Manage Services Module — Step-by-step Implementation Plan

This plan breaks the work into ordered, actionable steps with file targets, code notes, commands, and verification checks.

---

0. Preparation

1. Create a feature branch: feature/manage-services.
2. Ensure solution builds: dotnet build.
3. Confirm DB migrations can run locally and you have a test provider user.

---

1. Domain layer — Add Service entity

Files: Domain/Entities/Service.cs

Steps:

1. Add Service class with properties:
    - Guid Id (PK)
    - string ProviderId
    - Guid CategoryId
    - string ServiceName (required)
    - string? Description
    - decimal Price (required)
    - string PricingUnit (enum or string: "Per Project", "Per Hour", "Per Day")
    - string? ImageUrls (JSON array stored as text)
    - bool IsActive (default true)
    - bool IsDeleted (default false)
    - DateTime CreatedAt (set in constructor)
    - DateTime? UpdatedAt
2. Add constructor or factory to set Id and CreatedAt.
3. Add helpful methods: MarkDeleted(), SetActive(bool), AddImageUrls(List<string>) (optional).

Design note: keep ImageUrls as JSON string for simplicity; repository can serialize/deserialize to List<string>.

---

2. Application layer — DTO and repository interface

Files: Application/DTOs/ServiceDto.cs, Application/Interfaces/IServiceRepository.cs

Steps:

1. Create ServiceDto with fields matching UI needs plus server-only metadata:
    - Guid? Id
    - Guid CategoryId
    - string ServiceName
    - string? Description
    - decimal Price
    - string PricingUnit
    - IFormFileCollection? Images or List<IFormFile> for uploads
    - List<string>? ExistingImageUrls for edit
    - bool IsActive
2. Create IServiceRepository methods:
    - Task<Service?> GetAsync(Guid id)
    - Task<IEnumerable<Service>> GetByProviderAsync(string providerId, bool includeDeleted = false)
    - Task<IEnumerable<Service>> GetPublicActiveByCategoryAsync(Guid categoryId)
    - Task AddAsync(Service service)
    - Task UpdateAsync(Service service)
    - Task SoftDeleteAsync(Guid id)
    - Task SaveChangesAsync() (or rely on DbContext commit pattern)

---

3. Infrastructure — DbContext, Repository, Migration

Files: Infrastructure/Persistence/ApplicationDbContext.cs, Infrastructure/Repositories/ServiceRepository.cs, Migrations/*

Steps:

1. Modify ApplicationDbContext:
    - Add public DbSet<Service> Services { get; set; }
    - Configure entity via OnModelCreating (required fields, max lengths, default values, indexes on ProviderId, IsDeleted)
2. Implement ServiceRepository:
    - Inject ApplicationDbContext.
    - Implement methods from IServiceRepository using EF Core queries that always filter IsDeleted == false unless includeDeleted=true.
    - Handle serialization: map List<string> ↔ ImageUrls JSON string.
    - For SoftDeleteAsync, set IsDeleted=true, IsActive=false, update UpdatedAt.
3. Create and apply migration:
    - dotnet ef migrations add AddServiceEntity
    - dotnet ef database update

Verification: ensure migration creates correct columns and default values.

---

4. Web layer — register services, controller, views

Files: Web/Program.cs, Web/Controllers/Provider/ServiceController.cs, Web/Views/Provider/Service/MyServices.cshtml, _CreateEditModal.cshtml

Steps:

1. Register repository in DI (Program.cs):
    - services.AddScoped<IServiceRepository, ServiceRepository>();
2. Create ServiceController (area: Provider or under /Provider):
    - Actions: Index() (list provider services), Get(Guid id) (GET single), Create(IFormCollection or ServiceDto), Edit(ServiceDto), SoftDelete(Guid id), ToggleActive(Guid id).
    - Map ServiceDto ↔ Service (explicit mapping inside controller or use AutoMapper). When handling files:
        - Save uploaded files to storage (local or cloud), collect URLs, merge with ExistingImageUrls.
        - Persist ImageUrls JSON to entity.

3. Views:
    - MyServices.cshtml: list with state column (Active/Inactive/Deleted not shown), actions: Create, Edit (modal), Soft Delete, Toggle Active.
    - _CreateEditModal.cshtml: form uses enctype="multipart/form-data", fields for ServiceName, CategoryId dropdown, Price, PricingUnit radio/select, Description, file input for multiple images, checkbox for IsActive.
4. Client behavior:
    - When editing, show existing images with option to remove; send ExistingImageUrls for retention.

---

5. Mapping strategy

Options:

- Manual mapping in controller (simple, clear). Show helper methods MapDtoToEntity and MapEntityToDto.
- Or add AutoMapper profile: ServiceProfile mapping Service ⇄ ServiceDto and custom ImageUrls conversions.

Recommendation: start manual, add AutoMapper later if mappings grow.

---

6. Tests & Verification

Commands:

- dotnet build
- dotnet ef database update

Manual test checklist (as provider):

1. Create service with images → appears in MyServices, shows images, IsActive=true.
2. Edit service: add/remove images, change price/category → changes persist, UpdatedAt set.
3. Toggle IsActive → provider can see it in list; public listing hides inactive.
4. Soft-delete service → disappears for provider listing when filtering deleted; verify IsDeleted=true in DB and not returned to public.
5. Public endpoint: only returns IsActive=true && IsDeleted=false.
6. Migration rollback and forward tests.