2011

Gestió de tasques informàtiques

PAC3 – Comentaris implementació

Treball fi de Carrera

Marçal Serrate Ollé
UOC
21/05/2011

Continguts

Disseny i Arquitectura

Per al disseny i implementació s'han seguit els conceptes de Domain-Driven Design i la implementació dels patrons d'aplicacions empresarials.

Com a referencia d'aplicació d'aquests conceptes tenim els següents recursos:

- **Domain-Driven Design: Tackling Complexity in the Heart of Software**.
  Autor: Eric Evans.
  Addison-Wesley 2004.

  Aquest llibre presenta els principis i bones pràctiques per a encarar dominis complexos mitjançant un llenguatge comú (Ubiquituous Language) entre experts del domini (negoci) i els desenvolupadors.

- **Patterns of Enterprise Application Architecture.**
  Autor: Martin Fowler.
  Addison-Wesley 2002.

  Aquest llibre presenta els patrons més importants en el desenvolupament de programari dirigit a aplicacions empresarials agrupats pels problemes que solucionen cada un d'ells.

Implementació

La solució de l'aplicació consta dels següents projectes i carpetes:

Carpeta	Projecte	Tipus	Descripció
0. Libs	n/a	*.dll	Tenim totes les llibreries de tercers utilitzades en el projecte.
1. Modeling	SerrateTFC.Modeling	Diagrames	Tenim alguns dels diagrames principals de l'aplicació.
2. Data	SerrateTFC.Database	Base de dades	Projecte per a mantenir sincronitzats els canvis d'esquemes de la base de dades.
	SerrateTFC.Infrastructure.Data	Llibreria de classes	Implementació dels repositoris d'accés a dades.
3. Domain	SerrateTFC.Domain	Llibreria de classes	Projecte on s'implementa el domini: entitats, servies de domini, etc.
	SerrateTFC.Domain.Data	Llibreria de classes	Interfícies de dades a utilitzar per la capa de domini.
4. Web	SerrateTFC.Localization	Llibreria de classes	Fitxers de recursos (resx) per a la localització de l'aplicació.
	SerrateTFC.Web	ASP.NET MVC 3	Capa web de l'aplicació.
5. WCF REST	SerrateTFC.RestService	WCF	Implementació dels serveis REST.
6. WP7	SerrateTFC.WindowsPhone	Silverlight per WP7	Capa mòbil de l'aplicació.

Agrupació Data

En aquesta agrupació tenim els projectes dedicats a la infraestructura de dades.

El projecte **SerrateTFC.Database** ens permet tenir controlats i sincronitzats tots els canvis d'esquema de la base de dades i d'aquesta manera poden crear posteriorment els scripts per crear-la de nou en qualsevol altre entorn. Per a més informació: http://msdn.microsoft.com/en-us/library/ff678491.aspx

El projecte **SerrateTFC.Infrastructure.Data** tenim els repositoris d'accés a dades en el que s'ha utilitzat Entity Framework 4.1.

En la classe MainUnitOfWork s'han creat las configuracions per a relacionar cada propietat de les entitats de domini amb les columnes de les taules de la base de dades.

El cas més rellevant és el de mapeig de la taula Users en què hem utilitzat **Table per Hierarchy** (TPH) a on habilitem el polimorfisme mitjançant la denormalització de la base de dades en una sola taula i mitjançant un discriminador sabem a quina classe en concret de la jerarquia de User hem de realitzar el mapeig:

```
            modelBuilder.Entity<User>()
                .Map<ProjectManager>(m =>
                {
                    m.Requires("UserTypeId").HasValue((int)Roles.ProjectManager);
                    m.MapInheritedProperties();
                })
                .Map<Administrator>(m =>
                {
                    m.Requires("UserTypeId").HasValue((int)Roles.Administrator);
                    m.MapInheritedProperties();
                })
                .Map<Resource>(m =>
                {
                    m.Requires("UserTypeId").HasValue((int)Roles.Resource);
                    m.Properties(p => new { HourlyRate = p.HourlyRate });
                    m.MapInheritedProperties();
                });
```

D'altra banda hem utilitzat el patró **Unit of Work** en la que manté un llistat de tots els canvis produïts en una entitat i coordina la persistència a la vegada de tots aquests canvis.

Per a més informació: http://msdn.microsoft.com/en-us/magazine/dd882510.aspx

Agrupació Domain

En aquesta agrupació tenim els projectes dedicats a la lògica de domini.

El projecte **SerrateTFC.Domain** tenim desenvolupats les principals artefactes del disseny orientat al domini.

- Entitats del domini: Les entitats del domini de l'aplicació actual.
- Serveis del domini: Les classes que utilitzarà les capes de presentació per interactuar amb les entitats.
- Value Objects: Són entitats que no tenen identitat, simplement representen el conjunt de tots els seus atributs.
- Repositories: Els contractes que han d'implementar els repositoris de la capa d'infraestructura.

El sentit de tenir separats els contractes de repositoris de la seva implementació en capes diferents es que obtenim el que es coneix com: *Persistence Ignorance* en què ens independitzem de la tecnologia emprada per a persistir les dades. Avui pot ser Entity Framework, demà NHibernate o alguna altra tecnologia.

En el projecte **SerrateTFC.Domain.Data** tenim les interfícies base que necessitarem en el domini per a poder interactuar amb la capa d'infraestructura de forma desacoplada.

Agrupació Web

En aquesta agrupació tenim els projectes relacionats amb la capa de presentació web.

En el projecte **SerrateTFC.Localization** tenim els fitxers de recursos que utilitzarà tota l'aplicació per a poder dotar-la de multi localització.

En el projecte **SerrateTFC.Web** tenim la implementació del projecte ASP.NET MVC 3 en la que hem utilitzat Razor com a view engine.

Inversion of Control

En aquest projecte tenim configurat la resolució de dependències de tota l'aplicació mitjançant **Inversion of Control (IoC):**

http://msdn.microsoft.com/en-us/library/ff921087.aspx

http://www.martinfowler.com/articles/injection.html

El que obtenim amb aquest patró és el total desacoblament de totes les capes de la nostra aplicació. En la nostra aplicació utilitzem la llibreria **Unity** com a framework de IoC què ens permet la resolució de dependències mitjançant dos patrons:

- **Dependency Injection:** Ens injecta la dependència en el constructor de la classe (el framework permet també injectar les dependències a propietats o mètodes), de la següent manera:

```
        private readonly IUnitOfWork _unitOfWork;
        private readonly ICustomerDomainService _customerDomainService;

        public CustomerController(
            IUnitOfWork unitOfWork,
            ICustomerDomainService customerDomainService)
        {
            this._unitOfWork = unitOfWork;
            this._customerDomainService = customerDomainService;
        }
```

- **Service Locator:** Obtenim les dependències sota demanda mitjançant l'ús d'un ServiceLocator:

```
        private readonly IProjectDomainService _projectDomainService;
        private readonly IUnitOfWork _unitOfWork;

        public MainService()
        {
            this._projectDomainService =
ServiceLocator.Current.GetInstance<IProjectDomainService>();
            this._unitOfWork = ServiceLocator.Current.GetInstance<IUnitOfWork>();
        }
```


En la classe **UnityDependencyResolver** registrem per a cada interfície quina implementació utilitzarem:

```
            this._container.RegisterType<ICustomerDomainService, CustomerDomainService>();
            this._container.RegisterType<ICustomerRepository, CustomerRepository>();
            this._container.RegisterType<IUserDomainService, UserDomainService>();
            this._container.RegisterType<IUserRepository, UserRepository>();
```

També s'ha hagut de crear un LifetimeManager personalitzat ja que per tenir la instància del UnitOfWork en tot el cicle de vida d'un request d'HTTP hem guardat la instància en el HttpContext d'ASP.NET: això s'ha realitzat en la classe **HttpContextLifetimeManager**.

Automapper

Com les entitats de domini no estan dissenyades per a la presentació a les pantalles sino pel domini de l'aplicació, s'utilitza tant per a la capa web com per a la capa de serveis, models de vista i edició especialitzats en el tractament de dades.

Per a la realització de la còpia dels valors de les propietats de les entitats de domini als objectes de visualització s'utilitza la llibreria AutoMapper que facilita el mapeig entre propietats.

Primerament realitzem la configuració del mapeig:

```
            Mapper.CreateMap<CustomerDetails, Customer>()
                .ForMember(dest => dest.Address, opt =>
                    opt.MapFrom(src => new Address(src.AddressPublicAddress, src.AddressCity,
src.AddressPostalCode, src.AddressCountry)));
```

Per a realitzar el mapeig simplement utilitzem la llibreria de la següent forma:

```
var customer = Mapper.Map<CustomerDetails, Customer>(details);
```


Models de presentació

Els models utilitzats ena la capa de presentació s'han configurat depenent de les necessitats de cada pantalla i s'han decorat amb els atributs necessaris: Validacions i les pertinents localitzacions:

```
    public class UserView
    {
        [ScaffoldColumn(false)]
        public Guid Id { get; set; }
        [Display(Name = "FirstName", ResourceType = typeof(Labels))]
        public string UserAccountFirstName { get; set; }
        [Display(Name = "LastName", ResourceType = typeof(Labels))]
        public string UserAccountLastName { get; set; }
        [Display(Name = "ResourceType", ResourceType = typeof(Labels))]
        public string ResourceType { get; set; }
    }
```

A l'atribut Display configurem el fitxer de recursos i la clau per a trobar la descripció per a la propietat.

```
    public class ProjectDetails
    {
        public Guid? Id { get; set; }
        [Required(ErrorMessageResourceName = "Validate_Required", ErrorMessageResourceType =
typeof(Labels))]
        [Display(Name = "Name", ResourceType = typeof(Labels))]
        public string Name { get; set; }
        [Display(Name = "Description", ResourceType = typeof(Labels))]
        [DataType(DataType.MultilineText)]
        public string Description { get; set; }
        [Required(ErrorMessageResourceName = "Validate_Required", ErrorMessageResourceType =
typeof(Labels))]
        [Display(Name = "Customer", ResourceType = typeof(Labels))]
        public Guid CustomerId { get; set; }
        [Required(ErrorMessageResourceName = "Validate_Required", ErrorMessageResourceType =
typeof(Labels))]
        [Display(Name = "ProjectManager", ResourceType = typeof(Labels))]
        public Guid ProjectManagerId { get; set; }
        [Required(ErrorMessageResourceName = "Validate_Required", ErrorMessageResourceType =
typeof(Labels))]
        [Display(Name = "StartDate", ResourceType = typeof(Labels))]
        public DateTime StartDate { get; set; }
        [Required(ErrorMessageResourceName = "Validate_Required", ErrorMessageResourceType =
typeof(Labels))]
        [Display(Name = "EndDate", ResourceType = typeof(Labels))]
        public DateTime EndDate { get; set; }
        [Display(Name = "IsRssEnabled", ResourceType = typeof(Labels))]
        public bool IsRssEnabled { get; set; }
        public IEnumerable<SelectListItem> ProjectManagers { get; set; }
        public IEnumerable<SelectListItem> Customers { get; set; }
    }
```

En els models d'edició especifiquem les validacions i el missatge localitzat a presentar quan no supera la validació i altres configuracions per a deixar les vistes més netes de codi.

Seguretat

Per a validar que l'usuari tingui el rol que li permeti accedir a una acció en particular s'ha desenvolupat un atribut personalitzat:

```
public class UserTypeAuthorizeAttribute : FilterAttribute, IAuthorizationFilter
    {
        private readonly Roles[] _roles;

        public UserTypeAuthorizeAttribute(params Roles[] roles)
        {
            _roles = roles;
        }

        public void OnAuthorization(AuthorizationContext filterContext)
        {
            IUserDomainService service =
ServiceLocator.Current.GetInstance<IUserDomainService>();
            var userName = HttpContext.Current.User.Identity.Name;
            User currentUser = service.GetUserByUsername(userName);

            bool isInRole = false;
            foreach (var item in _roles)
            {
                if (Compare(item, currentUser))
                {
                    isInRole = true;
                }
            }

            if (!isInRole)
            {
                throw new UnauthorizedAccessException();
            }
        }

        private bool Compare(Roles role, User user)
        {
            if (role == Roles.Administrator)
                return (user is Administrator);
            else if (role == Roles.ProjectManager)
                return (user is ProjectManager);
            else if (role == Roles.Resource)
                return (user is Resource);
            else
                return false;
        }
    }
```

Per a la seva utilització simplement situem l'atribut sobre el controlador o l'acció determinada:

```
    [UserTypeAuthorize(Roles.ProjectManager, Roles.Administrator)]
    public class ProjectController : BaseController
```

S'ha complementat la seguretat configurant el web.config de la manera següent:

```
    <authentication mode="Forms">
      <forms loginUrl="~/Account/LogOn" timeout="2880" />
    </authentication>

    <authorization>
      <deny users="?"/>
    </authorization>
```


Agrupació WCF

En el projecte **SerrateTFC.RestService** tenim definits els Data Transfer Objects (DTO) que són els objectes que es serialitzaran per enviar a través de la xarxa.

El contracte del servei REST definirem les operacions a realitzar i els formats dels missatges:

```
        [OperationContract]
        [WebGet(UriTemplate = "Tasks",
            BodyStyle = WebMessageBodyStyle.Bare,
            ResponseFormat = WebMessageFormat.Json)]
        List<TaskItem> GetTasks();
```

En aquest projecte també definirem un LifetimeManager propi per al framework **Unity** ja que en serveis WCF no tenim accés a HttpContext i per tant s'haurà d'utilitzar OperationContext. El recurs utilitzat per a la implementació d'aquest LifetimeManager s'ha extret del següent enllaç: http://blog.lowendahl.net/?p=243

Agrupació Windows Phone 7

En el projecte **SerrateTFC.WP7** tenim la capa de presentació mòbil mitjançant un projecte de Silverlight per a Windows Phone.

En aquest cas hem utilitzat el patró Model-View-ViewModel (MVVM) especialitzat per a aplicacions de Silverlight i WPF. Més informació de MVVM:

- http://msdn.microsoft.com/en-us/magazine/dd419663.aspx
- http://msdn.microsoft.com/en-us/magazine/dd458800.aspx

D'altra banda, per a consultar els serveis REST de WCF s'ha utilitzat la llibreria **RestSharp** que facilita la consulta de serveis REST per a WP7:

```
            var client = new RestClient(Globals.UrlService);
            var request = new RestRequest("Task/{id}", Method.GET);
            request.AddUrlSegment("id", id.Value.ToString());

            client.ExecuteAsync<TaskDetailsResponse>(request, (response) =>
            {
                Id = response.Data.Id;
                Name = response.Data.Name;
                Description = response.Data.Description;
                Priority = response.Data.Priority;
                Progress = response.Data.Progress;
                ProjectId = response.Data.ProjectId;
                AssignedId = response.Data.AssignedId;
                StartDate = DateTime.Parse(response.Data.StartDate);
                EndDate = DateTime.Parse(response.Data.EndDate);

                action();

            });
        }
```

Com l'execució de la consulta als serveis REST es realitza de forma asíncrona, s'han utilitzat controls de barra de progrés per a notificar la carga dels components de **Silverlight Toolkit for Windows Phone.**

Per a notificar a la vista quan s'havia completat una execució asíncrona s'ha utilitzat el delegat **Action** que permet executar un mètode donat:

```
        _viewModel.LogOn(
            (bool isLoggedIn) =>
            {
                if (isLoggedIn)
                {
                    if (LoggedOn != null)
                        LoggedOn(sender, e);
                }
                else
                {
                    lblError.Text = "Usuari/Contrasenya incorrectes";
                }
            });
```