Edit

Creating Person Application Service

An Application Service is used from client (presentation layer) to perform operations (use cases) of the application.

Application service interface and DTOs are located in **.Application.Shared** project. We are creating an application service to get people from the server. So, we're first creating an **interface** to define the person application service (while this interface is optional, we suggest you to create it):

C# Copy

```
using Abp.Application.Services;
using Abp.Application.Services.Dto;

namespace Acme.PhoneBookDemo.PhoneBook
{
    public interface IPersonAppService : IApplicationService
    {
        ListResultDto<PersonListDto> GetPeople(GetPeopleInput input);
    }
}
```

Feedback

An application service method gets/returns **DTO**s. **ListResultDto** is a pre-build helper DTO to return a list of another DTO. **GetPeopleInput** is a DTO to pass request parameters to **GetPeople** method. So, GetPeopleIntput and PersonListDto are defined as shown below:

```
public class GetPeopleInput
{
    public string Filter { get; set; }
}

public class PersonListDto : FullAuditedEntityDto
{
    public string Name { get; set; }

    public string Surname { get; set; }
```

CustomDtoMapper.cs is used to create mapping from **Person** to **PersonListDto**. **FullAuditedEntityDto** is inherited to implement audit properties automatically. See application service and DTO documentations for more information. We are adding the following mappings.

```
...
// PhoneBook (we will comment out other lines when the new DTOs are added)
configuration.CreateMap<Person, PersonListDto>();
//configuration.CreateMap<AddPhoneInput, Phone>();
//configuration.CreateMap<CreatePersonInput, Person>();
//configuration.CreateMap<Person, GetPersonForEditOutput>();
//configuration.CreateMap<Phone, PhoneInPersonListDto>();
```

After defining interface, we can implement it as shown below: (in **.Application** project)

```
public class PersonAppService : PhoneBookDemoAppServiceBase, IPersonAppService
{
    private readonly IRepository<Person> _personRepository;

    public PersonAppService(IRepository<Person> personRepository)
    {
        _personRepository = personRepository;
    }

    public ListResultDto<PersonListDto> GetPeople(GetPeopleInput input)
    {
        var people = _personRepository
            .GetAll()
            .WhereIf(
                !input.Filter.IsNullOrEmpty(),
                p => p.Name.Contains(input.Filter) ||
                     p.Surname.Contains(input.Filter) ||
                     p.EmailAddress.Contains(input.Filter)
            )
            .OrderBy(p => p.Name)
            .ThenBy(p => p.Surname)
            .ToList();

        return new ListResultDto<PersonListDto>(ObjectMapper.Map<List<PersonListDto
    }
}
```

We're injecting **personRepository** (it's automatically created by ABP) and using it to filter and get people from database.

WhereIf is an extension method here (defined in Abp.Linq.Extensions namespace). It performs Where condition, only if filter is not null or empty. **IsNullOrEmpty** is also an extension method (defined in Abp.Extensions namespace). ABP has many similar shortcut extension methods. **ObjectMapper.Map** method automatically converts list of Person entities to list of PersonListDto objects with using configurations in **CustomDtoMapper.cs** in **.Application** project.

Connection & Transaction Management

We don't manually open database connection or start/commit transactions manually. It's automatically done with ABP framework's Unit Of Work system. See UOW documentation for more.

Exception Handling

We don't handle exceptions manually (using a try-catch block). Because ABP framework automatically handles all exceptions on the web layer and returns appropriate error messages to the client. It then handles errors on the client and shows needed error information to the user. See exception handling document for more.

Next

- Using GetPeople Method From Angular Component