Configure Dependency Injection with Web API

Here you will learn how to configure and use IoC container for dependency injection with Web API.

There are many IoC containers available for dependency injection such as Ninject, Unity, castleWidsor, structuremap etc. Here we will use Ninject for dependency injection.

The following is our sample Web API that uses instance of a class that implements IRepository.

Example: Simple Web API Controller

public class StudentController : ApiController

{

private IRepository \_repo = null;

public StudentController(IRepository repo)

{

\_repo = repo;

}

public IList<Student> Get()

{

return \_repo.GetAll();

}

}

The following are IRepository and StudentRepository class.

Example: Repository

public interface IRepository

{

IList<Student> GetAll();

}

public class StudentRepository : IRepository

{

public IList<Student> GetAll()

{

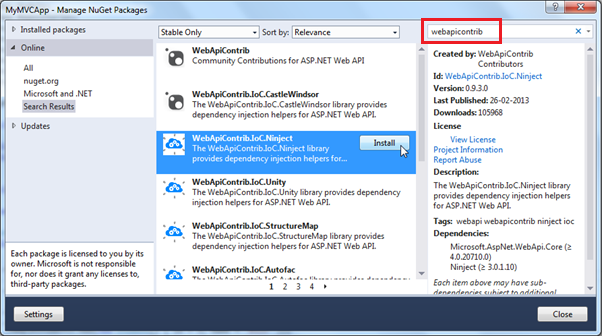
//return students from db here

}

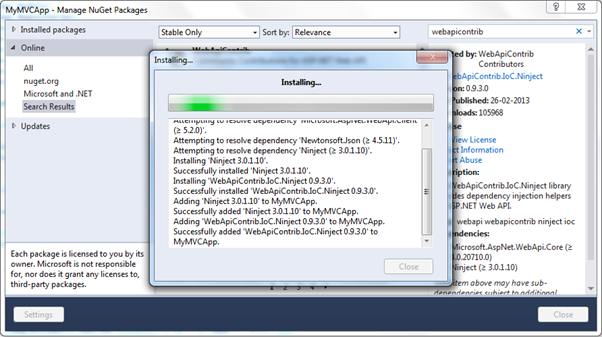
}

Now, let's use Ninject which will inject StudentRepository class in StudentController.

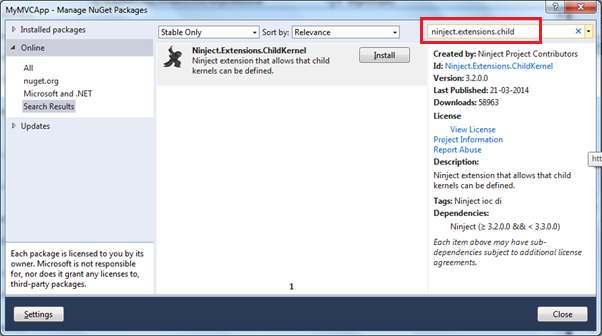
First of all, you need to install Ninject library for Web API using NuGet. To do this, right click on your project in the solution explorer -> click on **Manage NuGet packages..**. This will open NuGet popup. Now search for webapicontrib in the search box as shown below.

[](https://www.tutorialsteacher.com/Content/images/webapi/configure-DI-1.png)Web API Configuration

As you can see, this will list all the IoC containers for Web API. Select WebApiContrib.IoC.Ninject and click **Install**.

[](https://www.tutorialsteacher.com/Content/images/webapi/configure-DI-2.png)Web API Configuration

Now, you need to install Ninject.Extensions.ChildKernel. Search for it and install it.

[](https://www.tutorialsteacher.com/Content/images/webapi/configure-DI-3.png)Web API Configuration

So now after installing necessary packages for Ninject, we need to configure it.

In order to use dependency injection with Web API, we need to create a resolver class that implements IDependencyResolver interface. Here, we have created NinjectResolver class in Infrastructure folder in our Web API project as shown below.

Example: DI Resolver

public class NinjectResolver : IDependencyResolver

{

private IKernel kernel;

public NinjectResolver() : this(new StandardKernel())

{

}

public NinjectResolver(IKernel ninjectKernel, bool scope = false)

{

kernel = ninjectKernel;

if (!scope)

{

AddBindings(kernel);

}

}

public IDependencyScope BeginScope()

{

return new NinjectResolver(AddRequestBindings(new ChildKernel(kernel)), true);

}

public object GetService(Type serviceType)

{

return kernel.TryGet(serviceType);

}

public IEnumerable<object> GetServices(Type serviceType)

{

return kernel.GetAll(serviceType);

}

public void Dispose()

{

}

private void AddBindings(IKernel kernel)

{

// singleton and transient bindings go here

}

private IKernel AddRequestBindings(IKernel kernel)

{

kernel.Bind<IRepository>().To<StudentRepository>().InSingletonScope();

return kernel;

}

}

Now, we need to configure NijectResolver with Web API in the WebApiConfig class as shown below.

Example: Set DI Resolver

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

config.DependencyResolver = new NinjectResolver();

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

}

}

As you can see above, HttpConfiguration.DependencyResolver is set to NinjectResolver. So now, Web API will use NinjectResolver class to create the objects it needs.