Task Management – Proposed UX & Feature Improvements

This document describes proposed changes to evolve the current Task Management feature into an easy-to-use, daily ToDo list system. It is intended as input for Junie AI to assess and implement the required changes.

---

1. Goals

- Reduce friction for adding and updating tasks
- Improve visual clarity and scannability
- Encourage daily usage through good UX feedback
- Keep the system lightweight (not a full project-management tool)

---

2. Core UX Improvements (High Priority)

2.1 Quick Add Task

Description
Add a global or top-level input field:

```
+ Add task…
```

Behavior - Press `Enter` to create task immediately - Auto-assign defaults: - status = `Open` - priority = `Medium` - dueDate = `null`

Optional Enhancements - Simple natural-language parsing: - `Buy milk tomorrow` - `Fix bug !high` - `Prepare slides @work`

Value
Enables extremely fast task capture (key feature of modern ToDo apps).

2.2 Inline Editing

Description
Allow editing without opening the full edit form.

Interactions - Click title → inline text edit - Click priority badge → dropdown - Click status pill → cycle through statuses

Value
Reduces context switching and form fatigue.

---

3. Status & Workflow Enhancements

3.1 Extended Task Statuses

Replace current status model with:

- `Open`
- `In Progress`
- `Done`
- `Archived`

Notes - `Archived` tasks are hidden by default - Status transitions should be quick and reversible

---

3.2 Visual Status Encoding

Visual cues - Colored left border per status - Subtle background tint for `In Progress` - Reduced opacity for `Done`

Example - Open → Blue - In Progress → Yellow - Done → Green

Value
Users can scan task lists without reading text.

---

4. Task List Readability

4.1 View Density Toggle

Add a toggle - Comfortable view (current) - Compact view (single-line rows)

Compact row example

```
☐ Task 3   📅 Today   🗞️ High   In Progress
```

4.2 Relative Due Dates

Replace absolute dates where possible:

- Today
- Tomorrow
- In 3 days
- Overdue (2 days)

Value
Improves urgency perception and planning.

---

5. Filtering & Sorting

5.1 Smart Filter Chips

Replace or complement dropdown filter with quick-access chips:

- All
- Today
- Upcoming
- Overdue
- High Priority

Behavior - One click to apply - Can be combined with sorting

5.2 Persist User Preferences

Persist the following per user: - Selected filter - Sorting order - View mode (compact / comfortable)

Value
Makes the application feel personal and consistent.

---

6. Task Completion Feedback

6.1 Completion Animation

Behavior - Subtle checkmark animation - Task fades out or slides to completed section

Value
Provides positive feedback and motivation.

6.2 Completed Tasks Section

Instead of removing completed tasks immediately:

```
✓ Completed today (3)
```

- Section is collapsible
- Automatically groups by day

---

7. Lightweight Organization

7.1 Tags

Optional feature

- Allow simple tags (e.g. `@work`, `@home`)
- Display as small pills under task title
- Tags are optional, not mandatory

7.2 Due Date Shortcuts

In date picker, add quick actions:

- Today
- Tomorrow
- Next week
- No due date

---

8. Power User Features

8.1 Keyboard Shortcuts

Suggested shortcuts - `Enter` → Add task - `E` → Edit selected task - `D` → Mark done - `/` → Focus quick add

---

8.2 Multi-Select Actions

Enable checkbox selection mode:

- Mark multiple tasks done

- Change priority
- Archive tasks

---

9. Empty States & Guidance

9.1 Helpful Empty States

When no tasks are present, show:

> “You’re all done 🍓
> Add a task to get started.”

Optionally include example tasks.

---

10. Optional Future Enhancements (Out of Scope for MVP)

- Task reminders / notifications
- Recurring tasks (daily / weekly)
- Simple statistics dashboard:
- Tasks completed today
- Tasks completed this week

---

11. Suggested Implementation Order (MVP)

1. Quick Add input
2. Inline editing
3. In Progress status
4. Relative due dates
5. Smart filter chips

These changes alone significantly improve usability and adoption.

---

12. Minimal Data Model Changes

12.1 Task Entity

Extend the existing Task model as follows:

```
Task
- id: UUID
```

```
- title: string
- description: string
- status: enum { OPEN, IN_PROGRESS, DONE, ARCHIVED }
- priority: enum { LOW, MEDIUM, HIGH }
- dueDate: LocalDate | null
- tags: string[]
- createdAt: Instant
- updatedAt: Instant
- completedAt: Instant | null
```

Notes - `completedAt` enables grouping completed tasks by day - `tags` are optional and can be empty

---

13. API Changes (Backend)

13.1 Task Endpoints

Ensure the following operations are supported:

- Create task (minimal payload for Quick Add)
- Partial update (PATCH) for inline editing
- Bulk update (multi-select actions)
- Filter by:
- status
- dueDate (today / overdue / upcoming)
- priority
- tags

13.2 Example PATCH Payload

```
{
  "status": "IN_PROGRESS"
}
```

---

14. Frontend Component Changes

14.1 New / Updated Components

- `QuickAddTaskComponent`
- `TaskListComponent`
- `TaskRowComponent`
- `TaskFilterChipsComponent`
- `CompletedTasksSectionComponent`

14.2 UI State to Persist

Persist per user (localStorage or backend): - selectedFilter - sortOrder - viewMode

---

15. UX Animation & Feedback

15.1 Completion Animation

- Trigger on status change → DONE
- Fade-out + slide-down
- Move task to Completed section

15.2 Empty State Component

Reusable component for: - No tasks - No results after filtering

---

16. Implementation Tickets (Suggested Breakdown)

Ticket 1: Quick Add Task

- Add input field
- Implement Enter-to-create
- Backend support for minimal payload

Ticket 2: Status Model Extension

- Add IN_PROGRESS and ARCHIVED
- Update enums (FE & BE)
- Migration for existing tasks

Ticket 3: Inline Editing

- Title inline edit
- Priority dropdown
- Status cycling

Ticket 4: Relative Due Dates

- Date utility
- Overdue highlighting

Ticket 5: Smart Filters

- Filter chips UI
- Backend query support

Ticket 6: Completed Tasks Section

- Group by completedAt
- Collapsible UI

Ticket 7: View Density Toggle

- Compact vs Comfortable layout
- Persist selection

---

17. Junie AI – Recommended Prompt

Use the following instruction when handing this document to Junie AI:

```
You are an AI software engineer.

Given this specification, assess:
1. Required frontend changes (Angular)
2. Required backend changes (Spring Boot)
3. Necessary data model migrations
4. API changes and compatibility risks
5. Suggested implementation order

Produce:
- A concise change summary
- A list of impacted files/modules
- Any detected inconsistencies or risks
```

---

18. Definition of Done

- Quick Add works without opening a form
- Inline edits require no page reload
- Users can clearly see what to work on today
- Completed tasks provide visible progress
- Default experience feels fast and intuitive

---

19. Phase Plan (MVP vs Enhancements)

Phase 1 (MVP – fastest UX wins)

Deliver the features that convert the screen from a CRUD form into a daily-use ToDo list.

Included: 1. Quick Add input 2. Status model extension (IN_PROGRESS, ARCHIVED) 3. Inline editing (title, priority, status) 4. Relative due dates + overdue highlighting 5. Smart filter chips (All/Today/Upcoming/Overdue/High) 6. Persist UI preferences (local storage or backend)

Out of Phase 1: - Bulk actions - Tags - Compact/Comfortable view toggle - Completed tasks section - Animations (optional, can be minimal)

Phase 2 (Enhancements)

Included: - Completed tasks section (group by day) - Compact vs comfortable view - Multi-select bulk actions - Tags - Completion animations - Empty state refinements

---

20. Angular Implementation Skeleton (Pseudo-code)

> Assumption: Angular + Material (as the UI suggests). Adjust naming if your app uses different patterns.

20.1 Types

```
export type TaskStatus = 'OPEN' | 'IN_PROGRESS' | 'DONE' | 'ARCHIVED';
export type TaskPriority = 'LOW' | 'MEDIUM' | 'HIGH';

export interface Task {
  id: string;
  title: string;
  description: string;
  status: TaskStatus;
  priority: TaskPriority;
  dueDate: string | null; // ISO yyyy-MM-dd
  tags: string[];
  createdAt: string;
  updatedAt: string;
  completedAt: string | null;
}

export type SmartFilter = 'ALL' | 'TODAY' | 'UPCOMING' | 'OVERDUE' | 'HIGH';
export type ViewMode = 'COMFORTABLE' | 'COMPACT';
export type SortOrder = 'DUE_ASC' | 'PRIO_DESC' | 'UPDATED_DESC';
```

20.2 TaskService (API + State)

```
@Injectable({ providedIn: 'root' })
export class TaskService {
  private readonly tasks$ = new BehaviorSubject<Task[]>([]);
```

```
  readonly tasksObs$ = this.tasks$.asObservable();

  constructor(private http: HttpClient) {}

  load(params: { status?: TaskStatus; smartFilter?: SmartFilter; sort?:
SortOrder }): Observable<Task[]> {
    return this.http.get<Task[]>('/api/tasks', { params:
toHttpParams(params) }).pipe(
      tap(tasks => this.tasks$.next(tasks))
    );
  }

  quickAdd(title: string): Observable<Task> {
    const payload = { title };
    return this.http.post<Task>('/api/tasks', payload).pipe(
      tap(task => this.tasks$.next([task, ...this.tasks$.value]))
    );
  }

  patch(id: string, patch: Partial<Task>): Observable<Task> {
    return this.http.patch<Task>(`/api/tasks/${id}`, patch).pipe(
      tap(updated => this.tasks$.next(this.tasks$.value.map(t => t.id === id ?
updated : t)))
    );
  }

  bulkUpdate(ids: string[], patch: Partial<Task>): Observable<Task[]> {
    return this.http.patch<Task[]>('/api/tasks/bulk', { ids, patch }).pipe(
      tap(updatedList => {
        const map = new Map(updatedList.map(t => [t.id, t]));
        this.tasks$.next(this.tasks$.value.map(t => map.get(t.id) ?? t));
      })
    );
  }
}
```

20.3 QuickAddTaskComponent

```
<mat-form-field appearance="outline" class="quick-add">
  <mat-label>+ Add task…</mat-label>
  <input matInput [formControl]="ctrl" (keydown.enter)="submit()" />
</mat-form-field>
```

```
ctrl = new FormControl('', { nonNullable: true });
```

```
submit() {
  const text = this.ctrl.value.trim();
  if (!text) return;

  // Phase 1: no parsing
  this.taskService.quickAdd(text).subscribe({
    next: () => this.ctrl.setValue('')
  });
}
```

20.4 TaskRowComponent (Inline Editing)

Inline title:

```
<div class="task-row" [class.compact]="viewMode=='COMPACT'">
  <button mat-icon-button (click)="toggleDone()" aria-label="toggle done">
    <mat-icon>{{ task.status==='DONE' ? 'check_box' :
'check_box_outline_blank' }}</mat-icon>
  </button>

  <div class="title" (dblclick)="startEditTitle()">
    <ng-container *ngIf="!editingTitle; else editTpl">
      {{ task.title }}
    </ng-container>
    <ng-template #editTpl>
      <input [value]="task.title"
(keydown.enter)="saveTitle($event.target.value)" (blur)="cancelEditTitle()" />
    </ng-template>
  </div>

  <button mat-stroked-button (click)="cycleStatus()">{{ task.status }}</button>
  <button mat-stroked-button [matMenuTriggerFor]="prioMenu">{{ task.priority }}
</button>
  <mat-menu #prioMenu="matMenu">
    <button mat-menu-item (click)="setPriority('LOW')">LOW</button>
    <button mat-menu-item (click)="setPriority('MEDIUM')">MEDIUM</button>
    <button mat-menu-item (click)="setPriority('HIGH')">HIGH</button>
  </mat-menu>

  <div class="due" [class.overdue]="isOverdue(task.dueDate)">
    {{ formatRelativeDue(task.dueDate) }}
  </div>
</div>
```

```
cycleStatus() {
  const next = nextStatus(this.task.status);
  this.taskService.patch(this.task.id, { status: next, completedAt: next ===
'DONE' ? new Date().toISOString() : null })
    .subscribe();
}

setPriority(p: TaskPriority) {
  this.taskService.patch(this.task.id, { priority: p }).subscribe();
}

toggleDone() {
  const next = this.task.status === 'DONE' ? 'OPEN' : 'DONE';
  this.taskService.patch(this.task.id, { status: next, completedAt: next ===
'DONE' ? new Date().toISOString() : null })
    .subscribe();
}
```

20.5 Filter Chips Component

```
<div class="chips">
  <button mat-button [class.active]="filter=='ALL'"
(click)="set('ALL')">All</button>
  <button mat-button [class.active]="filter=='TODAY'"
(click)="set('TODAY')">Today</button>
  <button mat-button [class.active]="filter=='UPCOMING'"
(click)="set('UPCOMING')">Upcoming</button>
  <button mat-button [class.active]="filter=='OVERDUE'"
(click)="set('OVERDUE')">Overdue</button>
  <button mat-button [class.active]="filter=='HIGH'"
(click)="set('HIGH')">High</button>
</div>
```

---

21. Date Utilities (Relative Due + Overdue)

```
export function isOverdue(iso: string | null): boolean {
  if (!iso) return false;
  const due = new Date(iso + 'T00:00:00');
  const today = startOfToday();
  return due < today;
}

export function formatRelativeDue(iso: string | null): string {
```

```
  if (!iso) return 'No due date';
  const due = new Date(iso + 'T00:00:00');
  const today = startOfToday();
  const diffDays = Math.round((due.getTime() - today.getTime()) / (24 * 3600 *
1000));

  if (diffDays === 0) return 'Today';
  if (diffDays === 1) return 'Tomorrow';
  if (diffDays === -1) return 'Overdue (1 day)';
  if (diffDays < 0) return `Overdue (${Math.abs(diffDays)} days)`;
  return `In ${diffDays} days`;
}

function startOfToday(): Date {
  const d = new Date();
  d.setHours(0,0,0,0);
  return d;
}

export function nextStatus(s: TaskStatus): TaskStatus {
  switch (s) {
    case 'OPEN': return 'IN_PROGRESS';
    case 'IN_PROGRESS': return 'DONE';
    case 'DONE': return 'OPEN';
    case 'ARCHIVED': return 'OPEN';
  }
}
```

---

22. Spring Boot Implementation Skeleton (Pseudo-code)

22.1 Enums

```
public enum TaskStatus { OPEN, IN_PROGRESS, DONE, ARCHIVED }
public enum TaskPriority { LOW, MEDIUM, HIGH }
```

22.2 Entity

```
@Entity
@Table(name = "tasks")
public class TaskEntity {
  @Id
  private UUID id;
```

```
  @Column(nullable = false)
  private String title;

  @Column(nullable = false)
  private String description;

  @Enumerated(EnumType.STRING)
  @Column(nullable = false)
  private TaskStatus status;

  @Enumerated(EnumType.STRING)
  @Column(nullable = false)
  private TaskPriority priority;

  private LocalDate dueDate;

  @ElementCollection
  @CollectionTable(name = "task_tags", joinColumns = @JoinColumn(name =
"task_id"))
  @Column(name = "tag")
  private List<String> tags = new ArrayList<>();

  private Instant createdAt;
  private Instant updatedAt;
  private Instant completedAt;
}
```

22.3 DTOs

```
public record TaskCreateRequest(String title, String description, LocalDate
dueDate, TaskPriority priority, TaskStatus status, List<String> tags) {}
public record TaskQuickCreateRequest(String title) {}
public record TaskPatchRequest(String title, String description, LocalDate
dueDate, TaskPriority priority, TaskStatus status, Instant completedAt,
List<String> tags) {}

public record BulkPatchRequest(List<UUID> ids, TaskPatchRequest patch) {}
```

Rules - Quick create accepts only `title`. Backend fills defaults. - PATCH applies only non-null fields.

22.4 Controller

```
@RestController
@RequestMapping("/api/tasks")
public class TaskController {
```

```
  @GetMapping
  public List<TaskDto> list(
      @RequestParam(required = false) TaskStatus status,
      @RequestParam(required = false) String smartFilter,
      @RequestParam(required = false) String sort
  ) { ... }

  @PostMapping
  public TaskDto create(@RequestBody Map<String, Object> body) {
    // If body contains only title -> treat as quick create
    // Else -> full create
    ...
  }

  @PatchMapping("/{id}")
  public TaskDto patch(@PathVariable UUID id, @RequestBody TaskPatchRequest
patch) { ... }

  @PatchMapping("/bulk")
  public List<TaskDto> bulkPatch(@RequestBody BulkPatchRequest req) { ... }
}
```

22.5 Filtering Rules (Smart Filters)

- TODAY: dueDate == today AND status != DONE AND status != ARCHIVED
- UPCOMING: dueDate > today AND status != DONE AND status != ARCHIVED
- OVERDUE: dueDate < today AND status != DONE AND status != ARCHIVED
- HIGH: priority == HIGH AND status != DONE AND status != ARCHIVED

---

23. DB Migration Notes

If you currently store status as `OPEN/DONE` only: - Add `IN_PROGRESS`, `ARCHIVED` to enum - Set default for existing tasks: - status remains as-is - Add `completedAt` nullable column - Add tags table only when enabling tags (Phase 2)

---

24. “After” UI Target (Textual Mock)

Top area:

- Title: **Task Management**
- Quick Add: `+ Add task…`
- Smart filters: `[All] [Today] [Upcoming] [Overdue] [High]`
- Sorting dropdown: `Sort: Due date` + reset icon

Task row (compact):

```
☐ Fix login redirect     Today     HIGH     IN PROGRESS
```

Completed section:

```
✓ Completed today (3)  [Collapse]
```

---

25. Junie AI – Extended Prompt (Implementation Mode)

```
You are an AI software engineer working on an Angular + Spring Boot app.

Use this spec to:
1) Produce an implementation plan split into Phase 1 and Phase 2.
2) Identify backend changes: entity, DTOs, controller endpoints, filtering
logic.
3) Identify frontend changes: components, services, state handling, UI
persistence.
4) Provide concrete code diffs or pseudo-code per ticket.
5) List potential merge conflicts and safe migration strategy.

Output format:
- Phase 1: numbered steps + impacted files
- Phase 2: numbered steps + impacted files
- API contract table (endpoint, method, request, response)
- Risks / edge cases
```

---

26. Definition of Done (Extended)

- Phase 1 runs end-to-end (FE + BE) with no manual refresh
- Quick Add creates tasks in < 1 second perceived time
- Inline title edit and status cycle work reliably
- Relative due dates show “Today/Tomorrow/Overdue” correctly
- Filters are one-click and persisted
- Unit tests cover:
- smart filter semantics
- patch behavior (only non-null fields)

End of Document