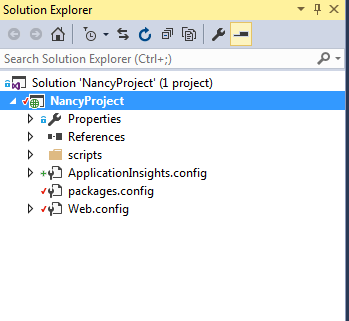
Nancy Tutorial

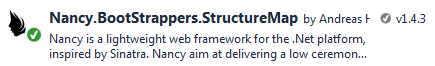
Installation:

Start an empty VS web application



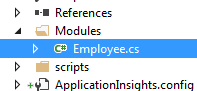
Install the Nuget packages for Nancy as following:





Create a Folder Module

Add a class file as module Employee.cs



1. **A ‘Hello world’ Get request:**

Open up the Employee.cs and write default route for Get method in constructor saying ‘hello Jyoti’

using Nancy;

namespace NancyProject.Modules

{

public class Employee:NancyModule

{

public Employee()

{

Get["/name"] = \_ => "Hello Jyoti!";///string output GET method

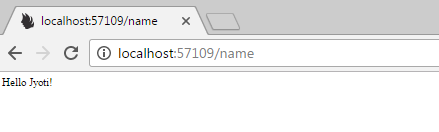
}

}

}

Hit F5 and in browser it shall show up the result. Make sure you have entered the full endpoint name

[http://localhost:<port>/name](http://localhost:%3cport%3e/name)



*For further testing we’ll use ‘Advanced Rest Client’ plug-in for chrome…because as I experienced (may be you have solution to that), Chrome does not provide an option in browser to force the Request header content-type to be “application/json”. This lack of flexibility may cause problems while running the endpoints from Chrome browser directly.*

Add few more routes now.

1. **Parameterized GET request:**

Expand the constructor code like following:

using Nancy;

using Nancy.ModelBinding;

using NancyProject.Models;

using System.Collections.Generic;

namespace NancyProject.Modules

{

public class Employee:NancyModule

{

public Employee()

{

Get["/name"] = \_ => "Hello Jyoti!";///string output GET method

///Parameterised GET method

Get["/employee/{id}"] = args =>

{

int id = args.id;

///*negotiate* uses default accept headers of 'json/application' for content negotiations

return Negotiate

.WithStatusCode(HttpStatusCode.OK)

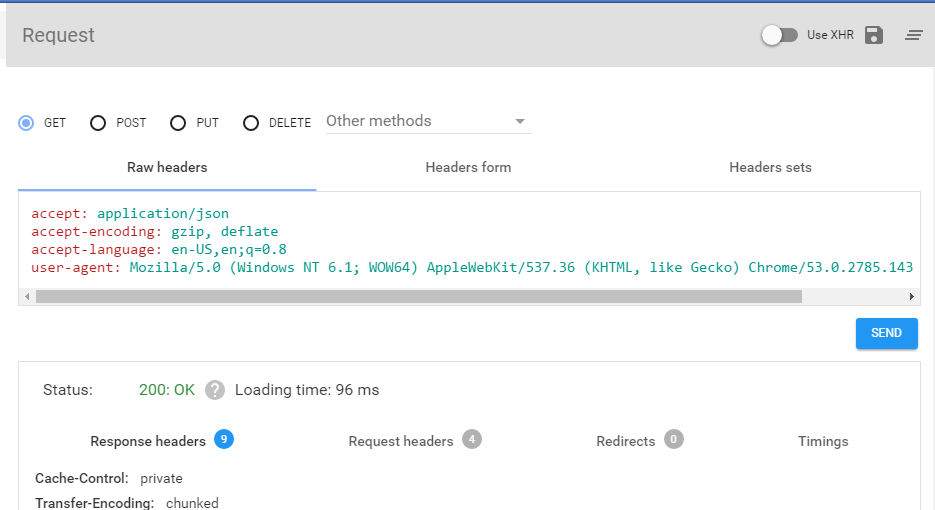
.WithModel(id);

};

}

}

In ARC, run the URL as following (*keep note of header content type being set to json/application*):



1. Parameterized GET request with Model binding: The code shall look like something below after adding the new Get method –

using Nancy;

using Nancy.ModelBinding;

using NancyProject.Models;

using System.Collections.Generic;

namespace NancyProject.Modules

{

public class Employee:NancyModule

{

public Employee()

{

Get["/name"] = \_ => "Hello Jyoti!";///string output GET method

///Parameterised GET method

Get["/employee/{id}"] = args =>

{

int id = args.id;

///negotiate uses default accept headers of 'json/application' for content negotiations

return Negotiate

.WithStatusCode(HttpStatusCode.OK)

.WithModel(id);

};

///multi-level parameterised GET with Model output

Get["/{role}/{numberofyears}"] = args => {

var empAssignment = this.Bind<Assignment>();///bind the incoming request to a defined Model structure. It requires parameters to have same name as the properties defined in the model.

///prepare output object model

var emps = new List<Emp>{

new Emp {Id=1,Assignment=empAssignment},

new Emp {Id=2,Assignment=empAssignment },

new Emp {Id=3,Assignment=empAssignment }

};

return Negotiate

.WithStatusCode(HttpStatusCode.OK)

.WithModel(emps);

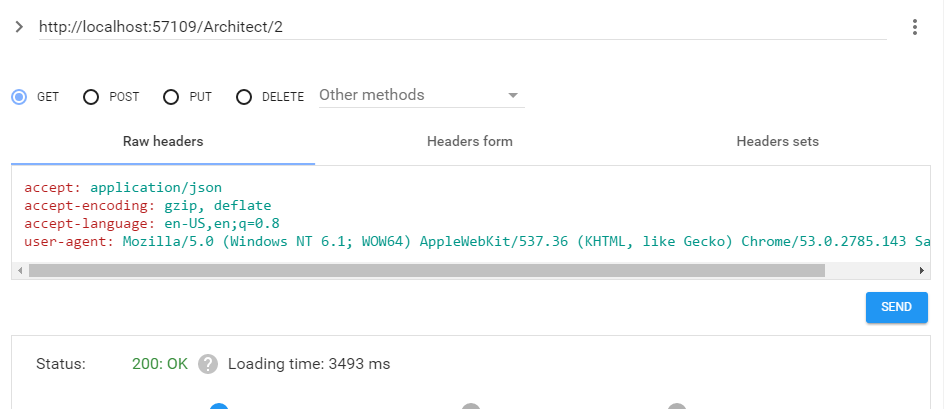
};

}

}

}

Build the program and send the request in ARC as following:



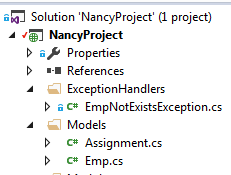
The JSO formatted output body response should look like as following:



Exception Handling:

A simple exception handling example:

Create a folder for *ExceptionHandlers* in the solution -> add a exceptionHandler class *EmpNotExistsExceptions.cs* as below:



Edit the exception class as following:

using System;

namespace NancyProject.ExceptionHandlers

{

public class EmpNotExistsException:System.Exception

{

public EmpNotExistsException()

{

//throw base.InnerException;

}

}

}

Create a sample exception scenario in code by editing that as following:

///Parameterised GET method

Get["/employee/{id}"] = args =>

{

int id = args.id;

**///Exception scenario**

if (id > 1000)

{

throw new ExceptionHandlers.EmpNotExistsException();

}

///negotiate uses default accept headers of 'json/application' for content negotiations

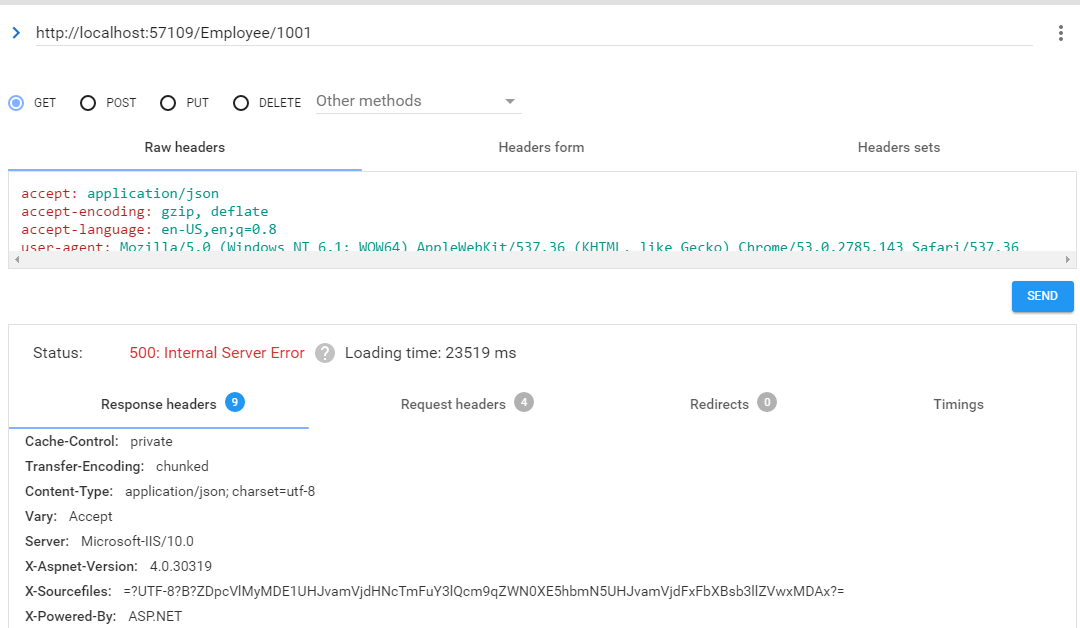
return Negotiate

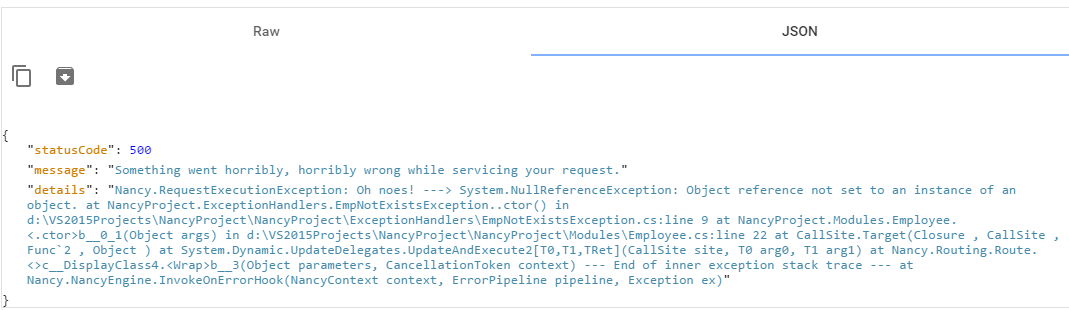
.WithStatusCode(HttpStatusCode.OK)

.WithModel(id);

};

Build the solution and check the output in ARC for empid>1000





Application wide Exception handler: for better maintenance of code, in order to have a centralized application-level exception handling, use the Nancy bootstrapper which is more like a startup file from MVC perspective. It would involve followings:

* Create a new folder – *NancyBootStrapper*
* Add a class file to this folder named as *NancyBooStrapper.cs*
* Edit the *NancyBootStrapper.cs* as following:

using Nancy;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using Nancy.Bootstrapper;

using Nancy.TinyIoc;

using System.Text;

namespace NancyProject.NancyBootStrapper

{

public class NancyBootStrapper:DefaultNancyBootstrapper

{

protected override void ApplicationStartup(TinyIoCContainer container, IPipelines pipelines)

{

///assign the handler for onError event

pipelines.OnError += (context, exception) =>

{

if (exception is ExceptionHandlers.EmpNotExistsException)

{

return new Nancy.Response()

{

StatusCode = HttpStatusCode.NotFound,

ContentType = "text/html",

Contents = (stream) =>

{

var errorMessage = Encoding.UTF8.GetBytes(exception.Message);

stream.Write(errorMessage, 0, errorMessage.Length);

}

};

}

return HttpStatusCode.InternalServerError;

};

//base.ApplicationStartup(container, pipelines);

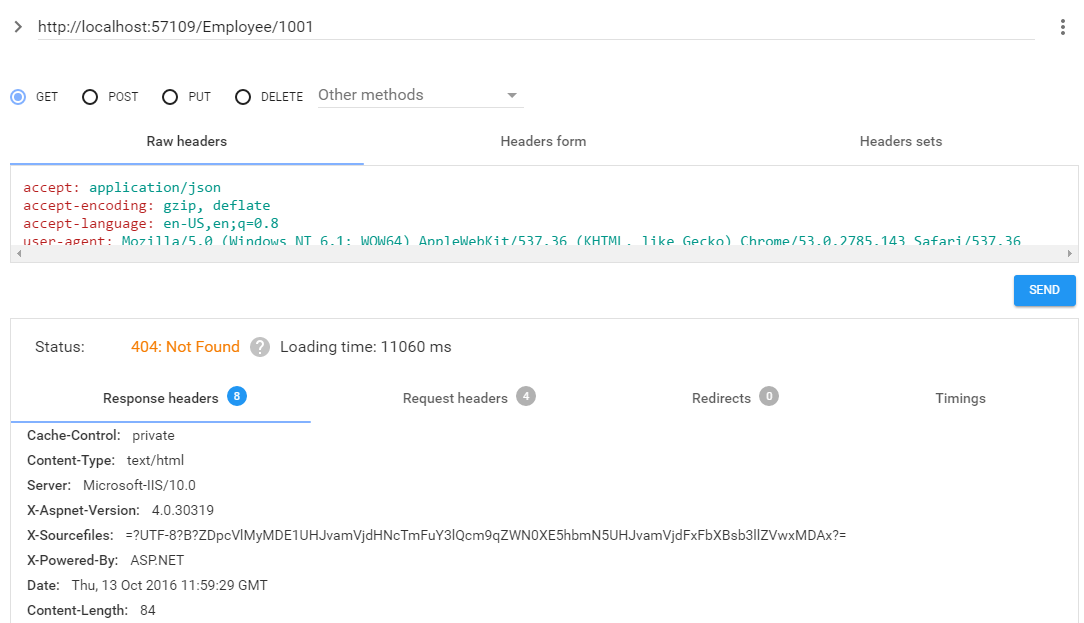
}

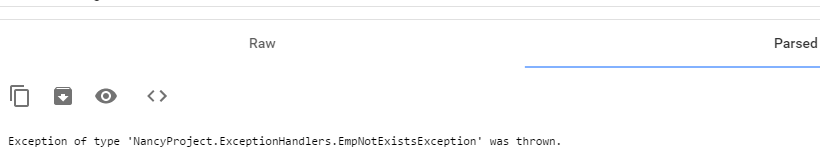
}

}

Now, compile theapplication and run, this exception handler shall come in play.

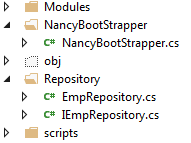
In ARC, you should see following screen grabs:





Dependency Injection using IRepository: In order to implement DI, follow these steps:

* Create a public interface IEmpRepository.cs in a folder Repository
* Add an implementation class for the above interface as EmpRepository.cs in the same folder. The folder structure shall look like as following:



* Now, edit the IEmpRepository.cs to add one method *GetEmpById*(id) signature as following:

using NancyProject.Models;

namespace NancyProject.Repository

{

public interface IEmpRepository

{

Emp GetEmpById(int id);

}

}

* Implement the above method in EmpRepository.cs as following:

using NancyProject.Models;

using NancyProject.ExceptionHandlers;

namespace NancyProject.Repository

{

public class EmpRepository : IEmpRepository

{

Emp IEmpRepository.GetEmpById(int id)

{

if (id > 1000)

{

throw new EmpNotExistsException();

}

return new Emp { Id = id,Assignment= new Assignment { Role = "Architect", NumberOfYears = 8 } };

}

}

}

Now, that Repository is all set, let’s incorporate that by injecting into application:

* In Employee.cs, inject the interface by editing the code as following:

using Nancy;

using Nancy.ModelBinding;

using NancyProject.Models;

using NancyProject.Repository;

using System.Collections.Generic;

namespace NancyProject.Modules

{

public class Employee:NancyModule

{

IEmpRepository \_iEmpRepository;

public Employee(IEmpRepository iEmpRepository)

{

\_iEmpRepository = iEmpRepository;

Get["/name"] = \_ => "Hello Jyoti!";///string output GET method

///Parameterised GET method

Get["/employee/{id}"] = args =>

{

int id = args.id;

var emp=\_iEmpRepository.GetEmpById(id);

//if (id > 1000)

//{

// throw new ExceptionHandlers.EmpNotExistsException();

//}

///negotiate uses default accept headers of 'json/application' for content negotiations

return Negotiate

.WithStatusCode(HttpStatusCode.OK)

//.WithModel(id);

.WithModel(emp);

};

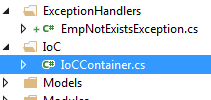
...

...

Note that in above code, besides, injecting the dependency, since the exception raise code was moved to EmpRepository.cs earlier, so, the same code has been commented out in constructor code above.

Now the type has been injected into the code, in order to make it work need to provide a DI framework. Please follow the steps below:

* Create a folder IoC and add a IoCContainer module to that as following:



The IoCContainer.cs shall look like the following:

using StructureMap;

using StructureMap.Graph;

namespace NancyProject.IoC

{

public class IoCContainer

{

public static void Configure(IContainer Container)

{

Container.Configure(config => config.Scan(c =>

{

c.TheCallingAssembly();

c.WithDefaultConventions();

}

));

}

}

}

* Let’s enable the IoC in Bootstrapper module. Comment out the existing class definition for NancyBootStrapper and create new definition by
  + Inheriting the class from Nancy.Bootstrappers.StructureMap .StructureMapNancyBootstrapper
  + In method ApplicationStartup, Changing the container parameter to type StructureMap.IContainer
  + Overiding the method ConfigureApplicationContainer

The final code of NancyBootStrapper.cs shall look like following:

using Nancy;

using Nancy.Bootstrapper;

using System.Text;

using Nancy.Bootstrappers.StructureMap;

using StructureMap;

using NancyProject.IoC;

namespace NancyProject.NancyBootStrapper

{

//public class NancyBootStrapper:DefaultNancyBootstrapper

//{

// protected override void ApplicationStartup(TinyIoCContainer container, IPipelines pipelines)

// {

// ///assign the handler for onError event

// pipelines.OnError += (context, exception) =>

// {

// if (exception is ExceptionHandlers.EmpNotExistsException)

// {

// return new Nancy.Response()

// {

// StatusCode = HttpStatusCode.NotFound,

// ContentType = "text/html",

// Contents = (stream) =>

// {

// var errorMessage = Encoding.UTF8.GetBytes(exception.Message);

// stream.Write(errorMessage, 0, errorMessage.Length);

// }

// };

// }

// return HttpStatusCode.InternalServerError;

// };

// //base.ApplicationStartup(container, pipelines);

// }

//}

public class NancyBootStrapper : StructureMapNancyBootstrapper

{

protected override void ConfigureApplicationContainer(IContainer existingContainer)

{

IoCContainer.Configure(existingContainer);

}

protected override void ApplicationStartup(IContainer container, IPipelines pipelines)

{

///assign the handler for onError event

pipelines.OnError += (context, exception) =>

{

if (exception is ExceptionHandlers.EmpNotExistsException)

{

return new Nancy.Response()

{

StatusCode = HttpStatusCode.NotFound,

ContentType = "text/html",

Contents = (stream) =>

{

var errorMessage = Encoding.UTF8.GetBytes(exception.Message);

stream.Write(errorMessage, 0, errorMessage.Length);

}

};

}

return HttpStatusCode.InternalServerError;

};

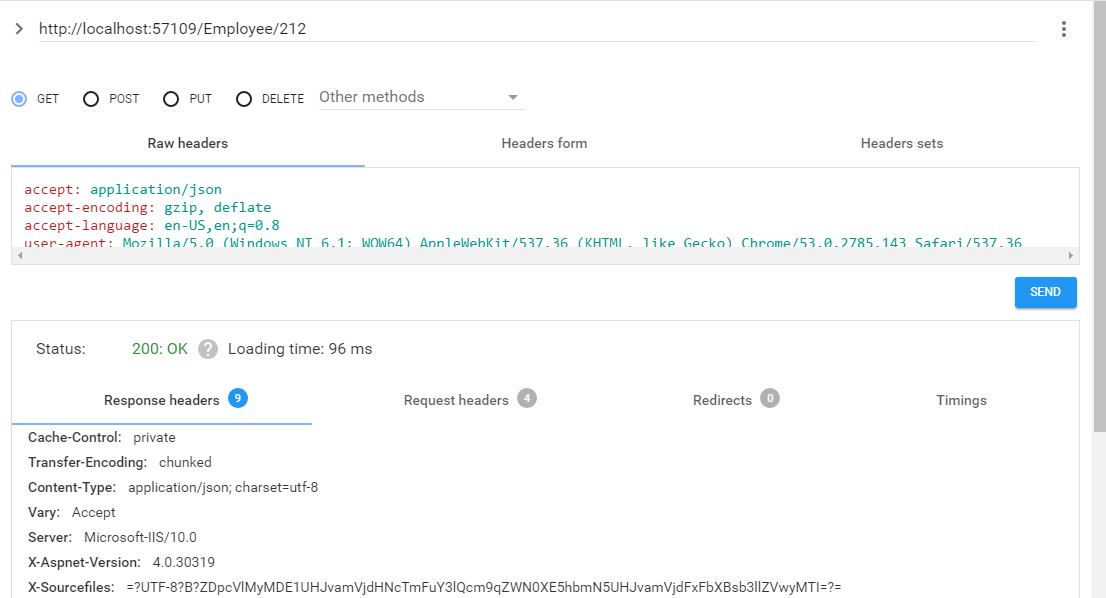
//base.ApplicationStartup(container, pipelines);

}

}

}

* Compile the code and run. In ARC, it shall show following response:





Just notice the input for Id=212, as it turns out in Response as well.

---