# Неделя 2. Лаба 1. ASP.NET: запускаем сервис задач и добавляем свою функцию

*Часть 2 из 2: разделы «5. Запрос отладчиком» — «⭐ Дополнительно». Первая часть лабы — в материалах предыдущей недели; перед продолжением выполните подготовку ниже.*

**Цель:** открыть проект в Visual Studio, запустить сервис и обратиться к его API из Postman, пройти запрос отладчиком через все слои и добавить новую функцию с тестами. **Время:** 120 мин. **Теория:** лекция 1. **Инструменты:** Visual Studio 2026, Postman, браузер.

## 5. Запрос отладчиком

Поставьте точки останова (**F9**):

- `TasksController.ChangeStatus`;
- `TaskService.GetAsync` и `TaskService.ChangeStatusAsync`;
- `InMemoryTaskRepository.GetAsync` и `InMemoryTaskRepository.SetStatusAsync` (папка `Data/InMemory`).

Создайте проект и задачу. Отправьте `PATCH` сначала со `"Done"`, затем с `"InProgress"`. Проходите по шагам (**F10** — шаг, **F11** — зайти в метод) и смотрите окна **Call Stack** и **Locals**.

При `"Done"` посмотрите в **Locals** на `task.Status` и дойдите до строки с `StatusRules.CanTransition`: переход запрещён, и контроллер возвращает `Problem(..., 409)`. До `ChangeStatusAsync` дело не дойдёт.

**Запишите для обоих запросов** цепочку вызовов и строку, где решается код ответа.

**Зачем:** умение пройти запрос отладчиком — самый быстрый способ понять чужой проект.

## 6. Чтение кода

Найдите в коде и ответьте одним-двумя предложениями:

| # | Вопрос |
|---|---|
| 1 | Где задан маршрут `api/tasks/{id}/status`? |
| 2 | Кто возвращает 400 при пустом `title`, если в контроллере нет проверки? |
| 3 | Где правило разрешённых переходов статусов и как его протестировать без HTTP? |
| 4 | Где решается, что ответ будет 404? Откуда в ответе 409 берётся текст `detail`? |
| 5 | Где выбирается хранилище и какое выбрано сейчас? Какая настройка это решает? |
| 6 | Почему `TaskService` не изменится, когда мы переключимся на PostgreSQL? |

## 7. Тесты

Остановите отладку (**Shift+F5**): пока сервис запущен, его DLL заблокированы и тесты не соберутся. Затем **Test → Test Explorer → Run All** (или `dotnet test` в терминале из `C:\dev\demo-api`).

Ожидаемо: **22 теста** (7 юнит + 15 интеграционных), все зелёные. БД не нужна: интеграционные тесты сейчас работают с хранилищем в памяти.

Откройте `tests/DemoApi.Tests/Integration/TasksApiTests.cs` и найдите тест, который проверяет запрет перехода `Todo → Done`.

## 8. Своя функция: статистика проекта

**Задача:** `GET /api/projects/{id}/stats` возвращает сводку по задачам проекта.

```
{ "projectId": 1, "total": 3, "todo": 2, "inProgress": 1, "done": 0, "overdue": 1 }
```

`overdue` — задачи со сроком раньше сегодняшнего дня и статусом не `Done`. Нет проекта → **404**.

Сделайте по чек-листу из лекции, слой за слоем:

1. **Domain:** `record ProjectStats(int ProjectId, int Total, int Todo, int InProgress, int Done, int Overdue)`. Эту запись сервис и отдаст клиенту в JSON. Добавьте над ней `/// <summary>` и `/// <param>` для каждого поля — по образцу `Project.cs`: тексты появятся в Swagger.
2. **Data:** метод `Task<ProjectStats> GetStatsAsync(int projectId, CancellationToken ct)` в интерфейсе `IProjectRepository` и в **обеих** реализациях:
   - `InMemoryProjectRepository` — через LINQ, внутри `lock (store.Sync)`. Просрочка: `t.DueDate < DateOnly.FromDateTime(DateTime.UtcNow) && t.Status != TaskItemStatus.Done`;
   - `ProjectRepository` (PostgreSQL) — SQL ниже, по образцу `ProjectRepository.GetAsync`. Он заработает в теме 3, и тогда тесты проверят и его.

   ```
   SELECT count(*),
          count(*) FILTER (WHERE status = 'todo'),
          count(*) FILTER (WHERE status = 'in_progress'),
          count(*) FILTER (WHERE status = 'done'),
          count(*) FILTER (WHERE due_date < current_date AND status <> 'done')
   FROM tasks
   WHERE project_id = $1
   ```

   Параметр — `cmd.Parameters.Add(Sql.Param(projectId, NpgsqlDbType.Integer))`, перед чтением — `await reader.ReadAsync(ct)`. `count(*)` возвращает `bigint`, поэтому `(int)reader.GetInt64(0)`.
3. **Services:** `ProjectService.GetStatsAsync` — просто вызывает репозиторий.
4. **Controllers:** действие `[HttpGet("{id:int}/stats")]` в `ProjectsController`. Сначала `projects.GetAsync(id, ct)`: нет проекта → `return NotFound();` (иначе клиент получит статистику из нулей для несуществующего проекта). Затем `return await projects.GetStatsAsync(id, ct);`. Над действием — XML-комментарий по образцу соседних: `/// <summary>`, `/// <param>`, `/// <response code="404">`.
5. **Tests** в `ProjectsApiTests` (добавьте в начало файла `using DemoApi.Domain;` — там `ProjectStats` и `TaskItemStatus`):
   - используйте хелперы из `Integration/TestApi.cs`: `CreateProjectAsync`, `CreateTaskAsync`, `ChangeStatusAsync`, `TestApi.Json`;
   - задачу со сроком создайте напрямую: `_client.PostAsJsonAsync("/api/tasks", new { projectId = project.Id, title = "Просрочена", dueDate = DateOnly.FromDateTime(DateTime.UtcNow.AddDays(-2)) });`
   - проект с тремя задачами: одна в `InProgress`, одна со сроком позавчера → проверить все числа;
   - неизвестный проект → 404.
6. **`DemoApi.http`:** добавьте запрос статистики.

Проверьте функцию запросом из Postman, а затем откройте Swagger: новая операция появится на странице сама, вместе с вашим описанием, кодами ответов и полями — описание строится по коду контроллера и XML-комментариям.

## Сдача

- [ ] Скриншот коллекции `demo-api` в Postman
- [ ] Коды ответов, `errors` и `detail` из п. 3
- [ ] Ответ на вопрос из п. 4
- [ ] Две цепочки вызовов из п. 5
- [ ] Ответы на 6 вопросов из п. 6
- [ ] Ответ `GET /api/projects/{id}/stats` для вашего проекта
- [ ] Все 24 теста зелёные, включая два новых

## ☆ Дополнительно

Правило: **нельзя создать задачу со сроком в прошлом** → 400. Решите, в каком слое его проверять: атрибут на записи запроса `CreateTaskRequest` или проверка в контроллере. Обоснуйте выбор, реализуйте и напишите тест.