Plan: Adopts the user-approved three-column layout which effectively organizes tasks by urgency (Overdue, Today, Upcoming). This structure prioritizes user attention on critical items first, addressing the core problem of "dashboard blindness." The main interaction is clicking tasks to "complete" them, which provides immediate visual feedback and reinforces a sense of progress. The app is responsive, stacking columns on mobile for readability. -->
    <!-- Visualization & Content Choices: The source mockup is a task list, so the primary presentation method is structured HTML lists. Urgency is conveyed through color-coded column headers (Red, Blue, Grey) and Unicode icons (🔥, 🎯, 🗓️). The most critical task (oldest overdue item) is dynamically highlighted in a "spotlight" element to guide user action. All interactions are handled with vanilla JavaScript, and no SVG/Mermaid or charting libraries are needed or used. -->
    <!-- CONFIRMATION: NO SVG graphics used. NO Mermaid JS used. -->
</head>
<body class="bg-[#1d1d1d] text-[#f0f0f0] font-sans p-4 sm:p-6 min-h-screen flex flex-col">

    <!-- HEADER -->
```

```
    <header id="dashboard-header" class="px-2 pb-4 mb-6 flex justify-between items-baseline
border-b border-gray-700">
        <!-- Greeting and Date will be populated by JS -->
    </header>

    <!-- MAIN CONTENT GRID (3 COLUMNS) -->
    <main class="flex-grow grid grid-cols-1 md:grid-cols-3 gap-6">

        <!-- COLUMN 1: OVERDUE -->
        <section id="overdue-column" class="bg-[#2a2a2a] rounded-lg flex flex-col overflow-hidden
border-t-4 border-red-500 shadow-lg">
            <h2 class="flex items-center text-lg font-bold uppercase tracking-wider text-white p-4
bg-gradient-to-b from-gray-800 to-[#2a2a2a]">
                <span class="text-xl mr-3">🔥</span>Overdue
            </h2>
            <div id="overdue-list" class="overflow-y-auto p-4 space-y-4">
                <!-- Overdue tasks will be populated by JS -->
            </div>
        </section>

        <!-- COLUMN 2: TODAY'S FOCUS -->
        <section id="today-column" class="bg-[#2a2a2a] rounded-lg flex flex-col overflow-hidden
border-t-4 border-blue-500 shadow-lg">
            <h2 class="flex items-center text-lg font-bold uppercase tracking-wider text-white p-4
bg-gradient-to-b from-gray-800 to-[#2a2a2a]">
                <span class="text-xl mr-3">🎯</span>Today's Focus
            </h2>
            <div id="today-list" class="overflow-y-auto p-4 space-y-4">
                <!-- Today's tasks will be populated by JS -->
            </div>
        </section>

        <!-- COLUMN 3: UPCOMING -->
        <section id="upcoming-column" class="bg-[#2a2a2a] rounded-lg flex flex-col
overflow-hidden border-t-4 border-gray-600 shadow-lg">
            <h2 class="flex items-center text-lg font-bold uppercase tracking-wider text-white p-4
bg-gradient-to-b from-gray-800 to-[#2a2a2a]">
                <span class="text-xl mr-3">🗓️</span>Upcoming
            </h2>
            <div id="upcoming-list" class="overflow-y-auto p-4 space-y-4">
                <!-- Upcoming tasks will be populated by JS -->
            </div>
        </section>
    </main>
```

```
    <!-- FOOTER -->
    <footer id="dashboard-footer" class="pt-6 mt-6 text-sm text-gray-500 flex justify-between
items-center flex-shrink-0">
        <div>Last synced: <span id="sync-time"></span></div>
        <div class="text-2xl cursor-pointer" title="Settings">⚙️</div>
    </footer>

    <script>
    document.addEventListener('DOMContentLoaded', () => {
        // --- CONFIGURATION ---
        const apiToken = 'cd5de7d99ffc0d2ae55d1912828d6819a41cdbe7';
        const REFRESH_INTERVAL = 5 * 60 * 1000; // 5 minutes

        // --- DOM ELEMENTS ---
        const headerEl = document.getElementById('dashboard-header');
        const overdueListEl = document.getElementById('overdue-list');
        const todayListEl = document.getElementById('today-list');
        const upcomingListEl = document.getElementById('upcoming-list');
        const syncTimeEl = document.getElementById('sync-time');

        // --- DATE & TIME HELPERS ---
        const formatDate = (date) => {
            return new Intl.DateTimeFormat('en-US', { weekday: 'long', month: 'long', day: 'numeric'
}).format(date);
        };

        const formatTime = (date) => {
             return new Intl.DateTimeFormat('en-US', { hour: 'numeric', minute: 'numeric', hour12:
true }).format(date);
        };

        const getDaysOverdue = (dueDate, today) => {
            const diffTime = today - dueDate;
            return Math.ceil(diffTime / (1000 * 60 * 60 * 24));
        };

        // --- RENDER FUNCTIONS ---

        function renderHeader() {
            const now = new Date();
            const hour = now.getHours();
            let greeting = 'Hello.';
            if (hour < 12) greeting = 'Good morning.';
```

```
        else if (hour < 18) greeting = 'Good afternoon.';
        else greeting = 'Good evening.';

        headerEl.innerHTML = `
            <h1 class="text-3xl font-bold text-white">${greeting}</h1>
            <div class="text-lg text-gray-400">${formatDate(now)}</div>
        `;
        syncTimeEl.textContent = formatTime(now);
    }

    function createTaskElement(task, type, today) {
        const taskEl = document.createElement('div');
        taskEl.className = 'bg-gray-700 p-4 rounded-lg shadow-md cursor-pointer task-card';
        taskEl.dataset.taskId = task.id;

        const content = `<div class="text-base font-semibold
text-gray-100">${task.content}</div>`;
        let subtext = '';

        const dueDate = new Date(task.due.date + 'T23:59:59');

        if (type === 'overdue') {
            const days = getDaysOverdue(dueDate, today);
            const color = days > 2 ? 'text-red-400' : 'text-amber-400';
            subtext = `<div class="text-sm font-bold ${color} mt-1">Overdue by ${days} day${days
> 1 ? 's' : ''}</div>`;
        } else if (type === 'upcoming') {
            const tomorrow = new Date(today);
            tomorrow.setDate(tomorrow.getDate() + 1);
            let dateLabel = formatDate(dueDate);
            if (dueDate.toDateString() === tomorrow.toDateString()) {
                dateLabel = 'Tomorrow';
            }
            subtext = `<div class="text-sm text-gray-400 mt-1">${dateLabel}</div>`;
        }

        taskEl.innerHTML = content + subtext;
        taskEl.addEventListener('click', () => handleTaskClick(task.id, taskEl));
        return taskEl;
    }

    function createSpotlightElement(task) {
         const spotlightEl = document.createElement('div');
```

```
            spotlightEl.className = 'bg-blue-900/50 border-l-4 border-cyan-400 p-4 rounded-lg
shadow-lg';
            spotlightEl.innerHTML = `
                <div class="text-sm font-bold uppercase tracking-wider text-cyan-400 mb-2">#1
Priority</div>
                <div class="text-base font-semibold text-gray-100">${task.content}</div>
            `;
            return spotlightEl;
        }

        function renderDashboard(allTasks) {
            const today = new Date();
            today.setHours(0, 0, 0, 0);

            // Clear existing lists
            overdueListEl.innerHTML = '';
            todayListEl.innerHTML = '';
            upcomingListEl.innerHTML = '';

            // Separate tasks into categories
            const overdue = allTasks.filter(t => t.due && new Date(t.due.date + 'T23:59:59') < today);
            const forToday = allTasks.filter(t => t.due && new Date(t.due.date).toDateString() ===
today.toDateString());
            const upcoming = allTasks.filter(t => t.due && new Date(t.due.date) > today);

            // Sort overdue tasks by oldest first
            overdue.sort((a, b) => new Date(a.due.date) - new Date(b.due.date));

            // Render Overdue Tasks
            overdue.forEach(task => overdueListEl.appendChild(createTaskElement(task, 'overdue',
today)));

            // Determine and Render #1 Priority (Spotlight)
            const spotlightTask = overdue.length > 0 ? overdue[0] : null;

            if (spotlightTask) {
                todayListEl.appendChild(createSpotlightElement(spotlightTask));
            }

            // Render Today's Tasks (excluding the spotlighted one)
            forToday.forEach(task => {
                if (!spotlightTask || task.id !== spotlightTask.id) {
                    todayListEl.appendChild(createTaskElement(task, 'today', today));
                }
```

```
        });

        // Render Upcoming Tasks
        upcoming.sort((a, b) => new Date(a.due.date) - new Date(b.due.date))
            .forEach(task => upcomingListEl.appendChild(createTaskElement(task, 'upcoming',
today)));
    }

    // --- API & DATA HANDLING ---

    async function fetchTasks() {
        try {
            const response = await fetch('https://api.todoist.com/rest/v2/tasks', {
                headers: { 'Authorization': `Bearer ${apiToken}` }
            });
            if (!response.ok) {
                throw new Error(`API Error: ${response.status} ${response.statusText}`);
            }
            return await response.json();
        } catch (error) {
            console.error("Failed to fetch tasks:", error);
            document.body.innerHTML = `<div class="h-screen w-screen flex items-center
justify-center text-red-400 text-lg">Error fetching data. Please check your API Token or network
connection.</div>`;
            return [];
        }
    }

    async function handleTaskClick(taskId, element) {
        element.classList.add('completed');
        try {
            const response = await fetch(`https://api.todoist.com/rest/v2/tasks/${taskId}/close`, {
                method: 'POST',
                headers: { 'Authorization': `Bearer ${apiToken}` }
            });
            if (!response.ok) {
                throw new Error('Failed to complete task via API.');
            }
            // Refresh dashboard after a short delay for visual effect
            setTimeout(fetchAndRenderDashboard, 400);
        } catch (error) {
            console.error("Failed to complete task:", error);
            element.classList.remove('completed'); // Revert on failure
            alert('Could not complete task. Please try again.');
```

```
        }
    }

    async function fetchAndRenderDashboard() {
        renderHeader(); // Update time
        const tasks = await fetchTasks();
        if(tasks) {
            renderDashboard(tasks);
        }
    }

    // --- INITIALIZE DASHBOARD ---
    function init() {
        fetchAndRenderDashboard();
        setInterval(fetchAndRenderDashboard, REFRESH_INTERVAL);
    }

    init();
});
</script>
</body>
</html>
```