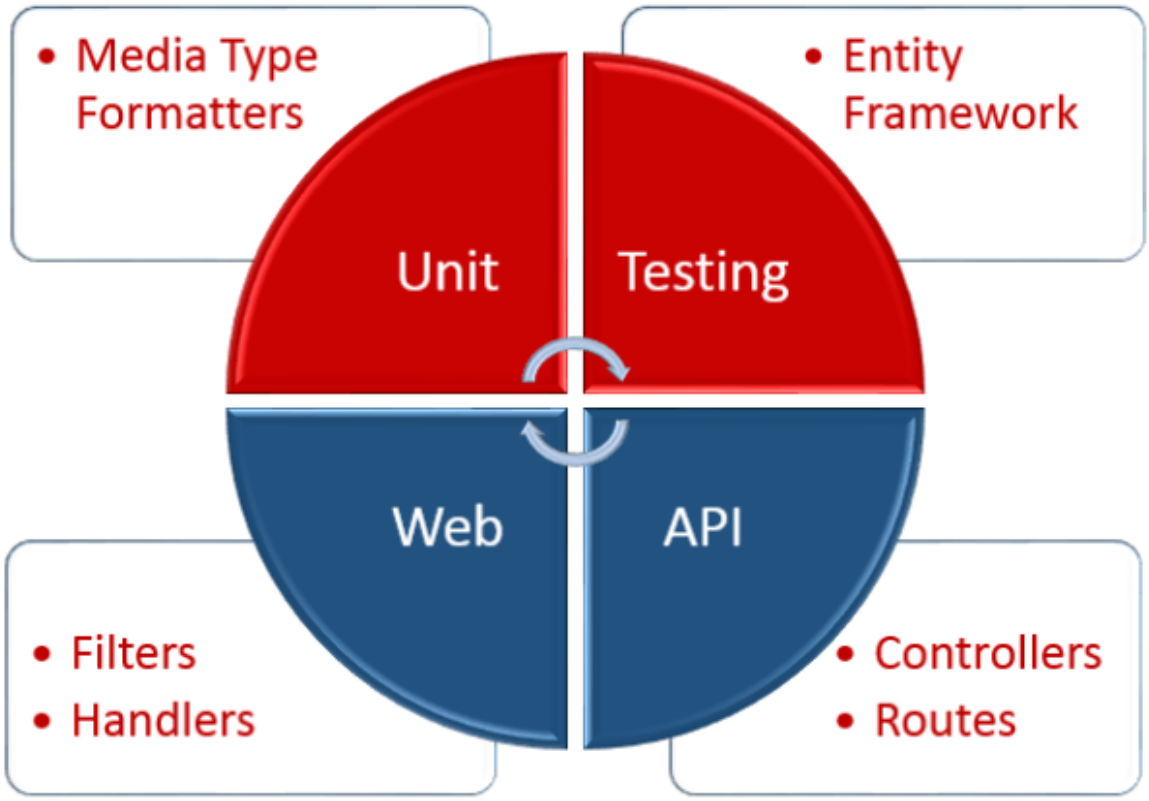
chsakell's Blog

ANYTHING AROUND ASP.NET MVC,WEB API, WCF, ENTITY FRAMEWORK & C#

ASP.NET Web API Unit Testing

BY CHRISTOS S. *on* MAY 10, 2015 · (8)

Unit testing can be beneficial to many aspects in software develepment, from the lowest level that is the source code to the highest level and the end user's experience. Writing automated tests helps finding defects earlier in the development lifecycle process which leads to fewer late nights or weekend work (happier developers). Since defects are resolved before production, less defects reach end users (happier clients). It also increases reliability of source code, since if the base code doesn't change all tests should always return the same results. Last but not least, anyone that decides to write unit tests is also forced to write testable code which leads to better software development practices.



Web API Unit Testing

ASP.NET Web API stack has many aspects that firstly must be well understood before writing unit tests against it and that's what makes it difficult. This post is a full stack Web API Unit testing tutorial which means will show you how to unit test all the layers and components exist in your Web API application. Let's see what we are gonna see on this post:

- **Web API Solution Best Practices**: Create a loosely coupled, scalable and testable Web API application
- **Entity Framework Unit testing**: Mocking generic repositories and testing the service layer
- **Web API Controllers testing**: Direct and integration testing
- **Web API Filters unit testing**: Direct and integration testing
- **Web API Message Handlers unit testing**: Direct and integration testing
- **Web API Media type Formatters unit testing**
- **Web API routing unit testing**

I will break the post in two main sections. The first one will be the one where we 're gonna structure the application and the second one will be the actual Unit testing. For the first one I will follow the **Generic repository pattern** which I have already describe in **this** post. If you feel

familiar with those concepts and you just want to read about how the unit testing is done, you can skip this step. Mind though that part of this section will be the **Controller registration** of a referenced library which has an important role in our Unit testing.

Section One: Structuring the Web API Application

Create a new blank solution named *UnitTestingWebAPI* and add the following projects:

1. **UnitTestingWebAPI.Domain**: Class library (Contains Entity Models)
2. **UnitTestingWebAPI.Data**: Class library (Contains Repositories)
3. **UnitTestingWebAPI.Services**: Class library (Contains Services)
4. **UnitTestingWebAPI.API.Core**: Class library (Contains Web API components such as Controllers, Filters, Message Handlers)
5. **UnitTestingWebAPI.API**: Empty ASP.NET Web Application (Web application to host Web API)
6. **UnitTestingWebAPI.Tests**: Class library (Contains the Unit Tests)

Switch to *UnitTestingWebAPI.Domain* and add the following classes:

Article.cs

```
public class Article
    {
        public int ID { get; set; }
        public string Title { get; set; }
        public string Contents { get; set; }
        public string Author { get; set; }
        public string URL { get; set; }
        public DateTime DateCreated { get; set; }
        public DateTime DateEdited { get; set; }

        public int BlogID { get; set; }
        public virtual Blog Blog { get; set; }

        public Article()
        {
        }
    }
```

Blog.cs

```
public class Blog
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public string URL { get; set; }
        public string Owner { get; set; }
        public DateTime DateCreated { get; set; }
        public virtual ICollection<Article> Articles { get; set; }

        public Blog()
```

```
        {
            Articles = new HashSet<Article>();
        }
    }
```

Repository Layer

Switch to *UnitTestingWebAPI.Data*, install Entity Framework from Nuget packages, add a reference to *UnitTestingWebAPI.Data* and add the following classes (create the respective folder if required):

Configurations/ArticleConfiguration.cs

```
public class ArticleConfiguration : EntityTypeConfiguration<Article>
    {
        public ArticleConfiguration()
        {
            ToTable("Article");
            Property(a => a.Title).IsRequired().HasMaxLength(100);
            Property(a => a.Contents).IsRequired();
            Property(a => a.Author).IsRequired().HasMaxLength(50);
            Property(a => a.URL).IsRequired().HasMaxLength(200);
            Property(a => a.DateCreated).HasColumnType("datetime2");
            Property(a => a.DateEdited).HasColumnType("datetime2");
        }
    }
```

Configurations/BlogConfiguration.cs

```
public class BlogConfiguration : EntityTypeConfiguration<Blog>
    {
        public BlogConfiguration()
        {
            ToTable("Blog");
            Property(b => b.Name).IsRequired().HasMaxLength(100);
            Property(b => b.URL).IsRequired().HasMaxLength(200);
            Property(b => b.Owner).IsRequired().HasMaxLength(50);
            Property(b => b.DateCreated).HasColumnType("datetime2");
        }
    }
```

```
public class BloggerEntities : DbContext
    {
        public BloggerEntities()
            : base("BloggerEntities")
        {
            Configuration.ProxyCreationEnabled = false;
        }

        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Article> Articles { get; set; }

        public virtual void Commit()
        {
            base.SaveChanges();
        }

```

```
        protected override void OnModelCreating(DbModelBuilder modelBuil
        {
            modelBuilder.Configurations.Add(new ArticleConfiguration());
            modelBuilder.Configurations.Add(new BlogConfiguration());
        }
    }
```

BloggerInitializer.cs

```
public class BloggerInitializer : DropCreateDatabaseIfModelChanges<Blogg
    {
        protected override void Seed(BloggerEntities context)
        {
            GetBlogs().ForEach(b => context.Blogs.Add(b));

            context.Commit();
        }

        public static List<Blog> GetBlogs()
        {
            List<Blog> _blogs = new List<Blog>();

            // Add two Blogs
            Blog _chsakellsBlog = new Blog()
            {
                Name = "chsakell's Blog",
                URL = "http://chsakell.com/",
                Owner = "Chris Sakellarios",
                Articles = GetChsakellsArticles()
            };

            Blog _dotNetCodeGeeks = new Blog()
            {
                Name = "DotNETCodeGeeks",
                URL = "dotnetcodegeeks",
                Owner = ".NET Code Geeks",
                Articles = GetDotNETGeeksArticles()
            };

            _blogs.Add(_chsakellsBlog);
            _blogs.Add(_dotNetCodeGeeks);

            return _blogs;
        }

        public static List<Article> GetChsakellsArticles()
        {
            List<Article> _articles = new List<Article>();

            Article _oData = new Article()
            {
                Author = "Chris S.",
                Title = "ASP.NET Web API feat. OData",
                URL = "http://chsakell.com/2015/04/04/asp-net-web-api-feat-
                Contents = @"OData is an open standard protocol allowing
                           and interoperable RESTful APIs. It was initi
                           .NET Developers from WCF Data Services. Ther
```

```
                                OData services such as Node.js, PHP, Java an
                                Web API also supports OData and this post wi
            };

            Article _wcfCustomSecurity= new Article()
            {
                Author = "Chris S.",
                Title = "Secure WCF Services with custom encrypted token
                URL = "http://chsakell.com/2014/12/13/secure-wcf-services-w
                Contents = @"Windows Communication Foundation framework
                            concerning the security logic you will apply
                            used for certain kind and levels of security
                            some types of security. There are some times
                            WCF security available options and hence, yo
                            accoarding to your business needs."
            };

            _articles.Add(_oData);
            _articles.Add(_wcfCustomSecurity);

            return _articles;
        }

        public static List<Article> GetDotNETGeeksArticles()
        {
            List<Article> _articles = new List<Article>();

            Article _angularFeatWebAPI = new Article()
            {
                Author = "Gordon Beeming",
                Title = "AngularJS feat. Web API",
                URL = "http://www.dotnetcodegeeks.com/2015/05/angularjs-feat
                Contents = @"Developing Web applications using AngularJS
                            this architecture in case you have in mind a
                            post backs to the server while each applicat
            };

            _articles.Add(_angularFeatWebAPI);

            return _articles;
        }

        public static List<Article> GetAllArticles()
        {
            List<Article> _articles = new List<Article>();
            _articles.AddRange(GetChsakellsArticles());
            _articles.AddRange(GetDotNETGeeksArticles());

            return _articles;
        }
    }
```

Infrastructure/Disposable.cs

```
public class Disposable : IDisposable
    {
        private bool isDisposed;
```

```
4
5           ~Disposable()
6           {
7               Dispose(false);
8           }
9
10          public void Dispose()
11          {
12              Dispose(true);
13              GC.SuppressFinalize(this);
14          }
15          private void Dispose(bool disposing)
16          {
17              if (!isDisposed && disposing)
18              {
19                  DisposeCore();
20              }
21
22              isDisposed = true;
23          }
24
25          // Ovveride this to dispose custom objects
26          protected virtual void DisposeCore()
27          {
28          }
29      }
```

Infrastructure/IDbFactory.cs

```
1   public interface IDbFactory : IDisposable
2       {
3           BloggerEntities Init();
4       }
```

Infrastructure/DbFactory.cs

```
1   public class DbFactory : Disposable, IDbFactory
2       {
3           BloggerEntities dbContext;
4
5           public BloggerEntities Init()
6           {
7               return dbContext ?? (dbContext = new BloggerEntities());
8           }
9
10          protected override void DisposeCore()
11          {
12              if (dbContext != null)
13                  dbContext.Dispose();
14          }
15      }
```

Infrastrure/IRepository.cs

```
1   public interface IRepository<T> where T : class
2       {
3           // Marks an entity as new
4           void Add(T entity);
5           // Marks an entity as modified
```

```
        void Update(T entity);
        // Marks an entity to be removed
        void Delete(T entity);
        void Delete(Expression<Func<T, bool>> where);
        // Get an entity by int id
        T GetById(int id);
        // Get an entity using delegate
        T Get(Expression<Func<T, bool>> where);
        // Gets all entities of type T
        IEnumerable<T> GetAll();
        // Gets entities using delegate
        IEnumerable<T> GetMany(Expression<Func<T, bool>> where);
    }
```

Infrastructure/RepositoryBase.cs

```
public abstract class RepositoryBase<T> where T : class
    {
        #region Properties
        private BloggerEntities dataContext;
        private readonly IDbSet<T> dbSet;

        protected IDbFactory DbFactory
        {
            get;
            private set;
        }

        protected BloggerEntities DbContext
        {
            get { return dataContext ?? (dataContext = DbFactory.Init())
        }
        #endregion

        protected RepositoryBase(IDbFactory dbFactory)
        {
            DbFactory = dbFactory;
            dbSet = DbContext.Set<T>();
        }

        #region Implementation
        public virtual void Add(T entity)
        {
            dbSet.Add(entity);
        }

        public virtual void Update(T entity)
        {
            dbSet.Attach(entity);
            dataContext.Entry(entity).State = EntityState.Modified;
        }

        public virtual void Delete(T entity)
        {
            dbSet.Remove(entity);
        }

        public virtual void Delete(Expression<Func<T, bool>> where)
```

```
        {
            IEnumerable<T> objects = dbSet.Where<T>(where).AsEnumerable(
            foreach (T obj in objects)
                dbSet.Remove(obj);
        }

        public virtual T GetById(int id)
        {
            return dbSet.Find(id);
        }

        public virtual IEnumerable<T> GetAll()
        {
            return dbSet.ToList();
        }

        public virtual IEnumerable<T> GetMany(Expression<Func<T, bool>>
        {
            return dbSet.Where(where).ToList();
        }

        public T Get(Expression<Func<T, bool>> where)
        {
            return dbSet.Where(where).FirstOrDefault<T>();
        }

        #endregion

    }
```

Infrastrure/IUnitOfWork.cs

```
public interface IUnitOfWork
    {
        void Commit();
    }
```

Infrastrure/UnitOfWork.cs

```
 public class UnitOfWork : IUnitOfWork
     {
         private readonly IDbFactory dbFactory;
         private BloggerEntities dbContext;

         public UnitOfWork(IDbFactory dbFactory)
         {
             this.dbFactory = dbFactory;
         }

         public BloggerEntities DbContext
         {
             get { return dbContext ?? (dbContext = dbFactory.Init()); }
         }

         public void Commit()
         {
             DbContext.Commit();
         }
```

```
20      }
```

Repositories/ArticleRepository.cs

```
1   public class ArticleRepository : RepositoryBase<Article>, IArticleReposi
2       {
3           public ArticleRepository(IDbFactory dbFactory)
4               : base(dbFactory) { }
5
6           public Article GetArticleByTitle(string articleTitle)
7           {
8               var _article = this.DbContext.Articles.Where(b => b.Title ==
9
10              return _article;
11          }
12      }
13
14      public interface IArticleRepository : IRepository<Article>
15      {
16          Article GetArticleByTitle(string articleTitle);
17      }
```

Infrastructure/BlogRepository.cs

```
1   public class BlogRepository : RepositoryBase<Blog>, IBlogRepository
2       {
3           public BlogRepository(IDbFactory dbFactory)
4               : base(dbFactory) { }
5
6           public Blog GetBlogByName(string blogName)
7           {
8               var _blog = this.DbContext.Blogs.Where(b => b.Name == blogNa
9
10              return _blog;
11          }
12      }
13
14      public interface IBlogRepository : IRepository<Blog>
15      {
16          Blog GetBlogByName(string blogName);
17      }
```

Service layer

Switch to *UnitTestingWebAPI.Service* project, add references to *UnitTestingWebAPI.Domain*,*UnitTestingWebAPI.Data* and add the following two files:

ArticleService.cs

```
1   // operations you want to expose
2       public interface IArticleService
3       {
4           IEnumerable<Article> GetArticles(string name = null);
5           Article GetArticle(int id);
```

```
6           Article GetArticle(string name);
7           void CreateArticle(Article article);
8           void UpdateArticle(Article article);
9           void DeleteArticle(Article article);
10          void SaveArticle();
11      }
12
13      public class ArticleService : IArticleService
14      {
15          private readonly IArticleRepository articlesRepository;
16          private readonly IUnitOfWork unitOfWork;
17
18          public ArticleService(IArticleRepository articlesRepository, IUn
19          {
20              this.articlesRepository = articlesRepository;
21              this.unitOfWork = unitOfWork;
22          }
23
24          #region IArticleService Members
25
26          public IEnumerable<Article> GetArticles(string title = null)
27          {
28              if (string.IsNullOrEmpty(title))
29                  return articlesRepository.GetAll();
30              else
31                  return articlesRepository.GetAll().Where(c => c.Title.To
32          }
33
34          public Article GetArticle(int id)
35          {
36              var article = articlesRepository.GetById(id);
37              return article;
38          }
39
40          public Article GetArticle(string title)
41          {
42              var article = articlesRepository.GetArticleByTitle(title);
43              return article;
44          }
45
46          public void CreateArticle(Article article)
47          {
48              articlesRepository.Add(article);
49          }
50
51          public void UpdateArticle(Article article)
52          {
53              articlesRepository.Update(article);
54          }
55
56          public void DeleteArticle(Article article)
57          {
58              articlesRepository.Delete(article);
59          }
60
61          public void SaveArticle()
62          {
63              unitOfWork.Commit();
```

```
64          }
65
66          #endregion
67      }
```

BlogService.cs

```
1   // operations you want to expose
2       public interface IBlogService
3       {
4           IEnumerable<Blog> GetBlogs(string name = null);
5           Blog GetBlog(int id);
6           Blog GetBlog(string name);
7           void CreateBlog(Blog blog);
8           void UpdateBlog(Blog blog);
9           void SaveBlog();
10          void DeleteBlog(Blog blog);
11      }
12
13      public class BlogService : IBlogService
14      {
15          private readonly IBlogRepository blogsRepository;
16          private readonly IUnitOfWork unitOfWork;
17
18          public BlogService(IBlogRepository blogsRepository, IUnitOfWork
19          {
20              this.blogsRepository = blogsRepository;
21              this.unitOfWork = unitOfWork;
22          }
23
24          #region IBlogService Members
25
26          public IEnumerable<Blog> GetBlogs(string name = null)
27          {
28              if (string.IsNullOrEmpty(name))
29                  return blogsRepository.GetAll();
30              else
31                  return blogsRepository.GetAll().Where(c => c.Name == nam
32          }
33
34          public Blog GetBlog(int id)
35          {
36              var blog = blogsRepository.GetById(id);
37              return blog;
38          }
39
40          public Blog GetBlog(string name)
41          {
42              var blog = blogsRepository.GetBlogByName(name);
43              return blog;
44          }
45
46          public void CreateBlog(Blog blog)
47          {
48              blogsRepository.Add(blog);
49          }
50
```

```
        public void UpdateBlog(Blog blog)
        {
            blogsRepository.Update(blog);
        }

        public void DeleteBlog(Blog blog)
        {
            blogsRepository.Delete(blog);
        }

        public void SaveBlog()
        {
            unitOfWork.Commit();
        }

        #endregion
    }
```

Web API Core Components

Switch to *UnitTestingWebAPI.API.Core* and add references to *UnitTestingWebAPI.API.Domain* and *UnitTestingWebAPI.API.Service* projects. Install the following packages from Nuget Packages:

1. Entity Framework
2. Microsoft.AspNet.WebApi.Core
3. Microsoft.AspNet.WebApi.Client

Add the following Web API Controllers to a **Controllers** folder:

Controllers/ArticlesController.cs

```
public class ArticlesController : ApiController
    {
        private IArticleService _articleService;

        public ArticlesController(IArticleService articleService)
        {
            _articleService = articleService;
        }

        // GET: api/Articles
        public IEnumerable<Article> GetArticles()
        {
            return _articleService.GetArticles();
        }

        // GET: api/Articles/5
        [ResponseType(typeof(Article))]
        public IHttpActionResult GetArticle(int id)
        {
            Article article = _articleService.GetArticle(id);
```

```
21              if (article == null)
22              {
23                  return NotFound();
24              }
25
26              return Ok(article);
27          }
28
29          // PUT: api/Articles/5
30          [ResponseType(typeof(void))]
31          public IHttpActionResult PutArticle(int id, Article article)
32          {
33              if (!ModelState.IsValid)
34              {
35                  return BadRequest(ModelState);
36              }
37
38              if (id != article.ID)
39              {
40                  return BadRequest();
41              }
42
43              _articleService.UpdateArticle(article);
44
45              try
46              {
47                  _articleService.SaveArticle();
48              }
49              catch (DbUpdateConcurrencyException)
50              {
51                  if (!ArticleExists(id))
52                  {
53                      return NotFound();
54                  }
55                  else
56                  {
57                      throw;
58                  }
59              }
60
61              return StatusCode(HttpStatusCode.NoContent);
62          }
63
64          // POST: api/Articles
65          [ResponseType(typeof(Article))]
66          public IHttpActionResult PostArticle(Article article)
67          {
68              if (!ModelState.IsValid)
69              {
70                  return BadRequest(ModelState);
71              }
72
73              _articleService.CreateArticle(article);
74
75              return CreatedAtRoute("DefaultApi", new { id = article.ID },
76          }
77
78          // DELETE: api/Articles/5
```

```
        [ResponseType(typeof(Article))]
        public IHttpActionResult DeleteArticle(int id)
        {
            Article article = _articleService.GetArticle(id);
            if (article == null)
            {
                return NotFound();
            }

            _articleService.DeleteArticle(article);

            return Ok(article);
        }

        private bool ArticleExists(int id)
        {
            return _articleService.GetArticle(id) != null;
        }
    }
```

Controllers/BlogsController.cs

```
public class BlogsController : ApiController
    {
        private IBlogService _blogService;

        public BlogsController(IBlogService blogService)
        {
            _blogService = blogService;
        }

        // GET: api/Blogs
        public IEnumerable<Blog> GetBlogs()
        {
            return _blogService.GetBlogs();
        }

        // GET: api/Blogs/5
        [ResponseType(typeof(Blog))]
        public IHttpActionResult GetBlog(int id)
        {
            Blog blog = _blogService.GetBlog(id);
            if (blog == null)
            {
                return NotFound();
            }

            return Ok(blog);
        }

        // PUT: api/Blogs/5
        [ResponseType(typeof(void))]
        public IHttpActionResult PutBlog(int id, Blog blog)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
```

```
36              }
37
38              if (id != blog.ID)
39              {
40                  return BadRequest();
41              }
42
43              _blogService.UpdateBlog(blog);
44
45              try
46              {
47                  _blogService.SaveBlog();
48              }
49              catch (DbUpdateConcurrencyException)
50              {
51                  if (!BlogExists(id))
52                  {
53                      return NotFound();
54                  }
55                  else
56                  {
57                      throw;
58                  }
59              }
60
61              return StatusCode(HttpStatusCode.NoContent);
62          }
63
64          // POST: api/Blogs
65          [ResponseType(typeof(Blog))]
66          public IHttpActionResult PostBlog(Blog blog)
67          {
68              if (!ModelState.IsValid)
69              {
70                  return BadRequest(ModelState);
71              }
72
73              _blogService.CreateBlog(blog);
74
75              return CreatedAtRoute("DefaultApi", new { id = blog.ID }, bl
76          }
77
78          // DELETE: api/Blogs/5
79          [ResponseType(typeof(Blog))]
80          public IHttpActionResult DeleteBlog(int id)
81          {
82              Blog blog = _blogService.GetBlog(id);
83              if (blog == null)
84              {
85                  return NotFound();
86              }
87
88              _blogService.DeleteBlog(blog);
89
90              return Ok(blog);
91          }
92
93          private bool BlogExists(int id)
```

```
        {
            return _blogService.GetBlog(id) != null;
        }
    }
```

Add the following filter which when applied, it reverses the order of a List of articles:

Filters/ArticlesReversedFilter.cs

```
public class ArticlesReversedFilter : ActionFilterAttribute
    {
        public override void OnActionExecuted(HttpActionExecutedContext
        {
            var objectContent = actionExecutedContext.Response.Content a
            if (objectContent != null)
            {
                List<Article> _articles = objectContent.Value as List<Ar
                if (_articles != null && _articles.Count > 0)
                {
                    _articles.Reverse();
                }
            }
        }
    }
```

Add the following **MediaTypeFormatter** which can return a comma serated representation of articles:

MediaTypeFormatters/ArticleFormatter.cs

```
public class ArticleFormatter : BufferedMediaTypeFormatter
    {
        public ArticleFormatter()
        {
            SupportedMediaTypes.Add(new MediaTypeHeaderValue("applicatio
        }

        public override bool CanReadType(Type type)
        {
            return false;
        }

        public override bool CanWriteType(Type type)
        {
            //for single article object
            if (type == typeof(Article))
                return true;
            else
            {
                // for multiple article objects
                Type _type = typeof(IEnumerable<Article>);
                return _type.IsAssignableFrom(type);
            }
        }

```

```
        public override void WriteToStream(Type type,
                                           object value,
                                           Stream writeStream,
                                           HttpContent content)
        {
            using (StreamWriter writer = new StreamWriter(writeStream))
            {
                var articles = value as IEnumerable<Article>;
                if (articles != null)
                {
                    foreach (var article in articles)
                    {
                        writer.Write(String.Format("[{0},\"{1}\",\"{2}\"
                                                   article.ID,
                                                   article.Title,
                                                   article.Author,
                                                   article.URL,
                                                   article.Contents));
                    }
                }
                else
                {
                    var _article = value as Article;
                    if (_article == null)
                    {
                        throw new InvalidOperationException("Cannot seri
                    }
                    writer.Write(String.Format("[{0},\"{1}\",\"{2}\",\"{
                                                   _article.ID,
                                                   _article.Title,
                                                   _article.Author,
                                                   _article.URL,
                                                   _article.Contents));
                }
            }
        }
    }
```

Add the following two *Message Handlers*. The first one is responsible to add a custom header in the response and the second one is able to terminate the request if applied:

MessageHandlers/HeaderAppenderHandler.cs

```
public class HeaderAppenderHandler : DelegatingHandler
    {
        async protected override Task<HttpResponseMessage> SendAsync(
                HttpRequestMessage request, CancellationToken cancellati
        {
            HttpResponseMessage response = await base.SendAsync(request,

            response.Headers.Add("X-WebAPI-Header", "Web API Unit testin
            return response;
        }
    }
```

HeaderAppenderHandler/EndRequestHandler.cs

```
public class EndRequestHandler : DelegatingHandler
    {
        async protected override Task<HttpResponseMessage> SendAsync(
            HttpRequestMessage request, CancellationToken cancellationTo
        {
            if (request.RequestUri.AbsoluteUri.Contains("test"))
            {
                var response = new HttpResponseMessage(HttpStatusCode.OK
                {
                    Content = new StringContent("Unit testing message ha
                };

                var tsc = new TaskCompletionSource<HttpResponseMessage>(
                tsc.SetResult(response);
                return await tsc.Task;
            }
            else
            {
                return await base.SendAsync(request, cancellationToken);
            }
        }
    }
```

Add the following **DefaultAssembliesResolver** which will be used for Controller registration from the Web Application project:

CustomAssembliesResolver.cs

```
public class CustomAssembliesResolver : DefaultAssembliesResolver
    {
        public override ICollection<Assembly> GetAssemblies()
        {
            var baseAssemblies = base.GetAssemblies().ToList();
            var assemblies = new List<Assembly>(baseAssemblies) { typeof
            baseAssemblies.AddRange(assemblies);

            return baseAssemblies.Distinct().ToList();
        }
    }
```

ASP.NET Web Application

Switch to *UnitTestingWebAPI.API* web application project and add references to *UnitTestingWebAPI.Core*,*UnitTestingWebAPI.Data* and *UnitTestingWebAPI.Service*. You will also need to install the following Nuget packages:

1. Entity Framework
2. Microsoft.AspNet.WebApi.WebHost
3. Microsoft.AspNet.WebApi.Core

4. Microsoft.AspNet.WebApi.Client
5. Microsoft.AspNet.WebApi.Owin
6. Microsoft.Owin.Host.SystemWeb
7. Microsoft.Owin
8. Autofac.WebApi2

Add a **Global Configuration file** if not exists and set the database initializer:

Global.asax

```
public class Global : System.Web.HttpApplication
    {

        protected void Application_Start(object sender, EventArgs e)
        {
            // Init database
            System.Data.Entity.Database.SetInitializer(new BloggerInitial
        }
    }
```

Also make sure you add a relevant connection string in the *Web.config* file:

```
<connectionStrings>
  <add name="BloggerEntities" connectionString="Data Source=(localdb)\v11
</connectionStrings>
```

External Controller Registration

Create an Owin Startup.cs file at the root of the Web application and paste the following code. This code will ensure to use WebApi controllers from the *UnitTestingWebAPI.API.Core* project (CustomAssembliesResolver) and inject the appropriate repositories and services when required (autofac configuration):

Startup.cs

```
public class Startup
    {
        public void Configuration(IAppBuilder appBuilder)
        {
            var config = new HttpConfiguration();
            config.Services.Replace(typeof(IAssembliesResolver), new Cus
            config.Formatters.Add(new ArticleFormatter());

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );

```

```
15
16              // Autofac configuration
17              var builder = new ContainerBuilder();
18              builder.RegisterApiControllers(typeof(BlogsController).Assem
19              builder.RegisterType<UnitOfWork>().As<IUnitOfWork>().Instanc
20              builder.RegisterType<DbFactory>().As<IDbFactory>().InstanceP
21
22              //Repositories
23              builder.RegisterAssemblyTypes(typeof(BlogRepository).Assembl
24                  .Where(t => t.Name.EndsWith("Repository"))
25                  .AsImplementedInterfaces().InstancePerRequest();
26              // Services
27              builder.RegisterAssemblyTypes(typeof(ArticleService).Assembl
28                 .Where(t => t.Name.EndsWith("Service"))
29                 .AsImplementedInterfaces().InstancePerRequest();
30
31              IContainer container = builder.Build();
32              config.DependencyResolver = new AutofacWebApiDependencyResol
33
34              appBuilder.UseWebApi(config);
35          }
36      }
```

At this point you should be able to fire the Web application and request articles or blogs using the following requests (port may be different in yours):

```
1  http://localhost:56032/api/articles
```

```
1  http://localhost:56032/api/blogs
```

Section Two: Unit Testing

We have completed structuring our application and it's time to unit test all of our components. Switch to *UnitTestingWebAPI.Tests* class library and add references to *UnitTestingWebAPI.Domain*, *UnitTestingWebAPI.Data*,*UnitTestingWebAPI.Service* and *UnitTestingWebAPI.API.Core*. Also make sure you install the following Nuget Packages:

1. Entity Framework
2. Microsoft.AspNet.WebApi.Core
3. Microsoft.AspNet.WebApi.Client
4. Microsoft.AspNet.WebApi.Owin
5. Microsoft.AspNet.WebApi.SelfHost
6. Micoroft.Owin
7. Owin
8. Micoroft.Owin.Hosting
9. Micoroft.Owin.Host.HttpListener
10. Autofac.WebApi2

11. NUnit
12. NUnitTestAdapter

As you see we are going to use **NUnit** to write our unit tests.

Services Unit Testing

When writing Unit tests, first you need to setup or initiate some variables to be used for the unit tests. With *NUnit* this is done via a function with an attribute `Setup` applied on it. This very function will run before any NUnit test is executed. Unit testing the Service layer is the first thing you need to do since all the Controller's constructors are injected with Services. Hence, you need to emulate repositories and service behavior before starting unit testing WebAPI Core components. In this example we 'll see how to emulate the *ArticleSevice*. This service's constructor is injected with instances of **IArticleRepository** and **IUnitOfWork** so all we have to do is create two "special" instances and inject them.

ArticleService Constructor

```
public ArticleService(IArticleRepository articlesRepository, IUnitOfWork
        {
            this.articlesRepository = articlesRepository;
            this.unitOfWork = unitOfWork;
        }
```

I said "special" cause those instance are not going to be real instances that actually can access the database.

Attention

Unit tests must run in memory and shouldn't access databases. All core functionality must be emulated by using frameworks such as **Mock** in our case. This way automated tests will be much more faster. The basic purpose of unit tests is more **testing component behavior** rather than testing real results.

Let's procceed with testing the ArticleService. Create a file named *ServiceTests* and for start add the following code:

ServicesTests.cs

```
    [TestFixture]
    public class ServicesTests
    {
        #region Variables
        IArticleService _articleService;
        IArticleRepository _articleRepository;
        IUnitOfWork _unitOfWork;
        List<Article> _randomArticles;
        #endregion
```

```
10
11          #region Setup
12          [SetUp]
13          public void Setup()
14          {
15              _randomArticles = SetupArticles();
16
17              _articleRepository = SetupArticleRepository();
18              _unitOfWork = new Mock<IUnitOfWork>().Object;
19              _articleService = new ArticleService(_articleRepository, _un
20          }
21
22          public List<Article> SetupArticles()
23          {
24              int _counter = new int();
25              List<Article> _articles = BloggerInitializer.GetAllArticles(
26
27              foreach (Article _article in _articles)
28                  _article.ID = ++_counter;
29
30              return _articles;
31          }
32
33          public IArticleRepository SetupArticleRepository()
34          {
35              // Init repository
36              var repo = new Mock<IArticleRepository>();
37
38              // Setup mocking behavior
39              repo.Setup(r => r.GetAll()                    (_randomArticles);
40
41              repo.Setu                                     int>()))
42                  .Retu                                     (
43                      i                                     a => a.ID.Equals(id))));
44
45              repo.Setu                                     cle>()))
46                  .Call                                     ewArticle =>
47                  {
48                      d                                     domArticles.Last().ID;
49                      d                                     rticleID + 1;
50                      n                                     ID;
51                      n                                     teTime.Now;
52                      _                                     cle);
53                  }));
54
55              repo.Setu                                     rticle>()))
56                  .Call                                      =>
57                      {
58                          var oldArticle = _randomArticles.Find(a => a.ID
59                          oldArticle.DateEdited = DateTime.Now;
60                          oldArticle = x;
61                      }));
62
63              repo.Setup(r => r.Delete(It.IsAny<Article>()))
64                  .Callback(new Action<Article>(x =>
65                  {
66                      var _articleToRemove = _randomArticles.Find(a => a.I
67
```

```
                    if (_articleToRemove != null)
                        _randomArticles.Remove(_articleToRemove);
                }));

            // Return mock implementation
            return repo.Object;
        }

        #endregion
    }
}
```

In the *SetupArticleRepository()* function we emulate our _articleRepository behavior, in other words we setup what results are expected from this repository instance when a specific function is called. Then we inject this instance in our _articleService's constructor and we are ready to go. Let's say that we want to test that the `_articleService.GetArticles()` behaves as expected. Add the following NUnit test in the same file:

ServiceShouldReturnAllArticles Test

```
[Test]
public void ServiceShouldReturnAllArticles()
{
    var articles = _articleService.GetArticles();

    Assert.That(articles, Is.EqualTo(_randomArticles));
}
```

Build the Tests project, run the test and make sure it passes. In the same way create the following tests:

Services Tests

```
[Test]
public void ServiceShouldReturnRightArticle()
{
    var wcfSecurityArticle = _articleService.GetArticle(2);

    Assert.That(wcfSecurityArticle,
        Is.EqualTo(_randomArticles.Find(a => a.Title.Contains("Secure WC
}

[Test]
public void ServiceShouldAddNewArticle()
{
    var _newArticle = new Article()
    {
        Author = "Chris Sakellarios",
        Contents = "If you are an ASP.NET MVC developer, you will certai
        Title = "URL Rooting in ASP.NET (Web Forms)",
        URL = "http://chsakell.com/2013/12/15/url-rooting-in-asp-net-web-for
    };

    int _maxArticleIDBeforeAdd = _randomArticles.Max(a => a.ID);
    _articleService.CreateArticle(_newArticle);
```

```
23
24      Assert.That(_newArticle, Is.EqualTo(_randomArticles.Last()));
25      Assert.That(_maxArticleIDBeforeAdd + 1, Is.EqualTo(_randomArticles.L
26  }
27
28  [Test]
29  public void ServiceShouldUpdateArticle()
30  {
31      var _firstArticle = _randomArticles.First();
32
33      _firstArticle.Title = "OData feat. ASP.NET Web API"; // reversed <sp
34      _firstArticle.URL = "http://t.co/fuIbNoc7Zh"; // short link
35      _articleService.UpdateArticle(_firstArticle);
36
37      Assert.That(_firstArticle.DateEdited, Is.Not.EqualTo(DateTime.MinVal
38      Assert.That(_firstArticle.URL, Is.EqualTo("http://t.co/fuIbNoc7Zh"));
39      Assert.That(_firstArticle.ID, Is.EqualTo(1)); // hasn't changed
40  }
41
42  [Test]
43  public void ServiceShouldDeleteArticle()
44  {
45      int maxID = _randomArticles.Max(a => a.ID); // Before removal
46      var _lastArticle = _randomArticles.Last();
47
48      // Remove last article
49      _articleService.DeleteArticle(_lastArticle);
50
51      Assert.That(maxID, Is.GreaterThan(_randomArticles.Max(a => a.ID)));
52  }
```

Web API Controllers Unit Testing

Now that we are familiar with emulating our services behavior we can procceed with unit testing Web API Controllers. First thing we need to do is Setup the variables to be used through our test, so create a *ControllerTests.cs* file and paste the following code:

```
1       [TestFixture]
2       public class ControllerTests
3       {
4           #region Variables
5           IArticleService _articleService;
6           IArticleRepository _articleRepository;
7           IUnitOfWork _unitOfWork;
8           List<Article> _randomArticles;
9           #endregion
10
11          #region Setup
12          [SetUp]
13          public void Setup()
14          {
15              _randomArticles = SetupArticles();
16
17              _articleRepository = SetupArticleRepository();
18              _unitOfWork = new Mock<IUnitOfWork>().Object;
19              _articleService = new ArticleService(_articleRepository, _un
```

```
20          }
21
22          public List<Article> SetupArticles()
23          {
24              int _counter = new int();
25              List<Article> _articles = BloggerInitializer.GetAllArticles(
26
27              foreach (Article _article in _articles)
28                  _article.ID = ++_counter;
29
30              return _articles;
31          }
32
33          public IArticleRepository SetupArticleRepository()
34          {
35              // Init repository
36              var repo = new Mock<IArticleRepository>();
37
38              // Setup mocking behavior
39              repo.Setup(r => r.GetAll()).Returns(_randomArticles);
40
41              repo.Setup(r => r.GetById(It.IsAny<int>()))
42                  .Returns(new Func<int, Article>(
43                      id => _randomArticles.Find(a => a.ID.Equals(id))));
44
45              repo.Setup(r => r.Add(It.IsAny<Article>()))
46                  .Callback(new Action<Article>(newArticle =>
47                  {
48                      dynamic maxArticleID = _randomArticles.Last().ID;
49                      dynamic nextArticleID = maxArticleID + 1;
50                      newArticle.ID = nextArticleID;
51                      newArticle.DateCreated = DateTime.Now;
52                      _randomArticles.Add(newArticle);
53                  }));
54
55              repo.Setup(r => r.Update(It.IsAny<Article>()))
56                  .Callback(new Action<Article>(x =>
57                  {
58                      var oldArticle = _randomArticles.Find(a => a.ID == x
59                      oldArticle.DateEdited = DateTime.Now;
60                      oldArticle.URL = x.URL;
61                      oldArticle.Title = x.Title;
62                      oldArticle.Contents = x.Contents;
63                      oldArticle.BlogID = x.BlogID;
64                  }));
65
66              repo.Setup(r => r.Delete(It.IsAny<Article>()))
67                  .Callback(new Action<Article>(x =>
68                  {
69                      var _articleToRemove = _randomArticles.Find(a => a.I
70
71                      if (_articleToRemove != null)
72                          _randomArticles.Remove(_articleToRemove);
73                  }));
74
75              // Return mock implementation
76              return repo.Object;
77          }
```

```
78
79          #endregion
80      }
81  }
```

WebAPI Controller classes are classes just like all others so we can test them respectively. Let's see if the **_articlesController.GetArticles()** does return all articles available:

ControlerShouldReturnAllArticles Unit test

```
1  [Test]
2  public void ControlerShouldReturnAllArticles()
3  {
4      var _articlesController = new ArticlesController(_articleService);
5
6      var result = _articlesController.GetArticles();
7
8      CollectionAssert.AreEqual(result, _randomArticles);
9  }
```

The most important line here is the highlighted one where the _articleService instance injection will ensure the service's behavior.
In the same way we ensure that the last article is returned when invoking **_articlesController.GetArticle(3)** since we setup only 3 articles.

ControlerShouldReturnLastArticle Unit test

```
1  [Test]
2  public void ControlerShouldReturnLastArticle()
3  {
4      var _articlesController = new ArticlesController(_articleService);
5
6      var result = _articlesController.GetArticle(3) as OkNegotiatedConten
7
8      Assert.IsNotNull(result);
9      Assert.AreEqual(result.Content.Title, _randomArticles.Last().Title);
10 }
```

Let's test that an invalid Update operation must fail and return a *BadRequestResult*. Recall the Update operation setup on the _articleRepository:

```
1  repo.Setup(r => r.Update(It.IsAny<Article>()))
2                  .Callback(new Action<Article>(x =>
3                  {
4                      var oldArticle = _randomArticles.Find(a => a.ID == x
5                      oldArticle.DateEdited = DateTime.Now;
6                      oldArticle.URL = x.URL;
7                      oldArticle.Title = x.Title;
8                      oldArticle.Contents = x.Contents;
9                      oldArticle.BlogID = x.BlogID;
10                 }));
```

So If we pass an non existing article this update should fail:

ControlerShouldPutReturnBadRequestResult Unit test

```
[Test]
public void ControlerShouldPutReturnBadRequestResult()
{
    var _articlesController = new ArticlesController(_articleService)
    {
        Configuration = new HttpConfiguration(),
        Request = new HttpRequestMessage
        {
            Method = HttpMethod.Put,
            RequestUri = new Uri("http://localhost/api/articles/-1")
        }
    };

    var badresult = _articlesController.PutArticle(-1, new Article() { T
    Assert.That(badresult, Is.TypeOf<BadRequestResult>());
}
```

Complete the Controller Unit testing by adding the following three tests which tests that updating first article succeeds, post new article succeeds and post new article fails respectivelly.

Controller Unit tests

```
[Test]
public void ControlerShouldPutUpdateFirstArticle()
{
    var _articlesController = new ArticlesController(_articleService)
    {
        Configuration = new HttpConfiguration(),
        Request = new HttpRequestMessage
        {
            Method = HttpMethod.Put,
            RequestUri = new Uri("http://localhost/api/articles/1")
        }
    };

    IHttpActionResult updateResult = _articlesController.PutArticle(1, n
    {
        ID = 1,
        Title = "ASP.NET Web API feat. OData",
        URL = "http://t.co/fuIbNoc7Zh",
        Contents = @"OData is an open standard protocol.."
    }) as IHttpActionResult;

    Assert.That(updateResult, Is.TypeOf<StatusCodeResult>());

    StatusCodeResult statusCodeResult = updateResult as StatusCodeResult

    Assert.That(statusCodeResult.StatusCode, Is.EqualTo(HttpStatusCode.N

    Assert.That(_randomArticles.First().URL, Is.EqualTo("http://t.co/fuIb
}
```

```
31  [Test]
32  public void ControlerShouldPostNewArticle()
33  {
34      var article = new Article
35      {
36          Title = "Web API Unit Testing",
37          URL = "http://chsakell.com/web-api-unit-testing",
38          Author = "Chris Sakellarios",
39          DateCreated = DateTime.Now,
40          Contents = "Unit testing Web API.."
41      };
42
43      var _articlesController = new ArticlesController(_articleService)
44      {
45          Configuration = new HttpConfiguration(),
46          Request = new HttpRequestMessage
47          {
48              Method = HttpMethod.Post,
49              RequestUri = new Uri("http://localhost/api/articles")
50          }
51      };
52
53      _articlesController.Configuration.MapHttpAttributeRoutes();
54      _articlesController.Configuration.EnsureInitialized();
55      _articlesController.RequestContext.RouteData = new HttpRouteData(
56      new HttpRoute(), new HttpRouteValueDictionary { { "_articlesControll
57      var result = _articlesController.PostArticle(article) as CreatedAtRo
58
59      Assert.That(result.RouteName, Is.EqualTo("DefaultApi"));
60      Assert.That(result.Content.ID, Is.EqualTo(result.RouteValues["id"]))
61      Assert.That(result.Content.ID, Is.EqualTo(_randomArticles.Max(a => a
62  }
63
64  [Test]
65  public void ControlerShouldNotPostNewArticle()
66  {
67      var article = new Article
68      {
69          Title = "Web API Unit Testing",
70          URL = "http://chsakell.com/web-api-unit-testing",
71          Author = "Chris Sakellarios",
72          DateCreated = DateTime.Now,
73          Contents = null
74      };
75
76      var _articlesController = new ArticlesController(_articleService)
77      {
78          Configuration = new HttpConfiguration(),
79          Request = new HttpRequestMessage
80          {
81              Method = HttpMethod.Post,
82              RequestUri = new Uri("http://localhost/api/articles")
83          }
84      };
85
86      _articlesController.Configuration.MapHttpAttributeRoutes();
87      _articlesController.Configuration.EnsureInitialized();
88      _articlesController.RequestContext.RouteData = new HttpRouteData(
```

```
89          new HttpRoute(), new HttpRouteValueDictionary { { "Controller", "Art
90          _articlesController.ModelState.AddModelError("Contents", "Contents i
91
92          var result = _articlesController.PostArticle(article) as InvalidMode
93
94          Assert.That(result.ModelState.Count, Is.EqualTo(1));
95          Assert.That(result.ModelState.IsValid, Is.EqualTo(false));
96      }
```

Take a good look the highlighted lines and see that we can unit test several aspects of our requests, such as *CodeStatus* returned or routing properties.

Message Handlers Unit Testing

You can test Message Handlers by creating an instance of **HttpMessageInvoker**, passing the Message Handler instance you want to test and invoke the *SendAsync* function. Create a MessageHandlerTests.cs file and paste the Setup code first:

MessageHandlerTests.cs

```
1       [TestFixture]
2       public class MessageHandlerTests
3       {
4           #region Variables
5           private EndRequestHandler _endRequestHandler;
6           private HeaderAppenderHandler _headerAppenderHandler;
7           #endregion
8
9           #region Setup
10          [SetUp]
11          public void Setup()
12          {
13              // Direct MessageHandler test
14              _endRequestHandler = new EndRequestHandler();
15              _headerAppenderHandler = new HeaderAppenderHandler()
16              {
17                  InnerHandler = _endRequestHandler
18              };
19          }
20          #endregion
21      }
22  }
```

We setup the *HeaderAppenderHandler's* inner handler another Handler that will terminate the request. Recall that the EndRequestHandler will end the request only if the Uri contains a *test* literal. Let's write the test now:

Direct ShouldAppendCustomHeader Unit Test

```
1   [Test]
2   public async void ShouldAppendCustomHeader()
3   {
4       var invoker = new HttpMessageInvoker(_headerAppenderHandler);
5       var result = await invoker.SendAsync(new HttpRequestMessage(HttpMeth
```

```
        new Uri("http://localhost/api/test/")), CancellationToken.None);

    Assert.That(result.Headers.Contains("X-WebAPI-Header"), Is.True);
    Assert.That(result.Content.ReadAsStringAsync().Result,
        Is.EqualTo("Unit testing message handlers!"));
}
```

Now let's pick up tha pace a little bit and make things quite more interesting. Let's say you want to make an integration test that is you want to test the actual behavior of your Message Handler when a request is dispatched to a controller's action. This would require to host the Web API and then run the unit test but is this possible here? Of course it, and this is the beauty when you create a highly loosely coupled application. All you have to do is **Self host** the web api and setup the appropriate configurations. In our case we are gonna host the web api and also setup Moq instances to be injected for Repositories and Services. Add the following *Startup.cs* file in the UnitTestingWebAPI.Tests project:

Hosting/Startup.cs

```
public class Startup
    {
        public void Configuration(IAppBuilder appBuilder)
        {
            var config = new HttpConfiguration();
            config.MessageHandlers.Add(new HeaderAppenderHandler());
            config.MessageHandlers.Add(new EndRequestHandler());
            config.Filters.Add(new ArticlesReversedFilter());
            config.Services.Replace(typeof(IAssembliesResolver), new Cus

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
            config.MapHttpAttributeRoutes();

            // Autofac configuration
            var builder = new ContainerBuilder();
            builder.RegisterApiControllers(typeof(ArticlesController).As

            // Unit of Work
            var _unitOfWork = new Mock<IUnitOfWork>();
            builder.RegisterInstance(_unitOfWork.Object).As<IUnitOfWork>

            //Repositories
            var _articlesRepository = new Mock<IArticleRepository>();
            _articlesRepository.Setup(x => x.GetAll()).Returns(
                    BloggerInitializer.GetAllArticles()
                );
            builder.RegisterInstance(_articlesRepository.Object).As<IArt

            var _blogsRepository = new Mock<IBlogRepository>();
            _blogsRepository.Setup(x => x.GetAll()).Returns(
                BloggerInitializer.GetBlogs
                );
```

```
            builder.RegisterInstance(_blogsRepository.Object).As<IBlogRe

            // Services
            builder.RegisterAssemblyTypes(typeof(ArticleService).Assembl
               .Where(t => t.Name.EndsWith("Service"))
               .AsImplementedInterfaces().InstancePerRequest();

            builder.RegisterInstance(new ArticleService(_articlesReposit
            builder.RegisterInstance(new BlogService(_blogsRepository.Ob

            IContainer container = builder.Build();
            config.DependencyResolver = new AutofacWebApiDependencyResol

            appBuilder.UseWebApi(config);
        }
    }
```

Notice that it's quite similar to the Startup class we wrote for the Web Application project, except that fake repositories and services are used. Now let's return and write this integration test:

ShouldCallToControllerActionAppendCustomHeader Unit test

```
[Test]
public void ShouldCallToControllerActionAppendCustomHeader()
{
    //Arrange
    var address = "http://localhost:9000/";

    using (WebApp.Start<Startup>(address))
    {
        HttpClient _client = new HttpClient();
        var response = _client.GetAsync(address + "api/articles").Result

        Assert.That(response.Headers.Contains("X-WebAPI-Header"), Is.Tru

        var _returnedArticles = response.Content.ReadAsAsync<List<Articl
        Assert.That(_returnedArticles.Count, Is.EqualTo( BloggerInitiali
    }
}
```

Since the request doesn't contain a *test* literal, it will reach the controller's action and also bring the results. Notice also that the custom header has also been appended.

Action Filters Unit Testing

Recall that we had created an **ArticlesReversedFilter** that when applied it reverses the order of the articles that should be returned. We can either direct unit test this filter or run an integration one. We will see how to do both of them. To direct test an action filter you need to run it's *OnActionExecuted* function by passing a instance of **HttpActionExecutedContext** as a parameter as follow:

ShouldSortArticlesByTitle Unit test (direct)

```
[Test]
public void ShouldSortArticlesByTitle()
{
    var filter = new ArticlesReversedFilter();
    var executedContext = new HttpActionExecutedContext(new HttpActionCo
    {
        Response = new HttpResponseMessage(),
    }, null);

    executedContext.Response.Content = new ObjectContent<List<Article>>(

    filter.OnActionExecuted(executedContext);

    var _returnedArticles = executedContext.Response.Content.ReadAsAsync

    Assert.That(_returnedArticles.First(), Is.EqualTo(_articles.Last()))
}
```

To run an integration test you need to self host the Web API and make the appropriate request. Mind that the filter must be registered in the Startup configuration class.

ShouldCallToControllerActionReverseArticles Unit test (integration)

```
[Test]
public void ShouldCallToControllerActionReverseArticles()
{
    //Arrange
    var address = "http://localhost:9000/";

    using (WebApp.Start<Startup>(address))
    {
        HttpClient _client = new HttpClient();
        var response = _client.GetAsync(address + "api/articles").Result

        var _returnedArticles = response.Content.ReadAsAsync<List<Articl

        Assert.That(_returnedArticles.First().Title, Is.EqualTo(BloggerI
    }
}
```

Media Type formatters Unit Testing

You have created some custom Media Type formatters and you want to test their behavior. Recall the `ArticleFormatter` we created in the *UnitTestingWebAPI.API.Core* project and it's able to return a comma separated string representation of articles. It can only write Article instances, not read ones or understand other type of classes. You need to set the Accept request header to *application/article* in order to apply the formatter. Let's see the Setup configuration of our tests:

```
    [TestFixture]
    public class MediaTypeFormatterTests
    {
        #region Variables
        Blog _blog;
```

```
        Article _article;
        ArticleFormatter _formatter;
        #endregion

        #region Setup
        [SetUp]
        public void Setup()
        {
            _blog = BloggerInitializer.GetBlogs().First();
            _article = BloggerInitializer.GetChsakellsArticles().First()
            _formatter = new ArticleFormatter();
        }
        #endregion
    }
}
```

You can test a MediaTypeFormatter by creating an instance of **ObjectContent**, passing the object to check if can be formatted by the respective formatter, and the formatter itself. If the formatter cannot read or write the passed object an exception will be thrown, otherwise not. For example let's ensure that the *ArticleFormatter* cannot understand Blog instances:

FormatterShouldThrowExceptionWhenUnsupportedType Unit test

```
[Test]
public void FormatterShouldThrowExceptionWhenUnsupportedType()
{
    Assert.Throws<InvalidOperationException>(() => new ObjectContent<Blog
}
```

On the other hand it must work fine with parsing Article objects:

FormatterShouldNotThrowExceptionWhenArticle Unit test

```
[Test]
public void FormatterShouldNotThrowExceptionWhenArticle()
{
    Assert.DoesNotThrow(() => new ObjectContent<Article>(_article, _forma
}
```

And here are some other tests you can run against your custom Media type formatters:

Media Type Formatters Unit tests

```
[Test]
        public void FormatterShouldHeaderBeSetCorrectly()
        {
            var content = new ObjectContent<Article>(_article, new Artic

            Assert.That(content.Headers.ContentType.MediaType, Is.EqualT
        }

        [Test]
        public async void FormatterShouldBeAbleToDeserializeArticle()
```

```
        {
            var content = new ObjectContent<Article>(_article, _formatte

            var deserializedItem = await content.ReadAsAsync<Article>(ne

            Assert.That(_article, Is.SameAs(deserializedItem));
        }

        [Test]
        public void FormatterShouldNotBeAbleToWriteUnsupportedType()
        {
            var canWriteBlog = _formatter.CanWriteType(typeof(Blog));
            Assert.That(canWriteBlog, Is.False);
        }

        [Test]
        public void FormatterShouldBeAbleToWriteArticle()
        {
            var canWriteArticle = _formatter.CanWriteType(typeof(Article
            Assert.That(canWriteArticle, Is.True);
        }
```

Routing Unit Testing

You want to test your routing configuration without hosting Web API. For this you 'll need a helper class that is able to return the Controller type or the controller's action from an instance of a `HttpControllerContext`. Before this you have to create an *HttpConfiguration* with your routing configuration setup in it. Let's see first the helper class:

Helpers/ControllerActionSelector.cs

```
public class ControllerActionSelector
    {
        #region Variables
        HttpConfiguration config;
        HttpRequestMessage request;
        IHttpRouteData routeData;
        IHttpControllerSelector controllerSelector;
        HttpControllerContext controllerContext;
        #endregion

        #region Constructor
        public ControllerActionSelector(HttpConfiguration conf, HttpRequ
        {
            config = conf;
            request = req;
            routeData = config.Routes.GetRouteData(request);
            request.Properties[HttpPropertyKeys.HttpRouteDataKey] = rout
            controllerSelector = new DefaultHttpControllerSelector(confi
            controllerContext = new HttpControllerContext(config, routeD
        }
        #endregion

        #region Methods
        public string GetActionName()
        {
```

```
            if (controllerContext.ControllerDescriptor == null)
                GetControllerType();

            var actionSelector = new ApiControllerActionSelector();
            var descriptor = actionSelector.SelectAction(controllerConte

            return descriptor.ActionName;
        }

        public Type GetControllerType()
        {
            var descriptor = controllerSelector.SelectController(request
            controllerContext.ControllerDescriptor = descriptor;
            return descriptor.ControllerType;
        }
        #endregion
    }
```

And now the RouteTests Setup configuration:

RouteTests.cs

```
[TestFixture]
public class RouteTests
{
    #region Variables
    HttpConfiguration _config;
    #endregion

    #region Setup
    [SetUp]
    public void Setup()
    {
        _config = new HttpConfiguration();
        _config.Routes.MapHttpRoute(name: "DefaultWebAPI", routeTemplate
    }
    #endregion

    #region Helper methods
    public static string GetMethodName<T, U>(Expression<Func<T, U>> expr
    {
        var method = expression.Body as MethodCallExpression;
        if (method != null)
            return method.Method.Name;

        throw new ArgumentException("Expression is wrong");
    }
    #endregion
```

Let's see that a request to *api/articles/5* invokes the ArticlesController.GetArticle(int id) function:

RouteShouldControllerGetArticleIsInvoked Unit test

```
[Test]
public void RouteShouldControllerGetArticleIsInvoked()
{
```

```
        var request = new HttpRequestMessage(HttpMethod.Get, "http://www.chsa

        var _actionSelector = new ControllerActionSelector(_config, request)

        Assert.That(typeof(ArticlesController), Is.EqualTo(_actionSelector.G
        Assert.That(GetMethodName((ArticlesController c) => c.GetArticle(5))
            Is.EqualTo(_actionSelector.GetActionName()));
    }
```

We used some reflection to get controller's action name. In the same way we can test that the post action is invoked:

RouteShouldPostArticleActionIsInvoked Unit test

```
    [Test]
    public void RouteShouldPostArticleActionIsInvoked()
    {
        var request = new HttpRequestMessage(HttpMethod.Post, "http://www.chs

        var _actionSelector = new ControllerActionSelector(_config, request)

        Assert.That(GetMethodName((ArticlesController c) =>
            c.PostArticle(new Article())), Is.EqualTo(_actionSelector.GetAct
    }
```

You will probably want to test that an invalid route is not working:

RouteShouldInvalidRouteThrowException Unit test

```
    [Test]
    public void RouteShouldInvalidRouteThrowException()
    {
        var request = new HttpRequestMessage(HttpMethod.Post, "http://www.chsa

        var _actionSelector = new ControllerActionSelector(_config, request);

        Assert.Throws<HttpResponseException>(() => _actionSelector.GetActionN
    }
```

Conclusion

We have seen many aspects of Unit Testing in Web API stack such as mocking the Service layer, unit testing Controllers, Message Handlers, Filters, Custom Media type Formatters and the routing configuration. Try to always writing unit tests for your application and you will never regret it. The most unit tests you write the more benefits you will get. For example a simple change in your repository may brake many aspects in your application. If the appropriate tests have been written, then in the first run you should see all broken parts of your application immediately. I hope you liked the post as much I did. You can download the source code for this project **here**.

In case you find my blog's content interesting, register your email to receive notifications of new posts and follow *chsakell's Blog* on its Facebook or Twitter accounts.

8 replies

Arturas

May 10, 2015 • 1:34 pm

Great article!

Catriel Martinez

May 11, 2015 • 9:02 pm

Spectacular article!!
I have a problem when I try to debug the app it returns the next error:
Could not load file or assembly 'System.Web.Http, Version=5.2.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35' or one of its dependencies. The located assembly's manifest definition does not match the assembly reference. (Exception from HRESULT: 0x80131040)

Do you know what it could be?

Christos S.

May 12, 2015 • 12:38 am

Hi Martinez, thanks!
It's seems that something went wrong with your dependencies versions. You probably have different version in the config file than the referenced one for the *System.Web.Http* or Autofac is searching for different version (5.2.0.0) than the referenced one (you should have 5.2.3.0). You can try some of the following resolutions:

- Clean or remove the dlls from the bin folder
- Set *Copy Local* to *true* in the properties for assemblies *System.Web.Http* & *System.Web.Http.WebHost*
- Re-install Web-API package by running in Nuget-package console the command **Update-Package Microsoft.AspNet.WebApi -reinstall**
- Make sure you have installed **System.Web.Http.WebHost** as well

If this doesn't work try the following change in the Web.config:

```
1  <dependentAssembly>
2      <assemblyIdentity name="System.Web.Http" publicKeyToken="31bf3856a
3      <bindingRedirect oldVersion="0.0.0.0-5.2.0.0" newVersion="5.2.3.0"
4    </dependentAssembly>
```

Catriel Martinez

May 12, 2015 • 1:49 pm

It worked perfect!!
I cleaned the solution, and set Copy Local to true and changed the Web.Config.
Thank you!!!!

Robert Eberhart

July 27, 2015 • 8:20 pm

Thank you very much for this response to Mr. Martinez. I searched all over for a solution to this problem. Before I arrived at your article, I performed the items you detailed in your checklist, but your suggestion of adding the dependentAssembly element to the *.config file solved the issue.

That's at least two hours I won't get back, but at least I know what to do in the future.

Thanks again!

tomazeli

August 18, 2015 • 6:25 pm

Keep in mind that when self hosting the WebAPI inside Unit tests, the user running the unit tests should have permission to bind (listening) on local ports (usually admin). So if you are planning to run the unit tests in a CI environment might be a problem. Another solution would be to run an in-memory httpserver. I haven't fully tested but there is an example here.
http://blogs.msdn.com/b/youssefm/archive/2013/01/28/writing-tests-for-an-asp-net-webapi-service.aspx?CommentPosted=true#commentmessage

Christos S.

August 18, 2015 • 6:52 pm

Very good point tomazeli, thanks!

huoxudong125

December 23, 2015 • 2:53 pm

Reblogged this on **ITelite**.