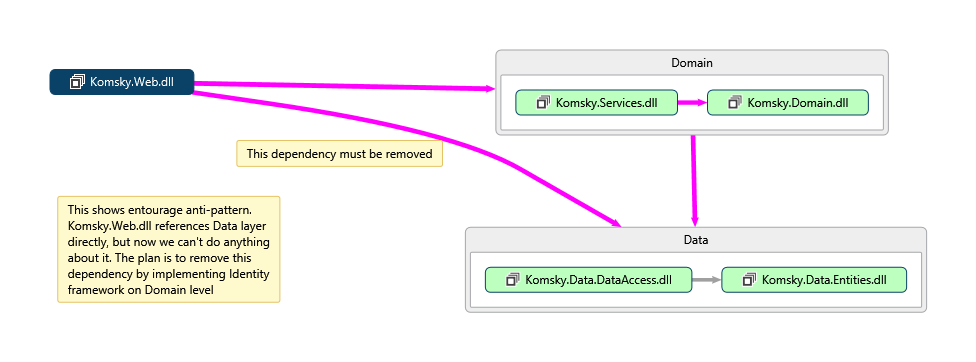
# Lab 09 B - Identity

*This lab is optional as it might take as long as 3 hours. Consider writing this code whey students still have plenty of time (eg. it’s Thursday morning).*

## Content

On Module 01 I’ve explained some design principles by using patterns and removing dependencies. But we didn’t removed dependencies correctly, because it required knowledge about Identity framework. In this Lab we implement Identity on service layer and remove dependency on Data layer, so it can be easily replaced with NHibernate or any ORM.



## General steps

1. Removing dependency on Data layer
2. Implementing Identity
3. Removing references to truly use Stairway pattern

## Task 1 – Removing dependency on Data Layer

If you have Visual Studio 2013 Enterprise or above, you can use Code Map feature to see dependencies visually. Use points from 1 to 2 to see them. If you don’t have Enterprise or Ultimate edition, start on point 3.

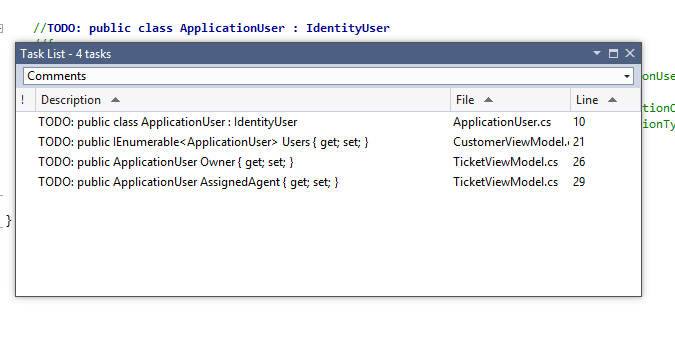
1. Right click on **Komsky.Web** project and choose *Show on Code Map*
2. Repeat this step with **Komsky.Services** and **Komsky.Data.DataAccess** to see dependencies on diagram.
3. Go to **Komsky.Web/References** and remove following references:
   1. Komsky.Data
   2. Komsky.Data.DataAccess
   3. Komsky.Data.Entities
4. Build and debug your project. What build errors have Visual Studio found? Let’s try to fix some of them:
   1. First error might be using Data Façade class for HomeController. Delete usings and objects from NinjectWebCommon and HomeController, as we don’t use it anymore.
   2. Briefly walk through the rest of the errors and delete all the usings that point to any Komsky.Data namespaces.

That’s it. We have successfully removed dependency on **Komsky.Data** layer. Or did we? Now the solution won’t build. So actually, we have just removed references, not dependencies! To fully remove dependency, we need to replace ApplicationUser class with another one – a new one, from domain level.

## Task 2 – Implementing Identity on domain level

In this task we focus on implementing only most basic required fields for user and role, to make solution work. For the simplicity, I keep Identity implementation in Domain and Services projects, but you could create new project for this purpose.

1. Open **Komsky.Data.Entities** project, and **ApplicationUser.cs** file inside it
2. Temporarily comment out ApplicationUser class, adding *TODO:* declaration for easier locating in the future.
   1. This will allow to track every ApplicationUser references, because a build error will be thrown.
   2. Adding *TODO:* declaration on the beginning of the comment allows us to track it with TaskList window, when switched into comments mode.



1. Open **Komsky.Domain** project
2. And add new class called ApplicationUserDomain into **Models/Identity** folder.
   1. Make this class public, inherit from IUser<string> generic interface. Implement missing members from interface.
3. Add new class named ApplicationRoleDomain into **Models/Identity** folder.
   1. Make this class public, inherit from IRole<string> generic interface. Implement missing members from interface.
4. Delete following references from **Komsky.Domain** project:
   1. EntityFramework
   2. EntityFramework.SqlServer
   3. Microsoft.AspNet.Identity.EntityFramework
   4. Every Komsky.Data reference
   5. Every Microsoft.Owin and Owin
5. Open **packages.config** file and manually delete following entries:
   1. EntityFramework
   2. Microsoft.AspNet.Identity.EntityFramework
   3. Every Microsoft.Owin and Owin
6. Build and debug your **Komsky.Domain** project (right click on project and build just this single project). Replace any instances of ApplicationUser to ApplicationDomainUser to fix errors.
7. If you have fixed all the errors and **Komsky.Domain** builds without any errors and warning, head to **Komsky.Services**
8. Again, build your project to see what errors we need to tackle. One of the major ones are Factories – they reference both data layer and domain layer.
9. Remove comment on ApplicationUser class in **Komsky.Data.Entities**, so it becomes available again.
10. Add two new factories for converting between ApplicationUser and ApplicationUserDomain. The factories should internally use following code:

public static class ApplicationUserDomainFactory

{

public static ApplicationUserDomain Create(ApplicationUser applicationUser)

{

return new ApplicationUserDomain

{

UserName = applicationUser.UserName,

Id = applicationUser.Id

};

}

public static ApplicationUserDomain CreateApplicationUserDomain(this ApplicationUser applicationUser)

{

return Create(applicationUser);

}

}

1. Update Ticket factories to use appropriate ApplicationUserDomain back and forth.
2. Build your **Komsky.Services** project and debug any errors. Project should build fine by now
3. Again, Data layer doesn’t need to reference EntityFramework. Remove every reference that has EntityFramework in its name.
4. Update packages.config by removing unnecessary EntityFramework packages.
5. Build again your project. Seems that we have broken something!

*And now comes the hardest part – ApplicationUserManager and ApplicationSignInManager classes! As our target is to remove dependencies on EntityFramework, we can’t rely on built-in classes like IdentityUser, UserStore etc. but rather on interfaces. And interfaces require implementing. If you feel that you are running out of time don’t hesitate to use finished solution, or better parts of it you don’t understand.*

1. Open **IdentityConfig.cs** file and replace every ApplicationUser with ApplicationUserDomain class. Use CTRL+F replace feature on current document. Exactly 17 occurrences should be found and replaced.
2. Add new folder to Komsky.Services called Identity, and move there IdentityConfig.cs
3. Add new class called UserStore and inherit from IUserStore<ApplicationUserDomain> generic interface. Implement missing members
4. We need to implement UserStore in similar manner as any handler, that is use DataFacade class to manage users.
   1. Add IDataFacade private field and initialize it the DI way
   2. Dispose \_dataFacade in Dispose method
5. Implement CreateAsync method by converting UserDomain into user while adding it into datafacade’s ApplicationUsers repository. Then return datafacade’s CommitAsync result.

public Task CreateAsync(ApplicationUserDomain user)

{

\_dataFacade.ApplicationUsers.Add(user.CreateApplicationUser());

return \_dataFacade.CommitAsync();

}

*Ouch! What is going on? Why createApplicationUser is underlined with red indicating error? Visual Studio wants us to add EntityFramework reference. Again, you have experienced Entourage pattern.*

1. Navigate to ApplicationUser definition and replace inheritance from IdentityUser with IUser<string>. Implement missing Properties and comment out GenerateIdentityAsync method.
2. Head back to UserStore definition and implement UpdateAsync and DeleteAsync the same way as CreateAsync.
3. FindByIdAsync method is more difficult, as we don’t have similar method on our repository. Navigate to IApplicationUserRepository and add two method signatures:
   1. Task<ApplicationUser> FindByIdAsync(string id);
   2. Task<ApplicationUser> FindByNameAsync(string name);
4. Implement those in ApplicationUserRepository class as follows:

public Task<ApplicationUser> FindByIdAsync(string id)

{

return Task.FromResult(GetById(id));

}

public Task<ApplicationUser> FindByNameAsync(string name)

{

return Task.FromResult(GetAll().First(x=>x.UserName == name));

}

1. As we are still in ApplicationUserRepository, seems that ApplicationUser is missing Email property. Navigate to definition and add public Email property of type String.
2. Head back to UserStore in **Services** project and finish implementing FindByIdAsync and FindByNameAsync. Use methods from repository you just created and convert to ApplicationUserDomain in ContinueWith method.

public Task<ApplicationUserDomain> FindByIdAsync(string userId)

{

return \_dataFacade.ApplicationUsers

.FindByIdAsync(userId)

.ContinueWith(x=>x.Result.CreateApplicationUserDomain());

}

public Task<ApplicationUserDomain> FindByNameAsync(string userName)

{

return \_dataFacade.ApplicationUsers

.FindByNameAsync(userName)

.ContinueWith(x => x.Result.CreateApplicationUserDomain());

}

1. Initialize ApplicationUserDomainManager instance with our new UserStore class. Also Initialize UserStore with DataFacade instance.

This is not perfect testable DI solution, but must do for now.

1. Add helper from ApplicationUser …
2. Replace inheritance on DbContext, remove throwIv
3. Replace ManageControllers ApplicationUserManager with ApplicationUserDomainManager
4. Replace AccountControllers ApplicationUserManager with ApplicationUserDomainManager
5. Add email to ApplicationUserDomain
6. Update factories
7. Update TicketViewModel
8. Startup.Auth.cs
9. Add PasswordHash, PhoneNumber to ApplicationUser and Domain
10. Update Tests
11. Sdf
12. Sdf
13. Sd
14. Fsd
15. Fsd
16. F
17. Sdf
18. Sdf
19. Sdf

[This lab is not finished. Consider inheriting from existing Identity object, instead of creating your own, and just map it – who cares, anyway?]