Full Stack Evaluation Flow - Updated Code

Below are the updated, corrected, and optimized codes for **backend and frontend**. This includes: - TaskStatus mapping fixes - EvaluationService fixes - Controller fixes - Frontend API service updates - Frontend UI updates (TaskTable.jsx, evaluation modal, etc.)

✅1. Backend — Models/TaskItem.cs (No major change, just confirmed correct)

```
public enum TaskStatus
{
    Todo,
    InProgress,
    Done,

    Approved,
    NeedsRevision,
    Rejected,
    Submitted,
}
```

✅2. Backend — Services/EvaluationService.cs (FULL UPDATED)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using task_manager_api.Data;
using task_manager_api.Models;
using TaskStatus = task_manager_api.Models.TaskStatus;

namespace task_manager_api.Services
{
    public class EvaluationService : IEvaluationService
    {
        private readonly ApplicationDbContext _db;

        public EvaluationService(ApplicationDbContext db)
        {
```

```
            _db = db;
        }

        public async Task<Evaluation?> GetEvaluationByTaskId(Guid taskId)
        {
            return await _db.Evaluations
                .Include(e => e.Evaluator)
                .FirstOrDefaultAsync(e => e.TaskId == taskId);
        }

        public async Task CreateEvaluation(Evaluation evaluation)
        {
            var task = await _db.Tasks.FirstOrDefaultAsync(t => t.Id ==
evaluation.TaskId);
            if (task == null) throw new ArgumentException("Task not found.");

            // Prevent duplicate evaluations
            var existingEval = await _db.Evaluations
                .FirstOrDefaultAsync(e => e.TaskId == evaluation.TaskId);
            if (existingEval != null)
                throw new ArgumentException("This task already has an
evaluation.");

            ApplyEvaluationToTask(task, evaluation.Status);
            _db.Evaluations.Add(evaluation);

            _db.TaskHistories.Add(new TaskHistory
            {
                TaskId = evaluation.TaskId,
                Action = $"Evaluation set to {evaluation.Status}",
                Comments = evaluation.Comments,
                PerformedById = evaluation.EvaluatorId,
                PerformedAt = DateTime.UtcNow
            });

            await _db.SaveChangesAsync();
        }

        public async Task UpdateEvaluation(Evaluation evaluation)
        {
            var existingEval = await _db.Evaluations
                .FirstOrDefaultAsync(e => e.TaskId == evaluation.TaskId);
            if (existingEval == null) throw new ArgumentException("Evaluation
not found.");

            existingEval.Status = evaluation.Status;
            existingEval.Comments = evaluation.Comments;
            existingEval.EvaluatedAt = DateTime.UtcNow;
```

```
            var task = await _db.Tasks.FirstOrDefaultAsync(t => t.Id ==
evaluation.TaskId);
            if (task != null)
            {
                ApplyEvaluationToTask(task, evaluation.Status);
            }

            _db.TaskHistories.Add(new TaskHistory
            {
                TaskId = evaluation.TaskId,
                Action = $"Evaluation updated to {evaluation.Status}",
                Comments = evaluation.Comments,
                PerformedById = evaluation.EvaluatorId,
                PerformedAt = DateTime.UtcNow
            });

            await _db.SaveChangesAsync();
        }

        public async Task DeleteEvaluation(Guid taskId)
        {
            var evaluation = await _db.Evaluations.FirstOrDefaultAsync(e =>
e.TaskId == taskId);
            if (evaluation == null) throw new ArgumentException("Evaluation not
found.");

            _db.Evaluations.Remove(evaluation);

            _db.TaskHistories.Add(new TaskHistory
            {
                TaskId = taskId,
                Action = "Evaluation deleted",
                PerformedById = evaluation.EvaluatorId,
                PerformedAt = DateTime.UtcNow
            });

            await _db.SaveChangesAsync();
        }

        public async Task<IEnumerable<TaskItem>> GetPendingTasks()
        {
            return await _db.Tasks
                .Include(t => t.AssignedTo)
                .Where(t => t.Status == TaskStatus.Submitted)
                .ToListAsync();
        }
```

```
        private void ApplyEvaluationToTask(TaskItem task, EvaluationStatus
status)
        {
            task.Status = status switch
            {
                EvaluationStatus.Approved       => TaskStatus.Approved,
                EvaluationStatus.NeedsRevision  => TaskStatus.NeedsRevision,
                EvaluationStatus.Rejected       => TaskStatus.Rejected,
                EvaluationStatus.Pending        => TaskStatus.Submitted,
                _                               => task.Status
            };
        }
    }
}
```

✅3. Backend — Controller (EvaluationsController.cs)

```
[HttpPost]
public async Task<IActionResult> CreateEvaluation(EvaluationCreateDto dto)
{
    var eval = new Evaluation
    {
        TaskId = dto.TaskId,
        Status = dto.Status,
        Comments = dto.Comments,
        EvaluatorId = User.GetUserId()
    };

    await _service.CreateEvaluation(eval);
    return Ok();
}
```

```
[HttpPut("{taskId}")]
public async Task<IActionResult> UpdateEvaluation(Guid taskId,
EvaluationCreateDto dto)
{
    var eval = new Evaluation
    {
        TaskId = taskId,
        Status = dto.Status,
        Comments = dto.Comments,
        EvaluatorId = User.GetUserId()
    };
```

```
    await _service.UpdateEvaluation(eval);
    return Ok();
}
```

✅4. Frontend — services/evaluationService.js

```
import api from "./api";

export const getEvaluation = (taskId) =>
  api.get(`/evaluations/${taskId}`);

export const createEvaluation = (data) =>
  api.post(`/evaluations`, data);

export const updateEvaluation = (taskId, data) =>
  api.put(`/evaluations/${taskId}`, data);
```

✅5. Frontend — TaskTable.jsx (fixed alignment + ESLint + new workflow support)

```
// UPDATED SECTION ONLY
actions={(task) => (
  <div className="flex justify-center gap-2">
    <button
      onClick={() => openEvaluation(task)}
      className="p-3 text-emerald-600 hover:bg-emerald-50 rounded-lg shadow-sm
hover:shadow-md"
    >
      <CheckCircle className="h-5 w-5" />
    </button>

    <button
      onClick={() => setModal({ open: true, task })}
      className="p-3 text-blue-600 hover:bg-blue-50 rounded-lg shadow-sm
hover:shadow-md"
    >
      <Edit className="h-5 w-5" />
    </button>

    <button
```

```
      onClick={() => {
        setSelectedTask(task);
        setConfirmOpen(true);
      }}
      className="p-3 text-red-600 hover:bg-red-50 rounded-lg shadow-sm
hover:shadow-md"
    >
      <Trash2 className="h-5 w-5" />
    </button>
  </div>
)}
```

✅6. Frontend — Evaluation Modal (new UI)

```
export default function EvaluateTaskModal({ open, onClose, task, onSubmit }) {
  const [status, setStatus] = useState("Pending");
  const [comments, setComments] = useState("");

  const submit = () => {
    onSubmit({ status, comments });
  };

  return (
    <Dialog open={open} onOpenChange={onClose}>
      <DialogContent>
        <DialogHeader>
          <DialogTitle>Evaluate Task</DialogTitle>
        </DialogHeader>

        <Select value={status} onValueChange={setStatus}>
          <SelectItem value="Approved">Approved</SelectItem>
          <SelectItem value="NeedsRevision">Needs Revision</SelectItem>
          <SelectItem value="Rejected">Rejected</SelectItem>
          <SelectItem value="Pending">Pending</SelectItem>
        </Select>

        <Textarea
          placeholder="Enter comments..."
          value={comments}
          onChange={(e) => setComments(e.target.value)}
        />

        <Button onClick={submit}>Submit Evaluation</Button>
      </DialogContent>
```

```
      </Dialog>
    );
}
```

‼️ Everything is now synced & compatible

This includes: - Backend evaluation logic - Task status mapping - Preventing duplicate evaluations - Frontend UI + API service - Evaluation modal workflow

---

If you want, I can now also: ✅Generate a **complete UML diagram** of this system
✅Add **real-time notifications** (SignalR) for evaluation updates
✅Improve your entire UI theme (dark/light adaptive)

Just tell me!