

Task Scoring Function Documentation

February 8, 2026

1 Original Code

The following is the original calculateTaskScore function.

```
1  function calculateTaskScore(task) {
2    // Base priority weights
3    const priorityWeights = {
4      [TaskPriority.LOW]: 1,
5      [TaskPriority.MEDIUM]: 2,
6      [TaskPriority.HIGH]: 3,
7      [TaskPriority.URGENT]: 4
8    };
9
10   // Calculate base score from priority
11   let score = (priorityWeights[task.priority] || 0) * 10;
12
13   // Add due date factor (higher score for tasks due sooner)
14   if (task.dueDate) {
15     const now = new Date();
16     const dueDate = task.dueDate;
17     const daysUntilDue = Math.ceil((dueDate - now) / (1000 * 60 * 60 * 24));
18
19     if (daysUntilDue < 0) { // Overdue tasks
20       score += 30;
21     } else if (daysUntilDue === 0) { // Due today
22       score += 20;
23     } else if (daysUntilDue <= 2) { // Due in next 2 days
24       score += 15;
25     } else if (daysUntilDue <= 7) { // Due in next week
26       score += 10;
27     }
28   }
29
30   // Reduce score for tasks that are completed or in review
31   if (task.status === TaskStatus.DONE) {
32     score -= 50;
33   } else if (task.status === TaskStatus.REVIEW) {
34     score -= 15;
35   }
36
37   // Boost score for tasks with certain tags
38   if (task.tags.some(tag => ["blocker", "critical", "urgent"].includes(tag))) {
39     score += 8;
40   }
41
42   // Boost score for recently updated tasks
43   const now = new Date();
44   const updatedAt = new Date(task.updatedAt);
45   const daysSinceUpdate = Math.floor((now - updatedAt) / (1000 * 60 * 60 * 24))
46 }
```

```

46     if (daysSinceUpdate < 1) {
47         score += 5;
48     }
49
50     return score;
51 }
```

2 Documented Code

The following is the `calculateTaskScore` function with comprehensive JSDoc annotation.

```

1 /**
2  * Defines the structure of a Task object expected by the scoring function.
3  * @typedef {Object} Task
4  * @property {string|number} priority - The priority level (matches keys in
5  *   TaskPriority).
6  * @property {Date} [dueDate] - The deadline for the task.
7  * @property {string} status - The current workflow status (matches TaskStatus)
8  *   .
9  * @property {string[]} tags - An array of string tags associated with the task
10 *   .
11 * @property {string|Date} updatedAt - The timestamp of the last update.
12 */
13
14 /**
15  * Calculates a relevance score for a task to determine its sort order or
16  * importance.
17 *
18  * The score is derived from a weighted algorithm considering:
19  * 1. Priority: Base score (Low=10 to Urgent=40).
20  * 2. Due Date: Bonuses applied for overdue tasks (+30), due today (+20),
21  *   due within 48h (+15), or due within 7 days (+10).
22  * 3. Status: Penalties applied for 'DONE' (-50) or 'REVIEW' (-15) statuses.
23  * 4. Tags: Bonus (+8) for "blocker", "critical", or "urgent" tags.
24  * 5. Recency: Bonus (+5) if updated within the last 24 hours.
25 *
26  * @param {Task} task - The task object to evaluate. Must contain valid status,
27  *   priority, tags, and update timestamp.
28 *
29  * @returns {number} An integer representing the calculated score. Higher
30  *   numbers
31  *   indicate higher importance/urgency.
32 *
33  * @throws {TypeError} Throws if task is null/undefined or if task.tags is not
34  *   an array.
35  * @throws {ReferenceError} Throws if TaskPriority or TaskStatus enums are not
36  *   defined in the scope.
37 *
38  * @example
39  * const myTask = {
40  *   priority: TaskPriority.HIGH, // Assumes value maps to 3
41  *   dueDate: new Date('2023-12-31'),
42  *   status: TaskStatus.TODO,
43  *   tags: ['critical', 'ui'],
44  *   updatedAt: '2023-12-30T10:00:00Z'
45  * };
46 *
47  * const score = calculateTaskScore(myTask);
48  * // Calculation: Base(30) + Due(15) + Tags(8) + Recency(5) = 58
49 */
50 function calculateTaskScore(task) {
51     // Base priority weights
```

```

45 const priorityWeights = {
46   [TaskPriority.LOW]: 1,
47   [TaskPriority.MEDIUM]: 2,
48   [TaskPriority.HIGH]: 3,
49   [TaskPriority.URGENT]: 4
50 };
51
52 // Calculate base score from priority
53 let score = (priorityWeights[task.priority] || 0) * 10;
54
55 // Add due date factor (higher score for tasks due sooner)
56 if (task.dueDate) {
57   const now = new Date();
58   const dueDate = task.dueDate;
59   // Calculate difference in days (ignoring time of day variations)
60   const daysUntilDue = Math.ceil((dueDate - now) / (1000 * 60 * 60 * 24));
61
62   if (daysUntilDue < 0) { // Overdue tasks
63     score += 30;
64   } else if (daysUntilDue === 0) { // Due today
65     score += 20;
66   } else if (daysUntilDue <= 2) { // Due in next 2 days
67     score += 15;
68   } else if (daysUntilDue <= 7) { // Due in next week
69     score += 10;
70   }
71 }
72
73 // Reduce score for tasks that are completed or in review
74 if (task.status === TaskStatus.DONE) {
75   score -= 50;
76 } else if (task.status === TaskStatus.REVIEW) {
77   score -= 15;
78 }
79
80 // Boost score for tasks with certain tags
81 // Note: Requires task.tags to be an initialized array
82 if (task.tags.some(tag => ["blocker", "critical", "urgent"].includes(tag))) {
83   score += 8;
84 }
85
86 // Boost score for recently updated tasks
87 const now = new Date();
88 const updatedAt = new Date(task.updatedAt);
89 const daysSinceUpdate = Math.floor((now - updatedAt) / (1000 * 60 * 60 * 24))
90 ;
91 if (daysSinceUpdate < 1) {
92   score += 5;
93 }
94
95 return score;
}

```

3 Improvements and Insights

3.1 High-Level Accomplishment

This function implements a weighted scoring algorithm. It converts qualitative task data (priority, deadline, status) into a single quantitative number. This score is used to sort tasks, ensuring the most urgent and important items appear at the top of a list while completed items sink to the bottom.

3.2 Step-by-Step Logic

- **Base Score:** Assigns 10–40 points based on Priority (Low to Urgent).
- **Urgency:** Adds 10–30 points based on the Due Date (Overdue tasks get the highest boost).
- **Status Penalty:** Subtracts points if the task is REVIEW (-15) or DONE (-50) to de-prioritize them.
- **Critical Flags:** Adds 8 points if tags include "blocker", "critical", or "urgent".
- **Recency:** Adds 5 points if the task was updated in the last 24 hours.

3.3 Assumptions & Edge Cases

- **Crash Risk:** Assumes `task.tags` is always an array. If it is null or undefined, the code will crash.
- **Global Dependencies:** Assumes `TaskPriority` and `TaskStatus` enums are defined globally.
- **Date Precision:** Simple division ignores Time Zones and Daylight Savings, potentially shifting "Due Today" items incorrectly.

3.4 Suggested Inline Comments

The following comments clarify complex logic within the function:

```
1 // Fallback to 0 ensures math doesn't result in NaN
2 let score = (priorityWeights[task.priority] || 0) * 10;
3
4 // Math.ceil converts milliseconds to days, ensuring "due in 1 hour" counts as
5 // 1 day
6 const daysUntilDue = Math.ceil((dueDate - now) / (1000 * 60 * 60 * 24));
7
8 // WARNING: potential crash if task.tags is null/undefined
9 if (task.tags.some(...))
```

3.5 Potential Improvements

- **Safety:** Change `task.tags.some` to `task.tags?.some` (optional chaining) to prevent crashes on missing tags.
- **Maintainability:** Move "Magic Numbers" (30, 50, 15) into a configuration object at the top of the file for easier tuning.
- **Accuracy:** Use a library like `date-fns` or UTC timestamps to fix potential Time Zone bugs in the date math.

4 Final Documented Version

The following is the original `calculateTaskScore` function.

```
1 /**
2  * Configuration constants for task scoring weights.
3  * Centralizing these makes the algorithm easier to tune without editing logic.
4  */
5 const SCORING_RULES = {
6     PRIORITY_MULTIPLIER: 10,
```

```

7  DUE_DATE: {
8    OVERDUE: 30,
9    TODAY: 20,
10   WITHIN_48H: 15,
11   WITHIN_WEEK: 10
12 },
13 STATUS_PENALTY: {
14   DONE: -50,
15   REVIEW: -15
16 },
17 TAGS: {
18   KEYWORDS: ["blocker", "critical", "urgent"],
19   BONUS: 8
20 },
21 RECENTY: {
22   DAYS: 1,
23   BONUS: 5
24 }
25 };
26
27 /**
28 * Defines the structure of a Task object expected by the scoring function.
29 * @typedef {Object} Task
30 * @property {string|number} priority - The priority level (matches keys in
31 *   TaskPriority).
32 * @property {Date} [dueDate] - The deadline for the task.
33 * @property {string} status - The current workflow status (matches TaskStatus)
34 * .
35 * @property {string[]} [tags] - An array of string tags associated with the
36 *   task.
37 * @property {string|Date} updatedAt - The timestamp of the last update.
38 */
39
40 /**
41 * Calculates a relevance score for a task using a Weighted Scoring Algorithm.
42 *
43 * The score is derived from five factors:
44 * 1. **Base Priority**: 10-40 points based on severity.
45 * 2. **Urgency**: +10 to +30 points for upcoming or overdue deadlines.
46 * 3. **Status**: Penalties for 'DONE' (-50) or 'REVIEW' (-15) to de-prioritize
47 *   finished work.
48 * 4. **Strategic Tags**: +8 points for "blocker", "critical", or "urgent" tags
49 * .
50 * 5. **Recency**: +5 points if updated within the last 24 hours.
51 *
52 * @param {Task} task - The task object to evaluate.
53 * @returns {number} An integer representing the calculated score. Higher =
54 *   more important.
55 *
56 * @example
57 * const score = calculateTaskScore({
58 *   priority: TaskPriority.HIGH,
59 *   dueDate: new Date(), // Due today
60 *   status: TaskStatus.TODO,
61 *   tags: ['critical'],
62 *   updatedAt: new Date()
63 * });
64 * // Result: 30 (Base) + 20 (Due) + 8 (Tags) + 5 (Recency) = 63
65 */
66 function calculateTaskScore(task) {
67   // Guard clause for safety
68   if (!task) return 0;

```

```

64 // 1. BASELINE: Calculate initial score from priority
65 // Uses a fallback of 0 to ensure valid math if priority is missing
66 const priorityMap = {
67   [TaskPriority.LOW]: 1,
68   [TaskPriority.MEDIUM]: 2,
69   [TaskPriority.HIGH]: 3,
70   [TaskPriority.URGENT]: 4
71 };
72 let score = (priorityMap[task.priority] || 0) * SCORING_RULES.
    PRIORITY_MULTIPLIER;
73
74 // 2. URGENCY: Add bonuses based on timeline
75 if (task.dueDate) {
76   const now = new Date();
77   const dueDate = task.dueDate;
78
79   // Calculate distinct days. Math.ceil ensures "due in 1 hour" counts as 1
    day.
80   // Note: Simple division ignores Daylight Savings Time nuances.
81   const daysUntilDue = Math.ceil((dueDate - now) / (1000 * 60 * 60 * 24));
82
83   if (daysUntilDue < 0) {
84     score += SCORING_RULES.DUE_DATE.OVERDUE;
85   } else if (daysUntilDue === 0) {
86     score += SCORING_RULES.DUE_DATE.TODAY;
87   } else if (daysUntilDue <= 2) {
88     score += SCORING_RULES.DUE_DATE.WITHIN_48H;
89   } else if (daysUntilDue <= 7) {
90     score += SCORING_RULES.DUE_DATE.WITHIN_WEEK;
91   }
92 }
93
94 // 3. LIFECYCLE: Apply penalties to de-prioritize finished work
95 if (task.status === TaskStatus.DONE) {
96   score += SCORING_RULES.STATUS_PENALTY.DONE; // Adding a negative number
97 } else if (task.status === TaskStatus.REVIEW) {
98   score += SCORING_RULES.STATUS_PENALTY.REVIEW;
99 }
100
101 // 4. CONTEXT: Manual overrides for specific flags
102 // Uses optional chaining (?.) to prevent crash if 'tags' is null/undefined
103 if (task.tags?.some(tag => SCORING_RULES.TAGS.KEYWORDS.includes(tag))) {
104   score += SCORING_RULES.TAGS.BONUS;
105 }
106
107 // 5. ACTIVITY: Boost recently active tasks
108 const now = new Date();
109 const updatedAt = new Date(task.updatedAt);
110 const daysSinceUpdate = Math.floor((now - updatedAt) / (1000 * 60 * 60 * 24))
    ;
111
112 if (daysSinceUpdate < SCORING_RULES.RECENCY.DAYS) {
113   score += SCORING_RULES.RECENCY.BONUS;
114 }
115
116 return score;
117

```