# Developing a WebApi application – Step by step tutorial

1. Create a blank solution
2. Choose REAL folders for the projects
3. Create new ASP.NET Web application -> Web Api
   1. Choose authentication type
4. Delete :
   1. Views
   2. Scripts
   3. Fonts
   4. Areas
   5. App\_Data
   6. In AppStart:
      1. BundleConfig
      2. FilterConfig
      3. RouteConfig
   7. Controllers -> HomeController.cs
   8. Favicon.ico
   9. Content
   10. Project\_Readme.html
   11. In Global.asax.cs
       1. FilterConfig…
       2. BundleConfig…
       3. RouteConfig…
       4. Area.Registration…
5. Fix Connection String
6. Create Code first Database
7. In App\_Start – DatabaseConfig.cs

public static void Initialize()

{

Database.SetInitializer(new MigrateDatabaseToLatestVersion<MusicSystemDbContext, Configuration>());

MusicSystemDbContext.Create().Database.Initialize(true);

}

1. In Global.asax.cs
   1. DatabaseConfig.Initialize();

## Creating Controllers

1. In a controller
   1. This.User is the currently logged user
      1. This.User.Identity.Name – username
      2. This.User.Identity.IsAuthenticated – is the user authenticated
   2. [Route("api/artists/all")] predefined route
   3. Create Folder “Somethings” for the transfer objects
      1. SomethingDetailsResponseModel
      2. SaveSomethingRequestModel

## Repository Pattern

1. Create IRepository interface
2. Create EfGenericRepository
3. In controller
   1. private readonly IRepository<Something> data;
   2. in Constructor
      1. this.data = new EfGenericRepository<Artist>(new MusicSystemDbContext());
   3. NB! When requesting multiple generic repositories, the dbContext has to be the same

var db = new SomethingDbContext();

this.somethings = new EfGenericRepository<Something>(db);

this.otherThings = new EfGenericRepository<OtherThing>(db);

## Dependency Inversion with Ninject

1. Install Ninject.Web.WebApi.WebHost
2. Rename NinjectWebCommon to NinjectConfig
3. In RegisterServices
   1. kernel.Bind<IMusicSystemDbContext>().To<MusicSystemDbContext>(); kernel.Bind(typeof(IRepository<>)).To(typeof(EfGenericRepository<>));
4. Multiple binding something Exception
   1. Update all ninject related packets
5. Multiple Context exception – because there are more than one contexts (just like in the repository pattern the NB!)
   1. Install Ninject.Extensions.Conventions
   2. kernel.Bind<IMusicSystemDbContext>().To<MusicSystemDbContext>()
      1. .InRequestScope() – every request has its own dbContext
      2. .InSingletonScope() – Only one context for the application
      3. .InThreadScope() – for multithreading applications
   3. Kernel.Bind(b => b.From(“MusicSystem.Data”)

.SelectAllClasses()

.BindDefaultInterface());

Needs Using Sys

## Enable CORS

1. Install Microsoft.AspNet.WebApi.Cors
2. In WebApiConfig
   1. Add config.EnableCors();
   2. [EnableCors(“\*”,“\*”,“\*”)]

## AutoMapper - Configuration and usage

1. Install AutoMapper
2. In controller
   1. Using Automapper.QueriableExtensions

Mapper.CreateMap<Album, AlbumDetailsResponseModel>()

.ForMember(

a => a.Artists,

opts => opts.MapFrom(a => a.Artists.Select(ar => ar.Name)))

.ForMember(  
a => a.Songs,   
opts => opts.MapFrom(a => a.Songs.Select(s => s.Title)));

var album = data.All().ProjectTo<AlbumDetailsResponseModel>()…

Create map is a slow operation, so we need to refactor

1. In App\_Start
   1. AutoMapperConfig.cs
2. Create Folder Infrastructure/Mappings
   1. Create IMapFrom interface
      1. Used for repeated members in the both classes
   2. Create IHaveCustomMappings interface
      1. Used for Different members in the classes
3. In Global.asax..cs
   1. AutoMapperConfig.RegisterMappings(Assembly.Load("MusicSystem.Api"));
4. For Reverse mapping
   1. var dbAlbum = Mapper.Map<Album>(model);

## Object Factory

1. In Infrastructure
   1. Create ObjectFactory.cs
   2. In NinjectConfig.cs -> CreateKernel()
      1. Infrastructure.ObjectFactory.Initialize(kernel);
   3. Used where you cannot use Dependency Injection, but we need an instance of that type