1. Nancy – córka Sinatry (Sinatra – prosty framework do Ruby).
2. Brak zbędnych referencji, większość kodu (webowego) napisana przez Autorów.
3. Podejście Super-Duper-Happy-Path, czyli: it just works, Easily customizable, Low ceremony, Low friction (tarcie)
4. Tworzenie pierwszej aplikacji:
    1. Pusty Web Application (dzięki temu mamy np. Web.config)
    2. Wersja as string
        1. Pokazać instalację z managera
        2. Pokazać instalację z konsoli (`Install-Package Nancy`),
        3. Instalacja hostingu (`Install-Package Nancy.Hosting.AspNet`)
        4. Pokazać, co dodało się do web.config (i powiedzieć, że Nancy przechwytuje Handlery),
        5. Moduły dziedziczą z klasy NancyModule
        6. Większość kodu modułów jest pisana w konstuktorach.
        7. Add Class → HomeModule.cs
        8. `using Nancy;`

```
namespace WebApplication1
{
    public class HomeModule:NancyModule
    {
        public HomeModule()
        {
            Get["/"] = _ => "Witaj świecie. Zażółć gęślą jaźń";
        }
    }
}
```

        9. budowanie Nancy polega na przeglądzie Dlli (przez refleksje), bo:
        10. Add Class → myMessageProvider
        11. `namespace WebApplication1`

```
{
    public interface IMessageProvider
    {
        string GetMessage();
    }
    public class myMessageProvider:IMessageProvider
    {
        public string GetMessage()
        {
            return "message from magic powers";
        }
    }
}
```

        12. W HomeModule:
        13. `public HomeModule(IMessageProvider messageProvider)`
        `Get["/msg"] = _ => messageProvider.GetMessage();`
        14. Odpowiedzialny za to jest kontener Dependency Injection o nazwie TinyIOC
    3. Wersja as html
        1. `Get["/rzr"] = _ => View["myView"];`
        2. Pokazać, że Internal error
        3. Zakładamy katalogi Views, Views->Home, plik HTML: myView.cshtml
        4. `@using System`

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Witamy w htmlu</title>
</head>
<body>
    page generated at @DateTime.Now.ToString("dd.MM.yyyy hh:mm")
</body>
</html>
```

        5. Instalujemy `Install-Package Nancy.ViewEngines.Razor.` Zaczyna działać
        6. Moduł Cars: dodajemy katalog: Views->Cars, w nim plik index.cshtml, będący kopią myView, ze zmienionym tytułem
        7. dodajemy klasę CarsModule: `using Nancy;`

```
namespace WebApplication1
{   public class CarsModule:NancyModule
    {   public CarsModule() : base("cars")
        {
            Get["/"] = _ => View["index"];
        }    } }
```

8. To nam pod adresem /cars wyświetli stronę
9. Dodajemy klasę Car:
10.
```
public class Car    {
public int Id { get; set; }
public int Vin { get; set; }
public int Brand { get; set; }}
```
11. Tworzymy jej obiekt i zwracamy go jako model (zamiast data access).
12.
```
Get["car/{id}"] = x =>
{
    var id = x.id;
    var model = new Car() {Brand = "Toyota", Vin = "JPA12345678901234", Id = id};
    return View["car", model];
};
```
13. Tworzymy widok car.cshtml, Jak index, zmieniony tytuł, w body car details:
14.
```
<ul>
<li>id :  @Model.Id</li>
<li>Vin :  @Model.Vin</li>
<li>Brand :  @Model.Brand</li>
</ul>
```
15. pokazać, że działa,
16. że działa tylko po podaniu `return View[model];`
17. a nawet return model; mądra Nancy.
18. Json, xml
19. Skąd wiadomo, że chcemy zobaczyć htmla, bo jest nagłówek request, że przeglądarka oczekuje html, nazywa się to content negotiations
20. _nancy
21. `using Nancy;`

```
using Nancy.Diagnostics;

namespace WebApplication1
{
    public class CustomBootstraper:DefaultNancyBootstrapper
    {
        protected override Nancy.Diagnostics.DiagnosticsConfiguration DiagnosticsConfiguration
        {
            get {return new DiagnosticsConfiguration() {Password = "asdf555", Path = "debug"};}
        }
    }
}
```

```
Console Application: Install-Package Nancy.Hosting.Self,
using System;
using Nancy;
using Nancy.Hosting.Self;
using WebApplication1;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            new NancyHost(new CustomBootstraper(),new Url("http://localhost:7345")).Start();
            Console.WriteLine("Working");
            Console.ReadLine();
        }
```

http://www.codeproject.com/Articles/680119/Creating-a-REST-Server-for-a-CRUD-Web-Application