Update-Package Newtonsoft.Json

1. В папла App\_Start -> WebApiConfig.cs се добавя:

config.Formatters.JsonFormatter.SupportedMediaTypes.Add(new MediaTypeHeaderValue("text/html"));

config.Formatters.JsonFormatter.SerializerSettings.ContractResolver =

new CamelCasePropertyNamesContractResolver();

Това се прави контролерите да връщат JSON string.

1. Създава се папка Repositories в Data модела
2. В нея има IRepository.cs интерфейс. Той описва основните дейности, които трябва да се имплементират :

public interface IRepository<T>

{

IQueryable<T> All();

T Find(object id);

void Add(T entity);

void Update(T entity);

void Remove(T entity);

T Remove(object id);

void SaveChanges();

}

1. Създава се клас GenericEfRepository.cs, той наследява и имплементира IRepository

public class GenericEfRepository<TEntity> : IRepository<TEntity> where TEntity : class

{

private DbContext dbContext;

private IDbSet<TEntity> entitySet;

public GenericEfRepository(DbContext dbContext)

{

this.dbContext = dbContext;

this.entitySet = dbContext.Set<TEntity>();

}

public IDbSet<TEntity> EntitySet

{

get { return this.entitySet; }

}

public System.Linq.IQueryable<TEntity> All()

{

return this.entitySet;

}

public TEntity Find(object id)

{

return this.entitySet.Find(id);

}

public void Add(TEntity entity)

{

this.ChangeState(entity, EntityState.Added);

}

public void Update(TEntity entity)

{

this.ChangeState(entity, EntityState.Modified);

}

public void Remove(TEntity entity)

{

this.ChangeState(entity, EntityState.Deleted);

}

public TEntity Remove(object id)

{

var entity = this.Find(id);

this.Remove(entity);

return entity;

}

public void SaveChanges()

{

this.dbContext.SaveChanges();

}

private void ChangeState(TEntity entity, EntityState state)

{

var entry = this.dbContext.Entry(entity);

if (entry.State == EntityState.Detached)

{

this.entitySet.Attach(entity);

}

entry.State = state;

}

}

1. Създава се папка UnitOfWork отново в Data слоя. В нея трябва да има един интерфейс и един клас. Интерфейсът IBugTrackerData.cs (името описва проекта) се описват колекции от моделите, който имаме, без енумерациите

public interface IBugTrackerData

{

IRepository<User> Users { get; }

IRepository<Bug> Bugs { get; }

IRepository<Comment> Comments { get; }

IUserStore<User> UserStore { get; }

void SaveChanges();

}

5.Създава се BugTrackerData.cs клас, който имплементира IBugTrackerData интерфейса. Той има 3 private полета, 2 конструктора, с и без параметри, няколко пропъртита и два метода. Конструктора без параметри вика този с параметър DBContext, като му предава контекст файла за проекта ( в този случай new BugTrackerDbContext() ).

public class BugTrackerData : IBugTrackerData

{

private readonly DbContext dbContext;

private readonly IDictionary<Type, object> repositories;

private IUserStore<User> userStore;

public BugTrackerData()

: this(new BugTrackerDbContext())

{

}

public BugTrackerData(DbContext dbContext)

{

this.dbContext = dbContext;

this.repositories = new Dictionary<Type, object>();

}

public IRepository<User> Users

{

get { return this.GetRepository<User>(); }

}

public IRepository<Bug> Bugs

{

get { return this.GetRepository<Bug>(); }

}

public IRepository<Comment> Comments

{

get { return this.GetRepository<Comment>(); }

}

public IUserStore<User> UserStore

{

get

{

if (this.userStore == null)

{

this.userStore = new UserStore<User>(this.dbContext);

}

return this.userStore;

}

}

public void SaveChanges()

{

this.dbContext.SaveChanges();

}

private IRepository<T> GetRepository<T>() where T : class

{

if (!this.repositories.ContainsKey(typeof(T)))

{

var type = typeof(GenericEfRepository<T>);

this.repositories.Add(typeof(T),

Activator.CreateInstance(type, this.dbContext));

}

return (IRepository<T>)this.repositories[typeof(T)];

}

}

1. В \*.Services проекта трябва да имаме папка „Infrastructure“, в която да има:

“AspNetUserProvider” и “IUserProvider”(възможно е да го има вече в папка „Providers“)

namespace News.Services.Infrastructure

{

public interface IUserProvider

{

bool IsAuthenticated { get; }

string GetUserId();

}

}

namespace News.Services.Infrastructure

{

using System.Threading;

using Microsoft.AspNet.Identity;

public class AspNetUserProvider : IUserProvider

{

public string GetUserId()

{

return Thread.CurrentPrincipal.Identity.GetUserId();

}

public bool IsAuthenticated {

get

{

return Thread.CurrentPrincipal.Identity.IsAuthenticated;

}

}

}

}

1. Създава се BasiApiControler : ApiControler

public class BaseApiController : ApiController

{

private readonly IBidSystemData db;

public BaseApiController()

: this(new BidSystemData(BidSystemDbContext.Create()), new AspNetUserProvider())

{

}

public BaseApiController(IBidSystemData data, IUserProvider userProvider)

{

this.db = data;

this.UserProvider = userProvider;

}

public IBidSystemData Db { get; set; }

protected IUserProvider UserProvider { get; set; }

protected ResponseMessageResult CreateResponseMessage(HttpStatusCode statusCode, string message)

{

return this.ResponseMessage(this.Request.CreateResponse(statusCode,

new

{

Message = message

}));

}

}

По този начин UserId няма да се взима от this.User.Identity.GetUserId а от:

this.UserProvider.GetUserId();

1. В контролерите се променя полето за инстанцията към базата и се правят два конструктора(един без и един с параметър). Първият вика вторият като му предава класа в UnitOfWork.

public class BugsController : BaseApiController

{

public BidsController()

{}

public BidsController(IBidSystemData data, IUserProvider userProvider)

: base(data, userProvider)

{ }

Това се прави за всеки контролер, който сме използвали и работи с базата. В контролерите където „светне“ се добавя .All() метода (от GenericEfRepository.cs). Там където имаме AddOrUpdate го заменяме с Update() метода.

1. Създава се папка „UnitTestsWithMoq“ в него се слага тестовия клас. Трябва да създадем MockContainer