Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής – Ανάπτυξη Λογισμικού και Τεχνητής Νοημοσύνης»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**ΕΚΜΑΘΗΣΗ ΤΟΥ ΠΛΑΙΣΙΟΥ ΑΝΑΠΤΥΞΗΣ (FRAMEWORK) .NET Core** **LEARNING THE WEB DEVELOPMENT FRAMEWORK .NET Core**
Ονοματεπώνυμο Φοιτητή	**Νικόλαος Ταξίδης**
Πατρώνυμο	**Θεόδωρος**
Αριθμός Μητρώου	**ΜΠΣΠ/19047**
Επιβλέπων	**Ευάγγελος Σακκόπουλος, Επίκουρος Καθηγητής**

Ημερομηνία Παράδοσης **Νοέμβριος 2021**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Ευάγγελος Σακκόπουλος Επίκουρος Καθηγητής	Ευθύμιος Αλέπης Αναπληρωτής Καθηγητής	Διονύσιος Σωτηρόπουλος Επίκουρος Καθηγητής

ΠΕΡΙΛΗΨΗ

Αυτή η εργασία επικεντρώνεται στην εκμάθηση του πλαισίου ανάπτυξης ιστοσελίδων για τη δημιουργία εφαρμογών Ιστού στην πλατφόρμα ASP.NET Core. Υπάρχουν 10 κεφάλαια όπου παρέχεται κώδικας της ASP.NET Core σε κάθε κεφάλαιο μαζί με την κατάλληλη εξήγηση. Η δομή των μαθημάτων είναι τέτοια ώστε να εξασφαλίζει μια ομαλή ρόη στοχευμένης μάθησης με τη βοηθητική χρήση του πηγαίου κώδικα και screenshots της εμφάνισης των προγραμμάτων.
Η εφαρμογή έχει σχεδιαστεί με τέτοιο τρόπο ώστε να μπορεί να είναι κατανοητή από τον οποιοδήποτε χρήστη.

ABSTRACT

This work focuses on learning the ASP.NET Core web framework. There are 10 chapters where ASP.NET Core code is provided in each chapter along with the appropriate explanation. The structure of the courses is such as to ensure a smooth flow of targeted learning with the help of the source code and screenshots of the appearance of the programs. The application is designed in such a way that it can be understood by any user.

Περιεχόμενα

Περιεχόμενα Εικόνων

ΕΙΣΑΓΩΓΗ - ΣΚΟΠΟΣ

Σκοπός της παρούσας εργασίας είναι η εκμάθηση της γλώσσας προγραμματισμού ASP.NET Core.

Η ASP.NET είναι ένα δημοφιλές πλαίσιο ανάπτυξης ιστοσελίδων για τη δημιουργία εφαρμογών Ιστού στην πλατφόρμα .NET. Η ASP.NET Core είναι η έκδοση ανοιχτού κώδικα της ASP.NET, που τρέχει σε macOS, Linux και Windows. Η ASP.NET Core κυκλοφόρησε για πρώτη φορά το 2016 και είναι ένας επανασχεδιασμός προηγούμενων εκδόσεων της ASP.NET μόνο για Windows.

Η δομή της εργασίας είναι χωρισμένη σε 10 κεφάλαια όπου παρέχεται κώδικας της ASP.NET Core μαζί με την κατάλληλη εξήγηση για το κάθε ένα.

Κάθε κεφάλαιο έχει κάποια προγράμματα που μπορεί ο χρήστης να τα τρέχει μαζί με την εξήγηση του κώδικα.
Επίσης, κάθε κεφάλαιο περιέχει και τη σχετική θεωρία της ASP.NET Core.

Ο χρήστης διαθέτει απαραίτητα το Visual Studio 2019 ή κάποια νεότερη έκδοσή του για να μπορεί να τρέχει τα προγράμματα του κάθε κεφαλαίου.

Για την υλοποίησή της χρησιμοποιήθηκαν όλα τα εργαλεία που χρησιμοποιεί η ASP.NET Core, όπως η γλώσσα προγραμματισμού C#, η γλώσσα σήμανσης CSHTML και Razor σελίδες.

Κεφάλαιο 1 - Τα πρώτα βήματα στην ASP.NET Core

Η ASP.NET Core είναι ένα δωρεάν διαδικτυακό πλαίσιο(framework) ανοιχτού κώδικα και διάδοχος της ASP.NET, που αναπτύχθηκε από τη Microsoft. Είναι ένα σπονδυλωτό framework που τρέχει τόσο στο πλήρες .NET Framework, στα Windows όσο και στην πολλαπλή πλατφόρμα .NET.

Ωστόσο, η έκδοση 3 της ASP.NET Core λειτουργεί μόνο σε .NET Core που υποστηρίζει το .NET Framework. Το framework είναι μια πλήρης επανεγγραφή που ενώνει τα προηγούμενα ξεχωριστά ASP.NET MVC και ASP.NET Web API σε ένα ενιαίο μοντέλο προγραμματισμού. Παρά το γεγονός ότι είναι ένα νέο πλαίσιο, βασισμένο σε μια νέα στοίβα ιστού, έχει υψηλό βαθμό συμβατότητας με την ASP.NET.

Οι εφαρμογές ASP.NET Core υποστηρίζουν εκδόσεις παράλληλα, στις οποίες διαφορετικές εφαρμογές, που εκτελούνται στο ίδιο μηχάνημα, μπορούν να στοχεύσουν διαφορετικές εκδόσεις της ASP.NET Core. Αυτό δεν είναι δυνατό με προηγούμενες εκδόσεις της ASP.NET.

Ακολουθούν τα πρώτα βήματα της ASP.NET Core.

Program.cs

```
class Program
   {
      public static void Main(string[] args)
      {
         Console.WriteLine("Γεια σου Κόσμε !!!!");
         Console.Read();
      }
 }
```

Απλώς τυπώνει στην κονσόλα `Γεια σου Κόσμε !!!!`.

Εικόνα 1 - Εμφάνιση στην κονσόλα: Γεια σου Κόσμε !!!! .

HelloWorldWeb

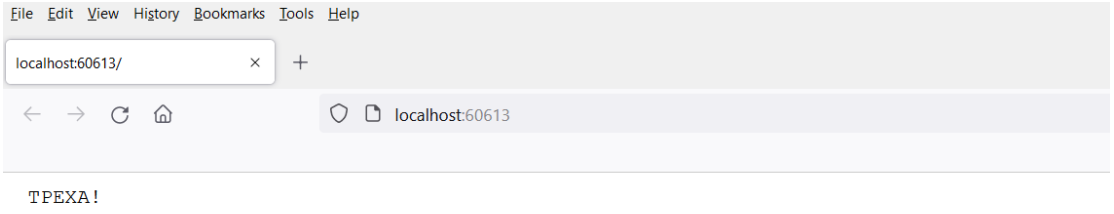
Startup.cs

```
public class Startup
  {
    public void Configure(IApplicationBuilder app)
    {
      app.Run(async (context) =>
      {
        await context.Response.WriteAsync(" ΤΡΕΧΑ! ");
      });
    }
  }
```

Τυπώνει το ΤΡΕΧΑ!.

Program.cs

```
namespace HelloWorldWeb
{
    public class Program
    {
        public static void Main(string[] args)
        {
            BuildWebHost(args).Run();
        }

        public static IWebHost BuildWebHost(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>()
                .Build();
    }
}
```



Εικόνα 2 - ΤΡΕΧΑ !

Όπως μπορείτε να δείτε παραπάνω, η μέθοδος Main () καλεί τη μέθοδο έκφρασης BuildWebHost () για να δημιουργήσετε web host με προεπιλεγμένες προεπιλογές.
Η έκφραση BuildWebHost μπορεί επίσης να γραφτεί ως μέθοδος που επιστρέφει το IWebHost όπως φαίνεται παρακάτω.

Το WebHost είναι μια στατική κλάση που μπορεί να χρησιμοποιηθεί για τη δημιουργία μιας παρουσίας των IWebHost και IWebHostBuilder με προεπιλεγμένες επιλογές(defaults). Η μέθοδος CreateDefaultBuilder () δημιουργεί μια νέα παρουσία του WebHostBuilder με προεπιλεγμένες επιλογές. Εσωτερικά, διαμορφώνει Kestrel, IISIntegration και άλλες διαμορφώσεις. Ακολουθεί η μέθοδος CreateDefaultBuilder ().

Βιβλιογραφία

https://en.wikipedia.org/wiki/ASP.NET_Core

Programming ASP.NET Core, First Edition by Dino Esposito του Dino Esposito

ASP.NET Core in Action του Andrew Lock

Κεφάλαιο 2 - Το πρώτο Project στην ASP.NET Core

Η ASP.NET Core είναι το μοντέλο εφαρμογών διαδικτυακού προσανατολισμού που λειτουργεί στην κορυφή της πλατφόρμας .NET Core. Παρόλο που το όνομα του μοντέλου εφαρμογής περιέχει το παλιό γνωστό πρόγραμμα αναπαραγωγής ASP.NET, τίποτα στην ASP.NET Core δεν είναι το ίδιο όπως στην προηγούμενη έκδοση της ASP.NET. [1]

Πρώτα απ 'όλα, η ASP.NET Core διαθέτει ένα ολοκαίνουργιο περιβάλλον χρόνου εκτέλεσης που υποστηρίζει ένα μόνο μοντέλο εφαρμογής την ASP.NET MVC. Αυτό σημαίνει ότι το νέο πλαίσιο ιστού (web framework) δεν έχει τίποτα κοινό με τις φόρμες ιστού και απολύτως καμία σχέση με το Web API.

Όλα είναι ολοκαίνουργια , λίγος μόνο κώδικας επαναχρησιμοποιείται από την ASP.NET. Ο σκοπός ήταν να υπάρξει ένα νέο σύστημα προγραμματισμού βασισμένο στην ASP.NET, αλλά πολύ πιο λειτουργικό.

Στον τομέα του μοντέλου προγραμματισμού της ASP.NET MVC, χρησιμοποιούνται κάποιοι εκτελεστές όπως ελεγκτές, προβολές και διαδρομές.

Ακολουθούν τα πρώτα Project της ASP.NET Core.

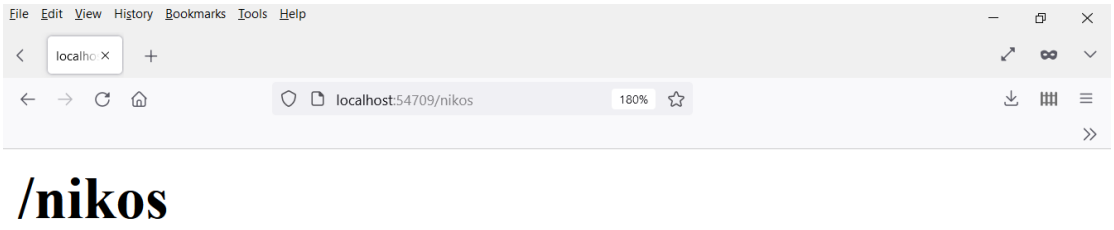
Echo

Program.cs

```
public class Program
   {
      public static void Main(string[] args)
      {
                var host = new WebHostBuilder()
             .UseKestrel()
              .UseIISIntegration()
            .Configure(app => {
          app.Run(async (context) => {
                var path = context.Request.Path;
                await context.Response.WriteAsync("<h1>" + path + "</h1>");
          });
              })
            .Build();
          host.Run();
      }
   }
```

Απλά τυπώνει το path του προγράμματος στην ιστοσελίδα.

Εδώ για παράδειγμα αν μπει η διεύθυνση /nikos τότε θα εμφανιστεί και στην οθόνη.



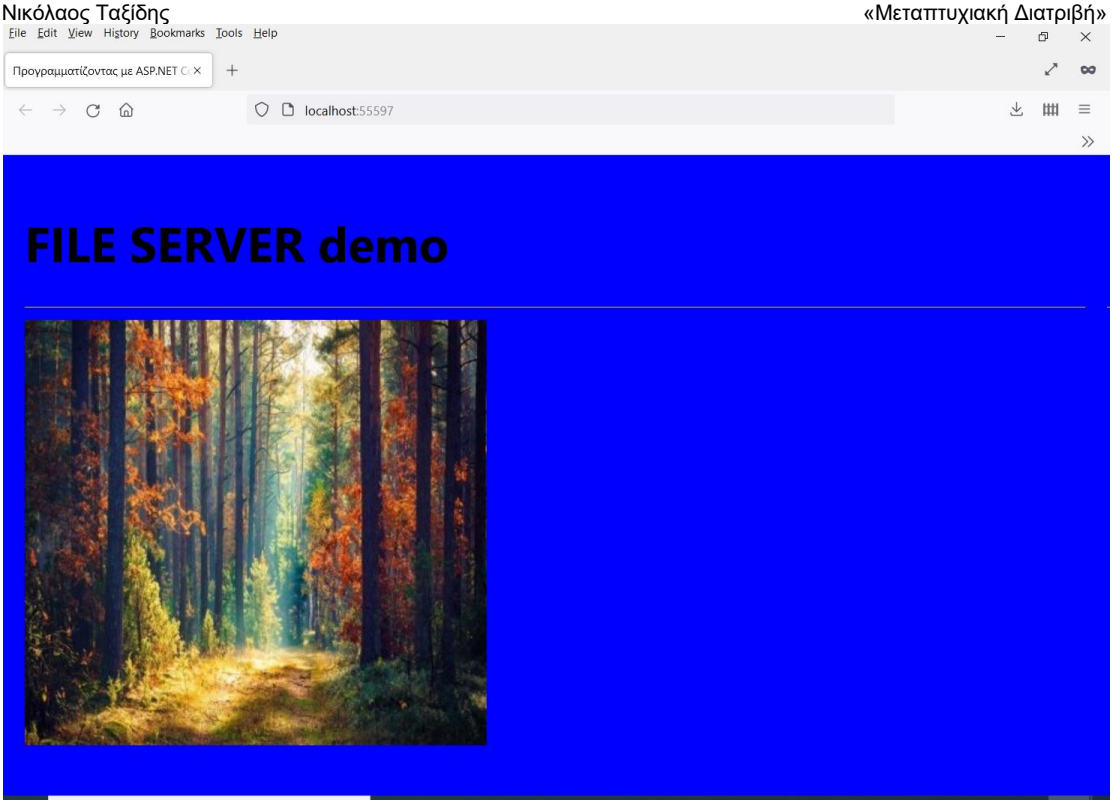
Εικόνα 3- διεύθυνση /nikos

Fileserver

Startup.cs

```
namespace Ch02.FileServer
{
    public class Startup
    {
        public void Configure(IApplicationBuilder app)
    {
      app.Use(async (context, nextMiddleware) =>
      {
        // πρώτο πέρασμα εδώ
        var isMobile = context.IsMobileDevice();
        context.Items["Mobile"] = isMobile;

        if (isMobile)
        {// Αν είναι κινητό
          await context.Response.WriteAsync("ΑΝΙΧΝΕΥΤΗΚΕ ΚΙΝΗΤΗ ΣΥΣΚΕΥΗ!");
          return;
        }

        await nextMiddleware();  // // proairetiko bhma

        // δεύτερο πέρασμα εδώ
      });

      // Ενεργοποίηση προβολής αρχείων από το διαμορφωμένο φάκελο ρίζας ιστού
(web root) (δηλαδή, WWWROOT)
      app.UseDefaultFiles();
      app.UseStaticFiles();

      // Ενεργοποίηση προβολής αρχείων από το φάκελο \Assets που βρίσκονται κάτω
από τον ριζικό φάκελο του ιστοτόπου
      app.UseStaticFiles(new StaticFileOptions()
      {
        FileProvider = new PhysicalFileProvider(
          Path.Combine(Directory.GetCurrentDirectory(), @"Assets")),
        RequestPath = new PathString("/Public/MyAssets")
      });

      app.Run(async (context) =>
      {
        await context.Response.WriteAsync(context.Items["Mobile"].ToString());
      });
    }
  }
```

Εμφανίζει το /Public/MyAssets στην ιστοσελίδα.
Η αρχική σελίδα είναι παρακάτω, αλλά βρίσκεται και έτσι: http://localhost:55597/index.html

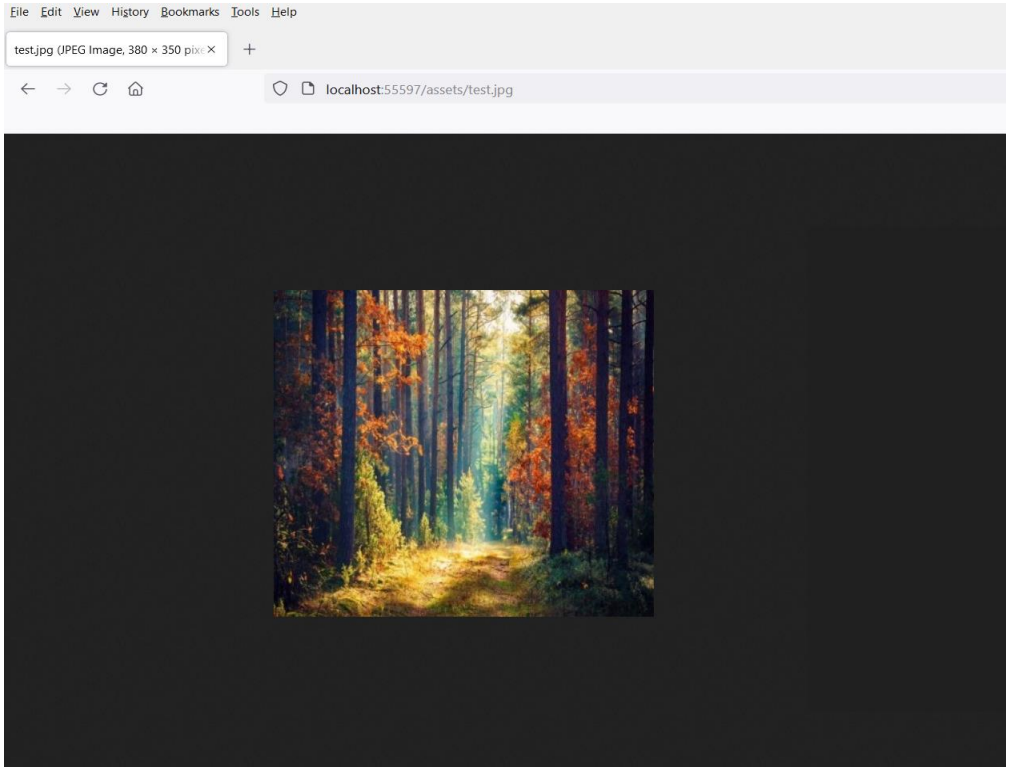


Εικόνα 4 – FILE SERVER demo

MobileHelpers.cs

```
namespace Ch02.FileServer.Common
{
    public static class MobileHelpers
    {// Αυτή η κλάση χρησιμοποιείται όταν τρέχει η εφαρμογή σε κινητό.

        public static bool IsMobileDevice(this HttpContext context)
        {
            var mobileCheck = new Regex(

@"android|(android|bb\d+|meego).+mobile|avantgo|bada\/|blackberry|blazer|compal
|elaine|fennec|hiptop|iemobile|ip(hone|od)|iris|kindle|lge
|maemo|midp|mmp|mobile.+firefox|netfront|opera m(ob|in)i|palm(
os)?|phone|p(ixi|re)\/|plucker|pocket|psp|series(4|6)0|symbian|treo|up\.(browse
r|link)|vodafone|wap|windows (ce|phone)|xda|xiino",
                RegexOptions.IgnoreCase | RegexOptions.Multiline |
RegexOptions.Compiled);

            var ua = context.Request.Headers["user-agent"].ToString();
            if (mobileCheck.IsMatch(ua))
                    return true;
            return false;
        }
    }
}
```

Η διεύθυνση http://localhost:55597/assets/test.jpg εμφανίζει την εικόνα test.jpg που βρίσκεται στον φάκελο \Assets.



Εικόνα 5 – ΕΙΚΟΝΑ ΔΑΣΟΥΣ

Η index.html:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Προγραμματίζοντας με ASP.NET Core -- Ch02</title>
    <link rel="stylesheet" href="/css/site.css" />
</head>
<body>
    <h1>FILE SERVER demo</h1>
    <hr />
  <!-- <img alt="test" src="/Public/MyAssets/test.jpg" /> -->
       <img alt="test" src="Public/MyAssets/test.jpg" />

</body>
       </html>
```

MiniWeb

Startup.cs

```
 public class Startup
  {
    // Χρησιμοποιήστε αυτήν τη μέθοδο για να προσθέσετε υπηρεσίες στο container συστήματος ASP.NET.
    public void ConfigureServices(IServiceCollection services)
    {
      // Περάστε το δικό σας παράδειγμα .
      services.AddSingleton<ICountryRepository>(new CountryRepository());
    }

    public void Configure(IApplicationBuilder app,
      IHostingEnvironment env,
      IServiceProvider provider)
    {
      app.Map("/country", countryApp =>
      {
        countryApp.Run(async (context) =>
        {
          var country = provider.GetService<ICountryRepository>();
          var query = context.Request.Query["q"];
          var list = country.AllBy(query).ToList();
          var json = JsonConvert.SerializeObject(list);

          await context.Response.WriteAsync(json);
        });
      });

//Δοκιμάστε αυτόν τον κώδικα για να το δείτε χωρίς την εφαρμογή app.Map , δίνεται  η ίδια έξοδος για κάθε διεύθυνση URL

            app.Run(async (context) =>
            {
              var country = provider.GetService<ICountryRepository>();
               var query = context.Request.Query["q"];
               var list = country.AllBy(query).ToList();
                var json = JsonConvert.SerializeObject(list);

               await context.Response.WriteAsync(json);
            });
        }
    }
}
```

Απλά εμφανίζει την λίστα χωρών Country από τον αποθηκευτικό χώρο ICountryRepository στην ιστοσελίδα:

Continent.cs

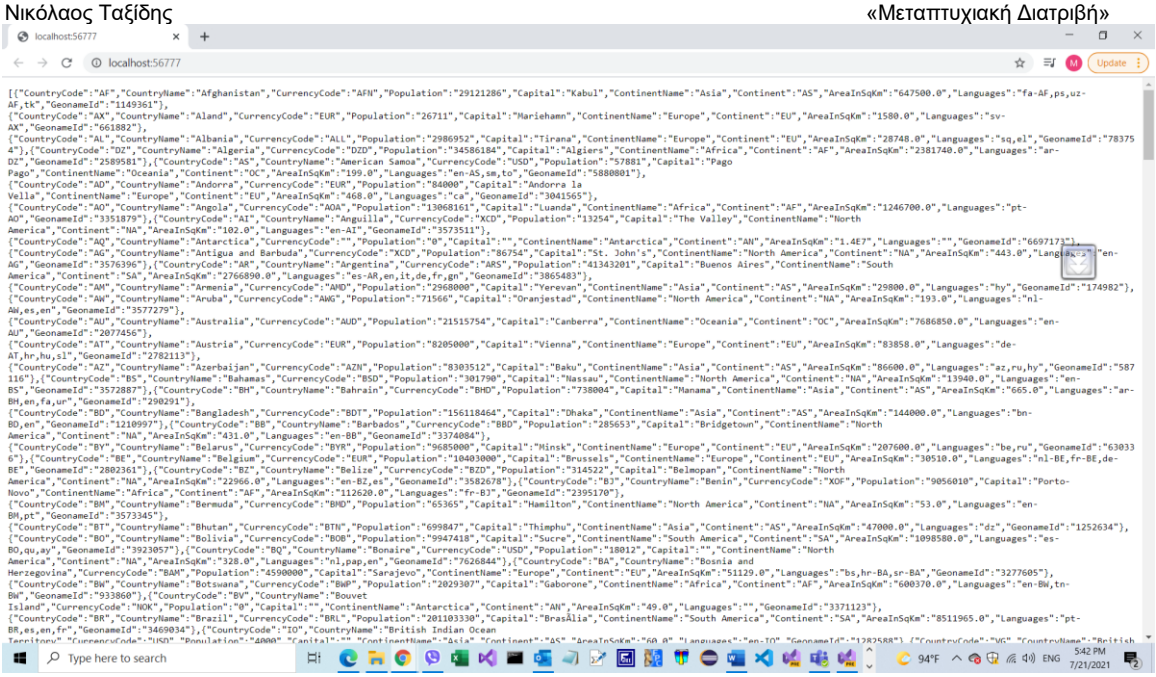
```
using System.ComponentModel;

namespace Ch02.MiniWeb.Persistence.Model
{
    public enum Continent
    {// Εδώ παρατίθενται οι ήπειροι.
        Unknown = 0,
```

```
        Europe = 1,
        Asia = 2,
        [Description("North America")]
        NorthAmerica = 3,
        [Description("South America")]
        SouthAmerica = 4,
        Africa = 5,
        Oceania = 6,
        Antarctica = 7
    }
}
```

Country.cs

```
namespace Ch02.MiniWeb.Persistence.Model
{
    // Δηλώθηκε ως STRUCT για αποφυγή επικάλυψης memrefs στο LINQ


    public partial struct Country
    {//Τα στοιχεία κάθε χώρας.
        public string CountryCode { get; set; }
        public string CountryName { get; set; }
        public string CurrencyCode { get; set; }
        public string Population { get; set; }
        public string Capital { get; set; }
        public string ContinentName { get; set; }
        public string Continent { get; set; }
        public string AreaInSqKm { get; set; }
        public string Languages { get; set; }
        public string GeonameId { get; set; }
    }
}
```

Εμφανίζονται όλες οι χώρες με τον κωδικό ονομασίας τους, το όνομα τους, τον κωδικό του νομίσματος που χρησιμοποιούν , τον αριθμό πληθυσμού τους, την πρωτεύουσα, την ήπειρο που ανήκουν , τον κωδικό της ηπείρου, την έκταση σε τετραγωνικά χιλιόμετρα, τις ομιλούμενες γλώσσες και τον γεωγραφικό κωδικό τους.



Εικόνα 6 - Χώρες

Βιβλιογραφία

Programming ASP.NET Core, First Edition by Dino Esposito του Dino Esposito

ASP.NET Core in Action του Andrew Lock

Κεφάλαιο 3 - Bootstrapping στην ASP.NET MVC

Η ASP.NET Core υποστηρίζει πλήρως το συγκεκριμένο μοντέλο εφαρμογής ASP.NET Model-View-Controller (MVC) στο οποίο η διεύθυνση URL μιας εισερχόμενης αίτησης επιλύεται σε ένα ζευγάρι χειριστηρίου / ενεργειών στοιχείων . Το στοιχείο ελεγκτή(*controller*) προσδιορίζει ένα όνομα κλάσης, το στοιχείο ενέργειας προσδιορίζει μια μέθοδο στην κλάση ελεγκτή.
Η επεξεργασία της αίτησης, επομένως, είναι θέμα εκτέλεσης της δεδομένης μεθόδου δράσης της δεδομένης κλάσης ελεγκτή.

Το μοντέλο εφαρμογής ASP.NET MVC στην ASP.NET Core είναι σχεδόν πανομοιότυπο με το μοντέλο εφαρμογής MVC που διατίθεται στην κλασική ASP.NET και δεν διαφέρει πολύ από τις εφαρμογές του ίδιου μοτίβου MVC που βρίσκονται σε άλλες πλατφόρμες ιστού όπως η CakePHP για PHP, η Rails για Ruby και η Django για Python. Το μοτίβο MVC είναι επίσης αρκετά δημοφιλές μεταξύ των πλαισίων front-end, κυρίως Angular και KnockoutJS.

Σε αυτό το κεφάλαιο, θα δειχθούν τα προκαταρκτικά βήματα που τελικά θα ρυθμίσουν τον αγωγό ASP.NET MVC Core και θα δειχθεί ο υπεύθυνος χειριστής για την πραγματική επεξεργασία τυχόν εισερχόμενων αιτημάτων.

Routes

Εδώ παρουσιάζεται η αρχική κλάση:

Startup.cs

```
public class Startup
   {
      public void ConfigureServices(IServiceCollection services)
      {
         services.AddMvc();

      }

      public void Configure(IApplicationBuilder app)
      {
         app.UseMvcWithDefaultRoute();
         app.Run(async (context) =>
         {
            await context.Response.WriteAsync(
              " Θα έλεγα ότι δεν υπάρχουν διαμορφωμένες δρομολογήσεις εδώ!");
         }); //Μήνυμα λάθους
      }
   }
```

Οι κάτωθι ελεγκτές χρησιμοποιούνται:

DateController.cs

```
public class DateController : Controller
   {
      public IActionResult Day(int offset)
      {
         var day = DateTime
            .Now
            .Date
            .AddDays(150 + offset)//Πρόσθεση 150 ημερών
            .ToString("ddd, d MMM yyyy"); //Η ημερομηνία
         return new ContentResult { Content = $"{day}" };
      }//Δείχνει την ημερομηνία 150 ημέρες από την σημερινή.
   }
```

HomeController.cs

```
public class HomeController : Controller
   {
      public IActionResult Index()
      {
         var controller = RouteData.Values["controller"];
         var action = RouteData.Values["action"];
         var catchall = RouteData.DataTokens["reason"] ?? "";

         var text = string.Format("{0}.{1} {2}", controller, action, catchall);
```

```
        var ar = new ContentResult { Content = text };
        return ar;
    }

    [Route("demo/{code}")]
    public IActionResult Read(
        Container container,
        int number,
        [FromQuery] int code,
        string headeracceptlanguage,
        [FromHeader(Name ="Accept-Language")] string language)
    {
        var lang = Request.Headers["Accept-Language"];
        return new ContentResult { Content = container.Number.ToString() };
    }
}
```

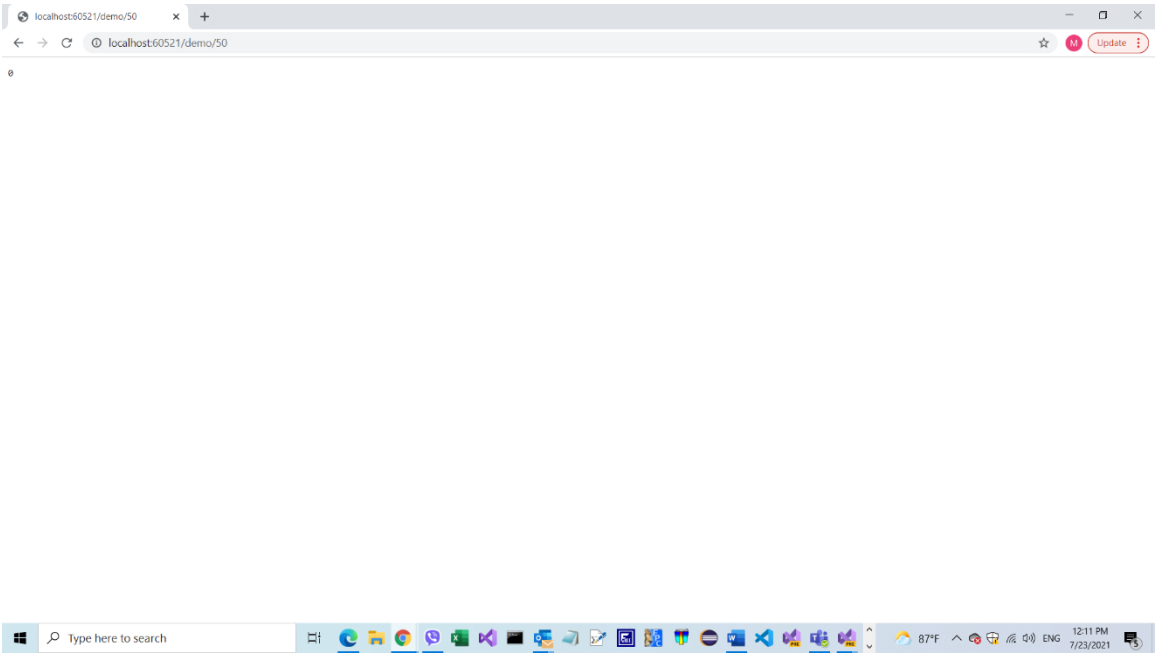
Container.cs

```
public class Container
{
    [BindNever]
    public int Number { get; set; }

    public int Code { get; set; }
}
```

Εκτελώντας τη διαδρομή `/date/day` έχουμε την εικόνα παρακάτω, αφού η μέρα δοκιμής ήτανε η 8/8/2021, δείχνει την ημερομηνία 150 ημέρες μετά:

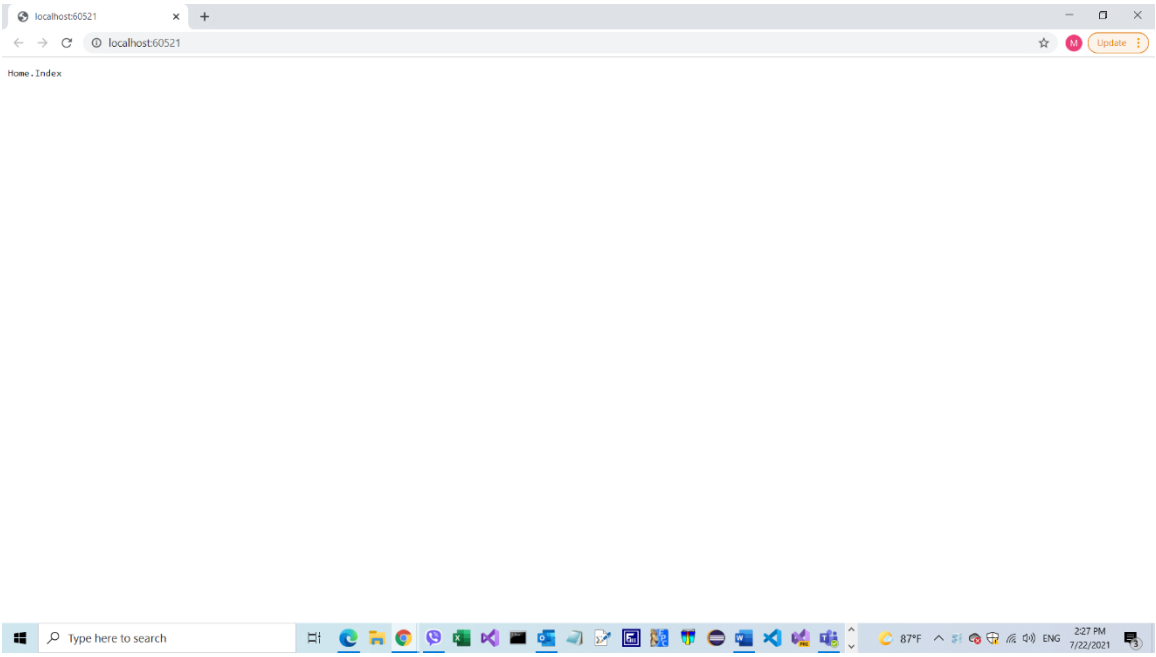
Εικόνα 7 – Διαδρομή /date/day

Εκτελώντας τη διαδρομή /demo/50 έχουμε την εικόνα παρακάτω:



Εικόνα 8 – Διαδρομή /demo/50

Το index εμφανίζει:



Εικόνα 9 – index

RoutesEx

Η κλάση startup ακολουθεί:

Startup.cs

```
public class Startup
   {
      public void ConfigureServices(IServiceCollection services)
      {
         var builder = services.AddMvcCore();
         builder.AddViews();
         builder.AddRazorViewEngine();

         // DEMO:
         services.Configure<RouteOptions>(options =>
               options.ConstraintMap.Add("your", typeof(YourRouteConstraint)));
      }

      public void Configure(IApplicationBuilder app, IHostingEnvironment env)
      {
         app.UseMvc(routes =>
         {
            routes.MapRoute(name: "route-today",
               template: "today/{offset}",
               defaults: new { controller = "date", action = "day", offset = 0 },
               constraints: new { offset = new IntRouteConstraint() });
            routes.MapRoute(name: "route-yesterday",
               template: "yesterday",
               defaults: new { controller = "date", action = "day", offset = -1 });
            routes.MapRoute(name: "route-tomorrow",
               template: "tomorrow",
               defaults: new { controller = "date", action = "day", offset = 1 });
            routes.MapRoute(name: "route-day",
               template: "date/day/{offset:int}",
               defaults: new { controller = "date", action = "day", offset = 0 });
         });
         app.UseMvc(routes =>
         {
            routes.MapRoute(name: "catch-all",
               template: "{*url}",
               defaults: new { controller = "home", action = "index" },
               constraints: new { },
               dataTokens: new { reason = "catch-all" });
         });
         app.Run(async (context) =>
         { //Εμφάνιση μηνύματος.
            await context.Response.WriteAsync("Geia Kosme !!!!");
         });
      }
   }
```

Οι κάτωθι ελεγκτές χρησιμοποιούνται:

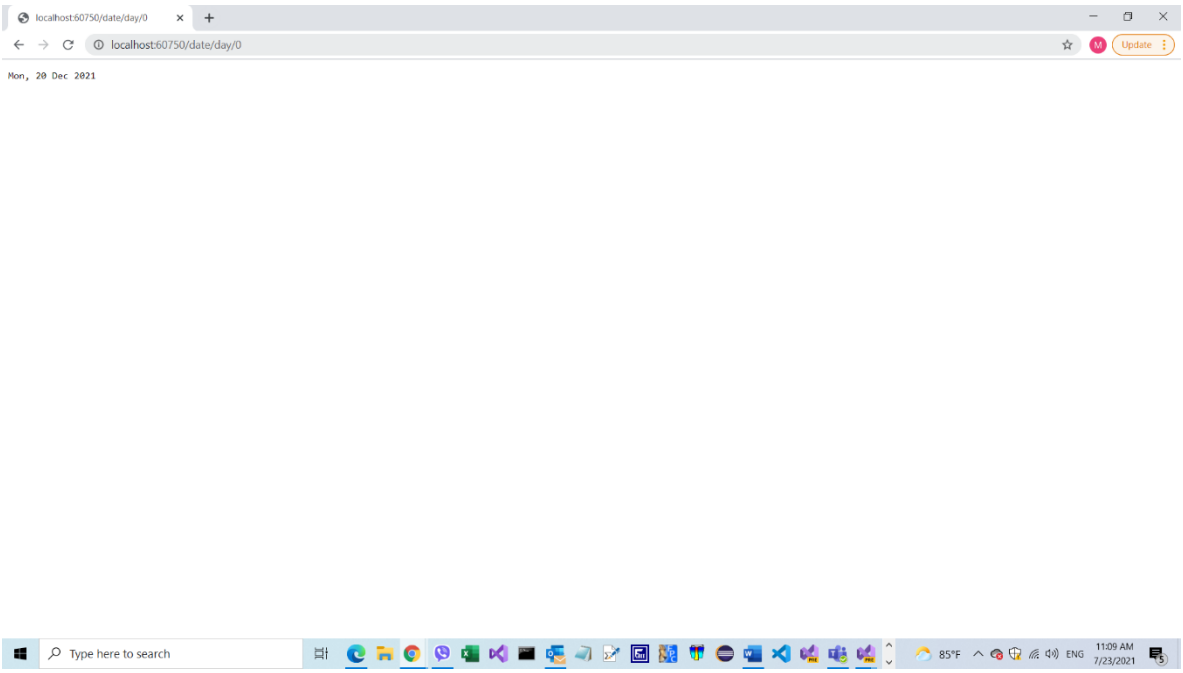
DateController.cs

```
public class DateController : Controller
```

```
  {
    public IActionResult Day(int offset)
    {
      var day = DateTime
        .Now
        .Date
        .AddDays(150 + offset)//Πρόσθεση των 150 επιπλέον ημερών.
        .ToString("ddd, d MMM yyyy");
      return new ContentResult { Content = $"{day}" };
    }
  }
```

Άρα αν η σημερινή ημερομηνία είναι η Τρίτη, 23 Ιουλίου 2021, 150 μέρες μετά είναι η Δευτέρα 20 Δεκεμβρίου 2021 η οποία θεωρείται και η ημέρα 0.

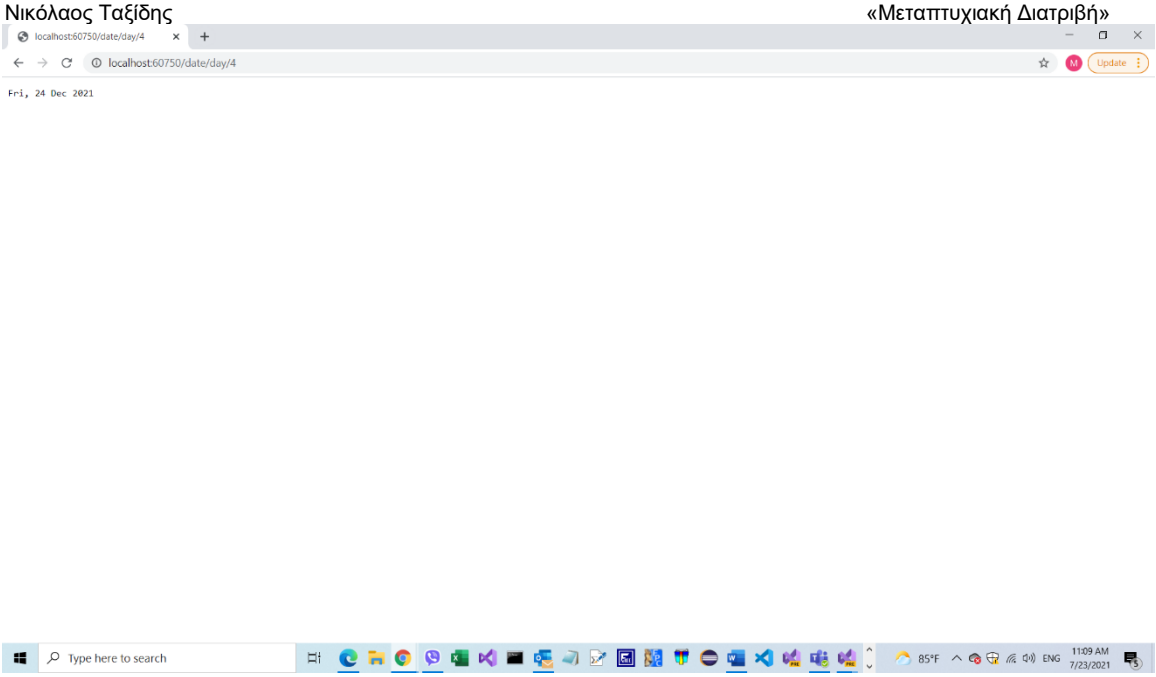
Εκτελώντας τη διαδρομή /date/day/0 έχουμε την εικόνα παρακάτω:



Εικόνα 10 – Διαδρομή /date/day/0

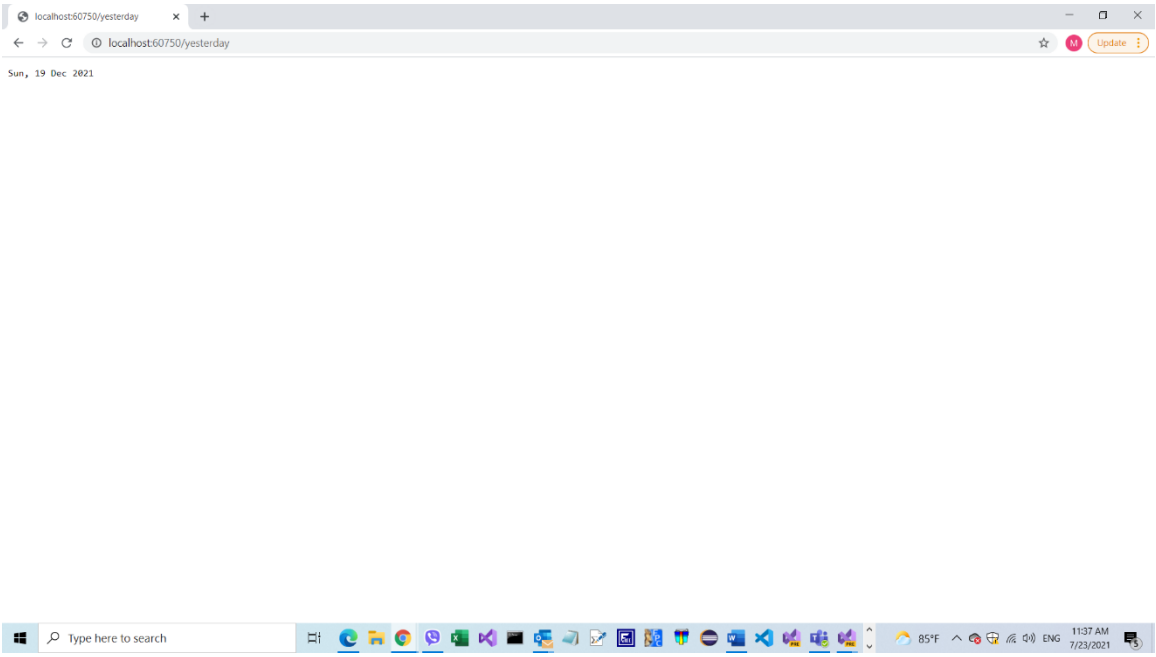
Αν υπολογίσουμε 4 ημέρες μετά την ημέρα 0 (20/12/2021) έχουμε:

Εκτελώντας τη διαδρομή /date/day/4 έχουμε την εικόνα παρακάτω:



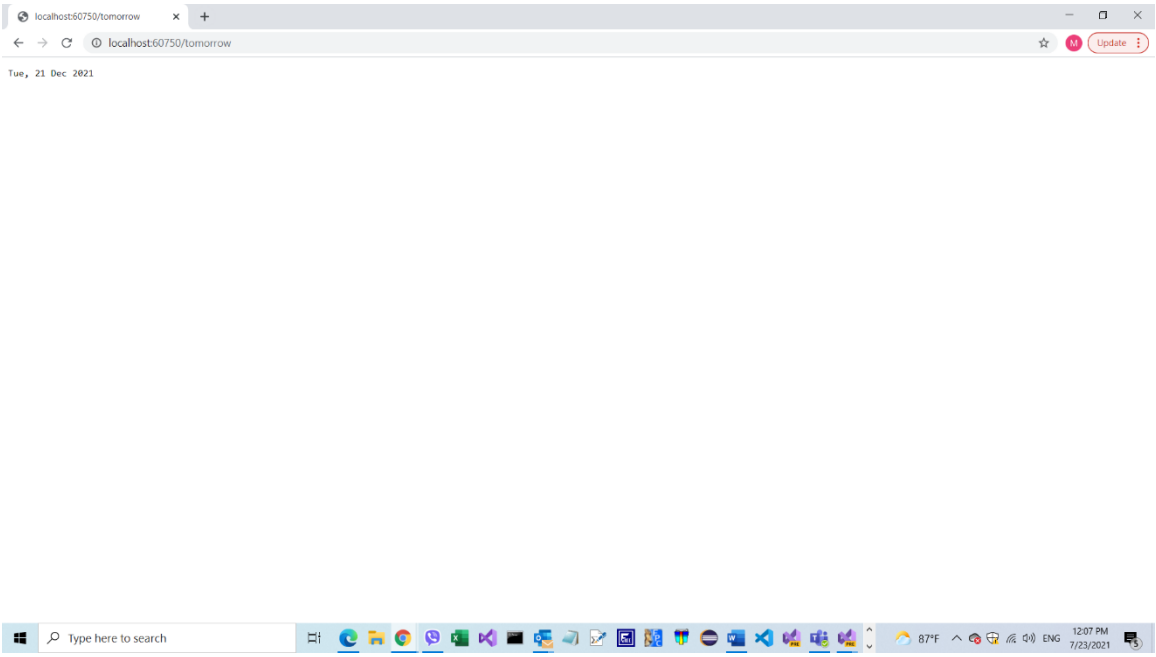
Εικόνα 11 - Διαδρομή /date/day/4

Εκτελώντας τη διαδρομή /yesterday έχουμε την εικόνα παρακάτω:



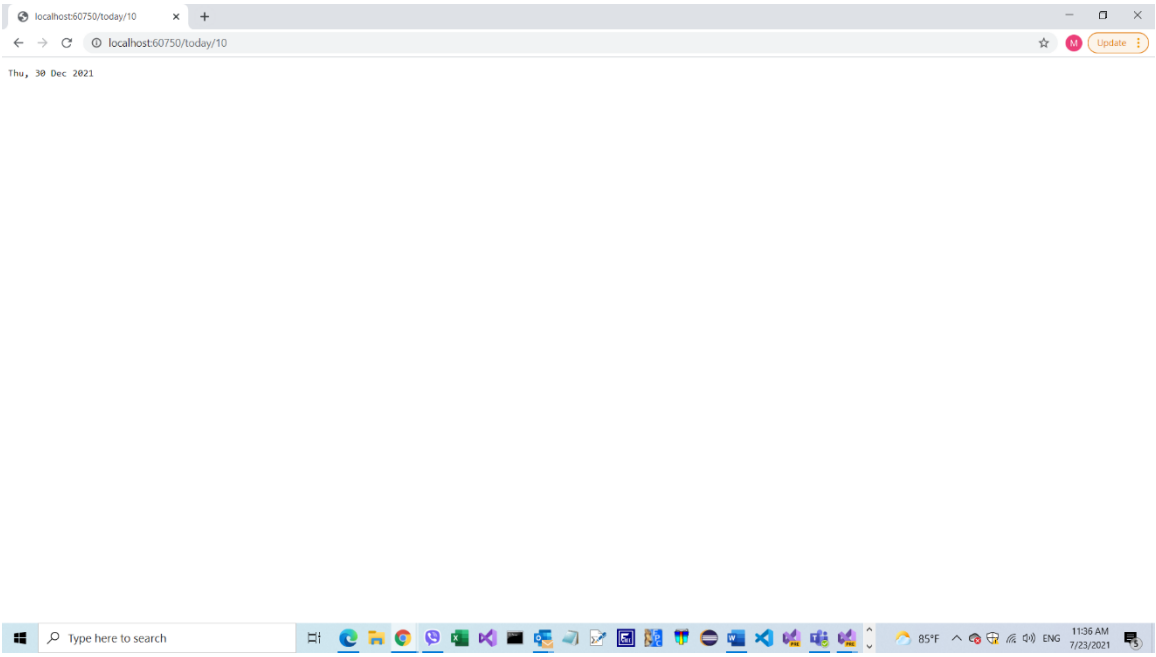
Εικόνα 12 - Διαδρομή /yesterday

Εκτελώντας τη διαδρομή /tomorrow έχουμε την εικόνα παρακάτω:



Εικόνα 13 - Διαδρομή /tomorrow

Εκτελώντας τη διαδρομή /today/10 έχουμε την εικόνα παρακάτω:

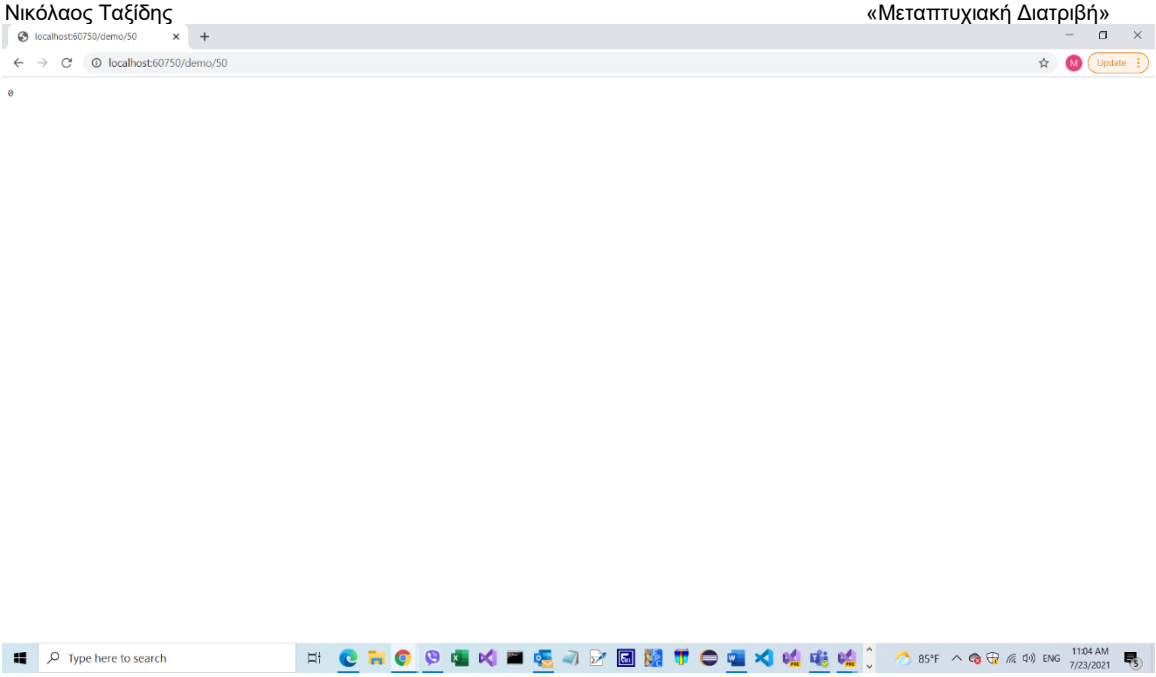


Εικόνα 14 - Διαδρομή /today/10

HomeController.cs

```
public class HomeController : Controller
   {
      public IActionResult Index()
      {
         var controller = RouteData.Values["controller"];
         var action = RouteData.Values["action"];
         var catchall = RouteData.DataTokens["reason"] ?? "";

         var text = string.Format("{0}.{1} {2}", controller, action, catchall);
         var ar = new ContentResult { Content = text };
         return ar;
      }

      [Route("demo/{code}")]
      public IActionResult Read(
         Container container,
         int number,
         [FromQuery] int code,
         string headeracceptlanguage,
         [FromHeader] string language)
      {
         var lang = Request.Headers["Accept-Language"];
         return new ContentResult { Content = container.Number.ToString() };
      }
   }

   public class Container
   {
      [BindNever]
      public int Number { get; set; }

      public int Code { get; set; }
   }
```

Εκτελώντας τη διαδρομή /demo/50 έχουμε την εικόνα παρακάτω:

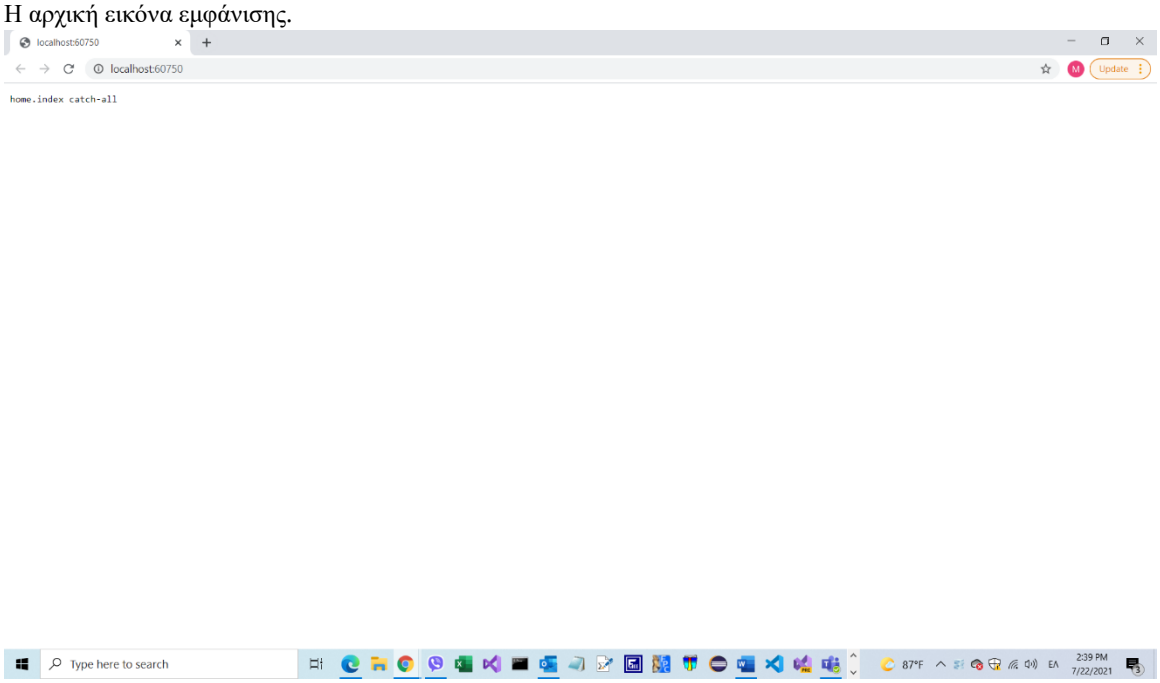


Εικόνα 15 - Διαδρομή /demo/50

YourRouteConstraint.cs

```
public class YourRouteConstraint : IRouteConstraint
   {
      public bool Match(HttpContext httpContext, IRouter route, string routeKey,
         RouteValueDictionary values, RouteDirection routeDirection)
      {
         throw new System.NotImplementedException();
      }
   }
```

Η αρχική εικόνα εμφάνισης.



Εικόνα 16 – Αρχική Διαδρομή

Βιβλιογραφία

https://en.wikipedia.org/wiki/Bootstrapping_(compilers)

Programming ASP.NET Core, First Edition by Dino Esposito του Dino Esposito

ASP.NET Core in Action του Andrew Lock

Κεφάλαιο 4 - Ελεγκτές(Controllers) στην ASP.NET MVC

Παρά τη ρητή αναφορά στο μοντέλο Model-View-Controller στο όνομα, το μοντέλο εφαρμογής ASP.NET MVC επικεντρώνεται ουσιαστικά σε έναν πυλώνα - τον ελεγκτή(Controller).

Ο ελεγκτής διέπει ολόκληρη την επεξεργασία μιας αίτησης. Καταγράφει δεδομένα εισόδου, ενορχηστρώνει τη δραστηριότητα των επιπέδων επιχειρήσεων και δεδομένων και, τέλος, ολοκληρώνει τα ανεπεξέργαστα δεδομένα που υπολογίστηκαν για το αίτημα σε μια έγκυρη απόκριση για τον καλούντα.

Κάθε αίτημα που περνά το φίλτρο δρομολόγησης URL αντιστοιχίζεται σε μια τάξη ελεγκτή και εξυπηρετείται εκτελώντας μια δεδομένη μέθοδο στην τάξη.

Επομένως, η τάξη ελεγκτή είναι το μέρος όπου οι προγραμματιστές γράφουν τον πραγματικό κωδικό που απαιτείται για την εξυπηρέτηση ενός αιτήματος.

Ας διερευνήσουμε εν συντομία ορισμένα χαρακτηριστικά των τάξεων ελεγκτών, συμπεριλαμβανομένων των λεπτομερειών εφαρμογής.

AttrRoutingΤο πρόγραμμα αποτελείται κυρίως από τους τρεις ελεγκτές :

InputController.cs

Η δράση echo επιστρέφει τα data στην ιστοσελίδα.

```
public IActionResult Echo()
        {
            var dedom = Request.Query["today"];
            return Ok(data);
        }
```

Η δράση Go() γράφει τα στοιχεία στην πόλη και στις μέρες στην σελίδα:

```
public IActionResult Go()
        {
            // Λήψη δεδομένων με μη αυτόματο τρόπο από το πρότυπο URL
            var polis = RouteData.Values["city"];
            var imeres = RouteData.Values["days"];
            return Ok(string.Format("Στην {0} για {1} ημέρες", polis, imeres));
        }
```

TourController.cs

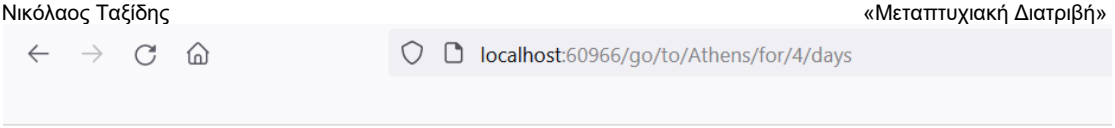
……

```
Δοκιμάστε:
//http://localhost:60966/go/to/Patra

//http://localhost:60966/go/to/Patra/for/5/days

//http://localhost:60966/go/to/Athens/for/4/days
```

Η δράση NewYork() επιστρέφει το όνομα της στην ιστοσελίδα.

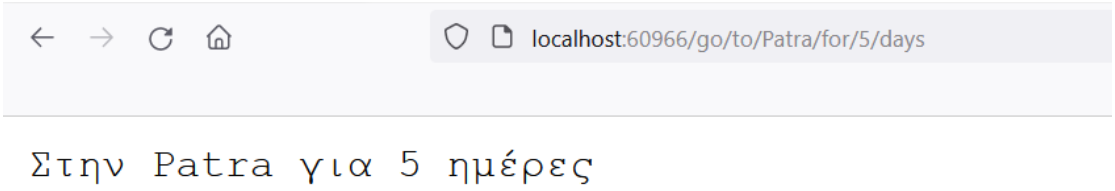
```
[Route("[controller]/[action]")]
        [ActionName("ath")]
        public IActionResult Athens()
        {
            var action = RouteData.Values["action"].ToString();
            return Ok(drasi);
        }
```

Επίσης κάθε μια από τις παρακάτω δράσεις επιστρέφει το όνομα της στην ιστοσελίδα:



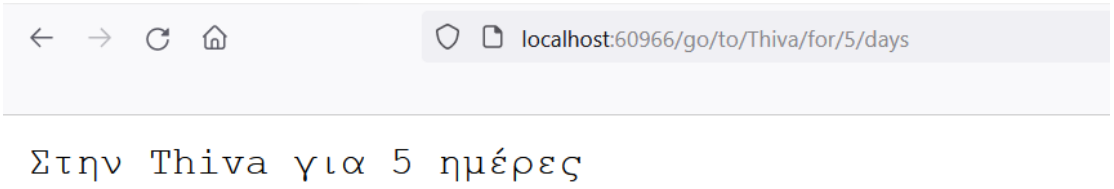
Εικόνα 17- Athens 4 ημέρες

```
public IActionResult Patra()
        {
            var drasi = RouteData.Values["action"].ToString();
            return Ok(drasi);
        }
```



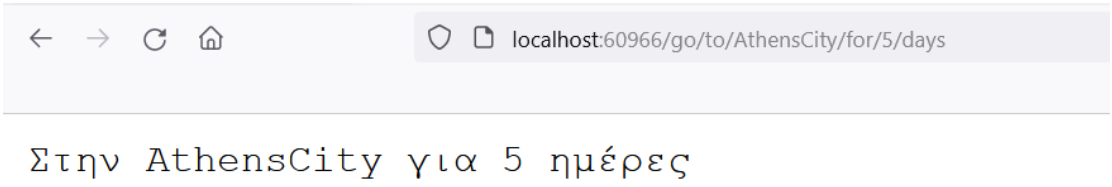
Εικόνα 18 - Patra 5 ημέρες

```
        [NonAction]
        public IActionResult Thiva()
        {
            var drasi = RouteData.Values["action"].ToString();
            return Ok(drasi);
        }
```



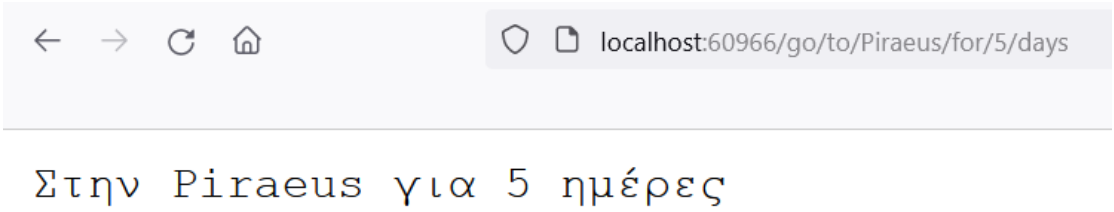
Εικόνα 19- Thiva 5 ημέρες

```
       [Route("aths")]
        public IActionResult AthensCity()
        {
            var drasi = RouteData.Values["action"].ToString();
            return Ok(drasi);
        }
```



Εικόνα 20- AthensCity 5 ημέρες

```
[Route("/ath")]
public IActionResult Piraeus()
{
    var action = RouteData.Values["action"].ToString();
    return Ok(drasi);
}
```

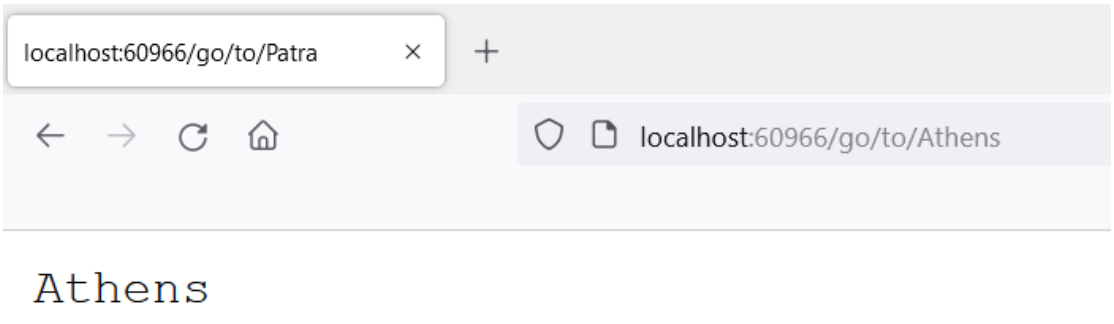


Εικόνα 21- Piraeus 5 ημέρες

VipTourController.cs

Η δράση Athens() επιστρέφει το όνομα της στην ιστοσελίδα.

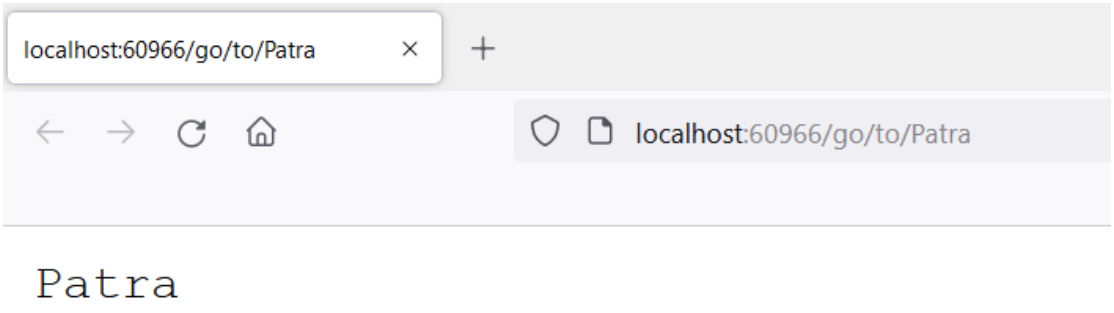
```
public IActionResult Athens()
    {
       var action = RouteData.Values["action"].ToString();
       return Ok(drasi);
    }
```



Εικόνα 22- Athens

Επίσης κάθε μια από τις παρακάτω δράσεις επιστρέφει το όνομα της στην ιστοσελίδα:

```
public IActionResult Patra()
{
   var action = RouteData.Values["action"].ToString();
   return Ok(drasi);
}
```

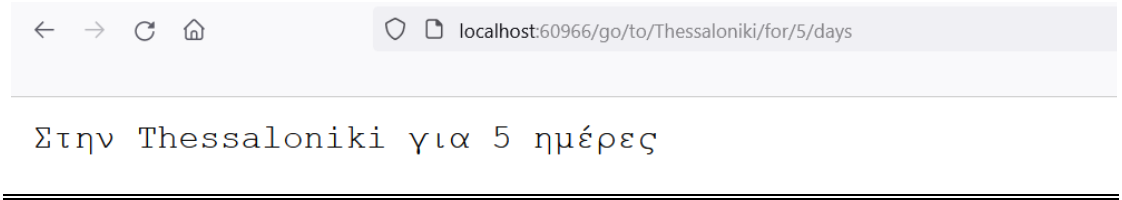


Εικόνα 23- Patra

Εδώ επιπλέον επιστρέφει στη Θεσσαλονίκη για όσες μέρες ζητείται.

```
[Route("for/{days}/days")]
```

```
public IActionResult Thessaloniki (int days)
{
    var drasi  = string.Format("Στην {0} για {1} ημέρες",
        RouteData.Values["action"].ToString(),
        days);
    return Ok(drasi);
}
```



Εικόνα 24- Thessaloniki 5 ημέρες

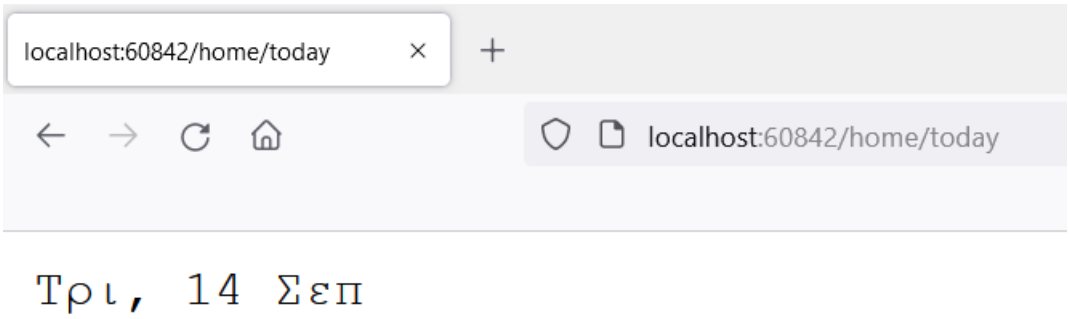
Poco

PocoController.cs

Η δράση Today επιστρέφει την ημερομηνία στην ιστοσελίδα.

```
        public IActionResult Today()
        {
            var pocoobj = new ContentResult {

                Content = DateTime.Now.ToString("ddd, d MMM")

            return pocoobj;

};
```

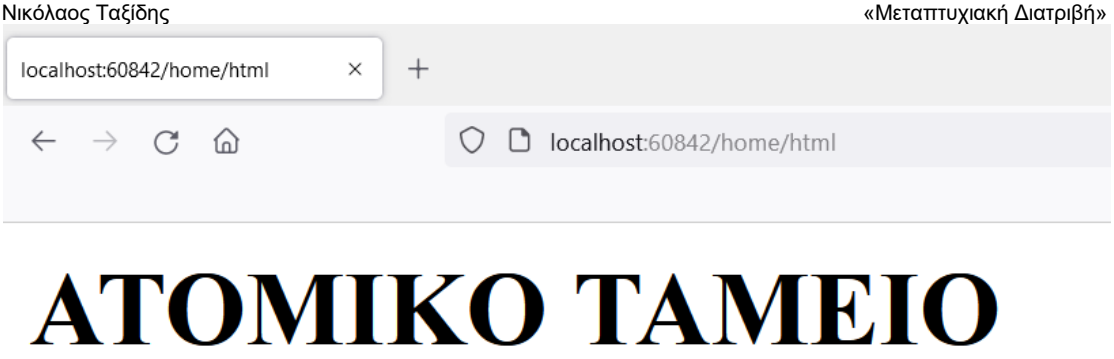
http://localhost:60842/home/today



Εικόνα 25- today, σημερινή ημερομηνία

Η δράση Html() επιστρέφει το κείμενο html " Geia se olous! " στην ιστοσελίδα.

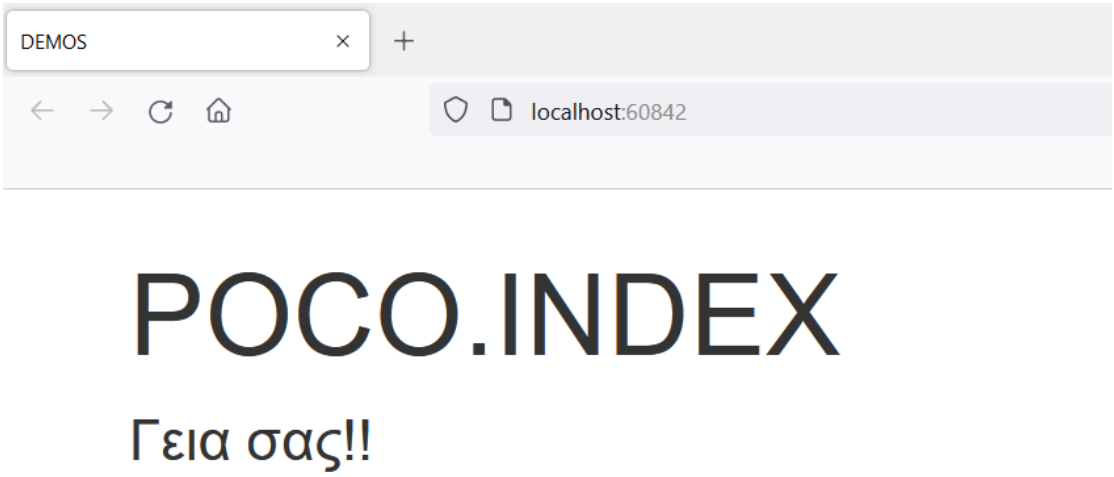
```
public IActionResult Html()
        {
            return new ContentResult()
            {
                Content = "<h1>ATOMIKO TAMEIO</h1>",
                ContentType = "text/html",
                StatusCode = 200
            };
        }
```



Εικόνα 26 – Εμφάνιση ΑΤΟΜΙΚΟ ΤΑΜΕΙΟ

Η δράση index προβάλλει το κείμενο "ΑΤΟΜΙΚΟ ΤΑΜΕΙΟ" στην ιστοσελίδα μέσω του viewdata.Model Myclass.

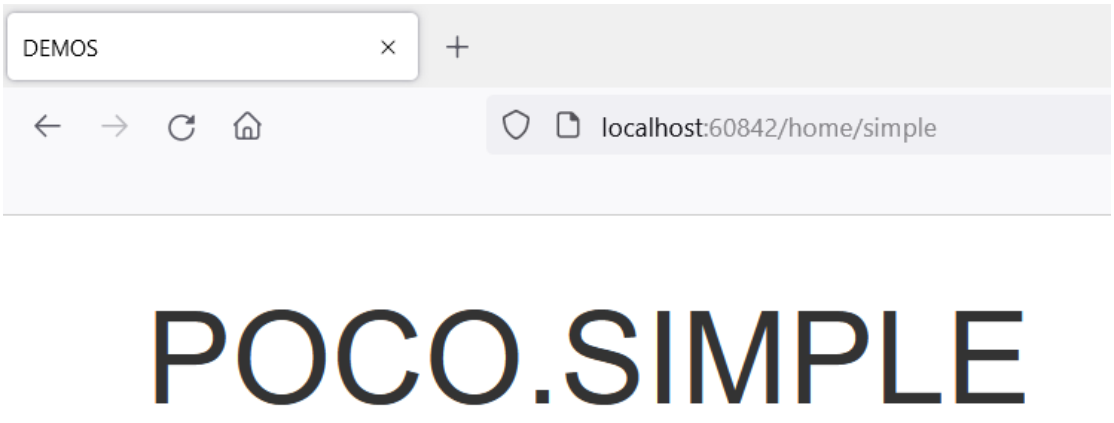
```
public IActionResult Index(
            [FromServices] IModelMetadataProvider provider)
        {
            var pocodata = new ViewDataDictionary<MyClass>(provider, new
ModelStateDictionary());
            pocodata.Model = new MyClass() { Title = " Γεια σας!!" };
            return new ViewResult() { ViewData = pocodata, ViewName = "index"
};
        }
```



Εικόνα 27 – POCO.INDEX - Εμφάνιση Γεια σας!!

Η δράση simple προβάλλει το κείμενο "POCO.Simple" στην ιστοσελίδα μέσω του ViewName.

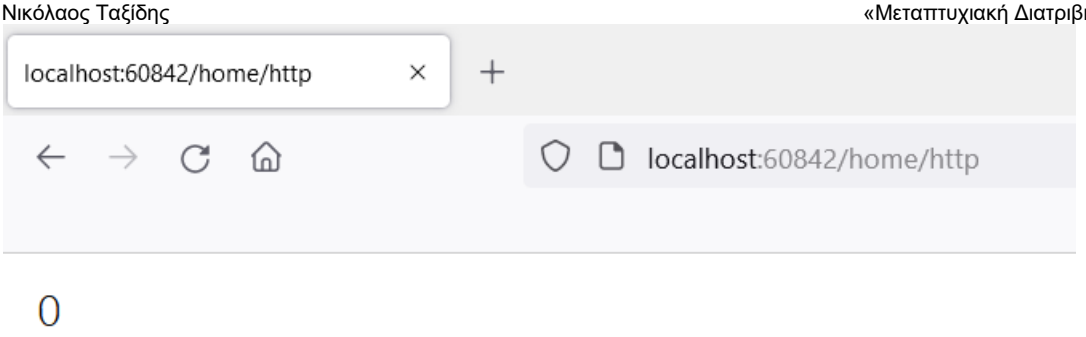
```
public IActionResult Simple()
        {
            return new ViewResult() { ViewName = "simple" };
        }
```



Εικόνα 28 – POCO.SIMPLE

Η δράση Http προβάλλει τον ακέραιο p στην ιστοσελίδα μέσω του Content.

```
        public IActionResult Http([FromQuery] int p = 0)
        {
            var elegktis = Context.RouteData.Values["controller"];
            return new ContentResult() { Content = p.ToString() };
        }
```



Το πρόγραμμα περιλαμβάνει το μοντέλο παρακάτω:

MyClass.cs

```
public class MyClass
    {
        public string Title { get; set; }
    }
```

Η Index.cshtml:

```
@model  Ch04.Poco.Common.MyClass

<h1>POCO.INDEX</h1>
<h4>@Model.Titlos</h4>
```

Η simple.cshtml:

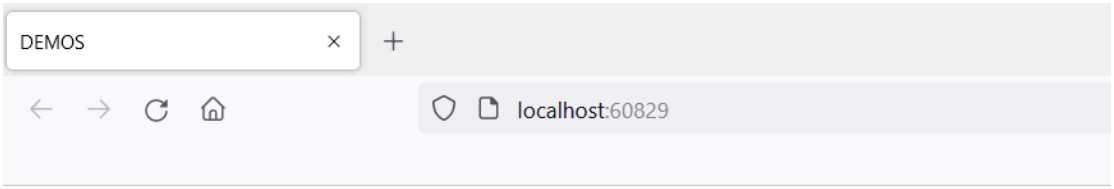
```
<h1>POCO.SIMPLE</h1>
```

Urls: localhost:60482/home/index, localhost:60482/home/simple, Http,Today, Html

HomeController.cs

Η δράση index δείχνει την αρχική ιστοσελίδα.

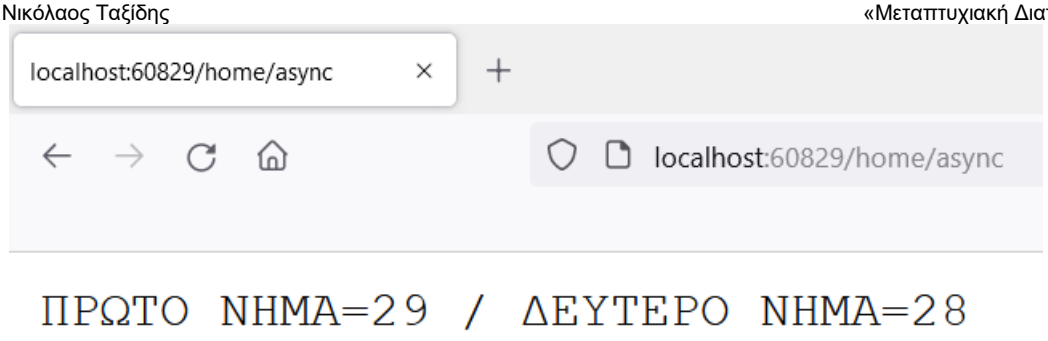
```
public IActionResult Index(string controller)
    {
      return View();
    }
```



Εικόνα 29 – HOME.INDEX

Η δράση async επιστρέφει την CurrentThread.ManagedThreadId στον παρόντα χρόνο .

```
    public async Task<IActionResult> Async()
    {
      var t1 = Thread.CurrentThread.ManagedThreadId.ToString();
      var client = new HttpClient();
      var before = DateTime.Now;

      await client.GetStringAsync("https://www.youtube.com/");

     var after = DateTime.Now;
     var t2 = Thread.CurrentThread.ManagedThreadId.ToString();
     return Content(string.Concat("ΠΡΩΤΟ ΝΗΜΑ=", t1, " / ΔΕΥΤΕΡΟ ΝΗΜΑ=", t2));
    }
```



Εικόνα 30 - ΝΗΜΑΤΑ

FilterController.cs
Το παρακάτω μοντέλο χρησιμοποιείται στη δράση Repeat.

```
  public class RepeatText
    {


public RepeatText()
        {
            Options = new RepeatTextOptions();
        }
        public string Text { get; set; }
        public int Number { get; set; }
        public RepeatTextOptions Options { get; set; }
    }

    public class RepeatTextOptions
    {
        public RepeatTextOptions()
        {
            Factor = 1;
        }
        public int Factor { get; set; }
    }
```

Στην παρακάτω συνάρτηση αν η δράση που εκτελείται είναι η index θέτει `StartTime = DateTime.Now;` την τρέχουσα ώρα και εκτελεί την συνάρτηση της κλάσης βάσης `base.OnActionExecuting(filterContext);`

```
protected DateTime StartTime;

        public override void OnActionExecuting(ActionExecutingContext
filterContext)
        {
            var action = filterContext.ActionDescriptor.RouteValues["action"];
            if (string.Equals(action, "index",
StringComparison.CurrentCultureIgnoreCase))
            {
```

```
                StartTime = DateTime.Now;
            }

            base.OnActionExecuting(filterContext);
        }
```

Η παρακάτω συνάρτηση αν η δράση που εκτελέστηκε είναι η index θέτει var timeSpan = DateTime.Now - StartTime;το διάστημα χρόνου που πέρασε, θέτει τον κατάλληλο header και εκτελεί την συνάρτηση της κλάσης βάσης base.OnActionExecuted(filterContext);

```
public override void OnActionExecuted(ActionExecutedContext filterContext)
        {
            var action = filterContext.ActionDescriptor.RouteValues["action"];
            if (string.Equals(action, "index",
StringComparison.CurrentCultureIgnoreCase))
            {
                var timeSpan = DateTime.Now - StartTime;
                filterContext.HttpContext.Response.Headers.Add(
                        "duration",
timeSpan.TotalMilliseconds.ToString(CultureInfo.InvariantCulture));
            }

            base.OnActionExecuted(filterContext);
        }
```

Η δράση αυτή παρακάτω θέτει το αλφαριθμητικό " Μόλις επεξεργάστηκε το Filter.Index" στο περιεχόμενο της απόκρισης που προβάλλεται στην αντίστοιχη ιστοσελίδα.

```
        public IActionResult Index()
        {
            var output = Content("Μόλις επεξεργάστηκε το Filter.Index");
            return output;
        }
```

Η δράση αυτή παρακάτω κατευθύνει τον χρήστη στην ιστοσελίδα http://www.google.com

```
        public IActionResult Google()
        {
            return Redirect("http://www.google.com");
        }
```

Η δράση αυτή παρακάτω κατευθύνει προς την εκτέλεση της δράσης repeat με δεδομένα στοιχεία του μοντέλου.

```
        public IActionResult GoRepeat()
        {
            var result = RedirectToAction(
                "repeat",
                "binding",
                new {text = "Jino", arithmos = 4});
            return result;
        }
    }
```

BindingController.cs

Η δράση αυτή παρακάτω επιστρέφει το String city στην ιστοσελίδα.

```
[Route("moveto/{city}")]
        public IActionResult Visit([FromQuery] string city)
        {
            return Ok(city);
        }
```

Η δράση αυτή παρακάτω επιστρέφει και χτίζει το String builder με το String text number φορές και το δείχνει στην ιστοσελίδα.

```
        public IActionResult Repeat(string text, int number)
        {
            var builder = new StringBuilder();
            for(var i=0; i<number; i++)
            {
                builder.AppendFormat("{0}, ", text);
            }
            return Ok(builder.ToString());
        }
```

Η δράση αυτή παρακάτω επιστρέφει και χτίζει το String builder με το String text input.Number φορές και το δείχνει στην ιστοσελίδα.

```
        public IActionResult Repeat1(RepeatText input)
        {
            var builder = new StringBuilder();
            for (var i = 0; i < input.Number; i++)
            {
                builder.AppendFormat("{0}, ", input.Text);
            }
            return Ok(builder.ToString());
        }
```

Η δράση αυτή παρακάτω επιστρέφει το String language στην ιστοσελίδα.

```
        public IActionResult Accept(
            [FromHeader(Name="Accept-Language")] string language)
        {
            return Ok(language);
        }
```

Η δράση αυτή παρακάτω δείχνει την ιστοσελίδα.

```
        [HttpGet]
        [ActionName("email")]
        public IActionResult EmailGet(IList<string> emails)
        {
            return View();
        }
```

Η δράση αυτή παρακάτω δείχνει την ιστοσελίδα.

```
        [HttpPost]
        [ActionName("email")]
        public IActionResult EmailPost(
             [Bind(Prefix="email")] IList<string> emails)
        {
```

```
            return View();
        }

         Η δράση αυτή παρακάτω δείχνει την ιστοσελίδα.
 public IActionResult Date(DateTime date, int day, int month, int year=2021)
        {
            return View();
        }
```

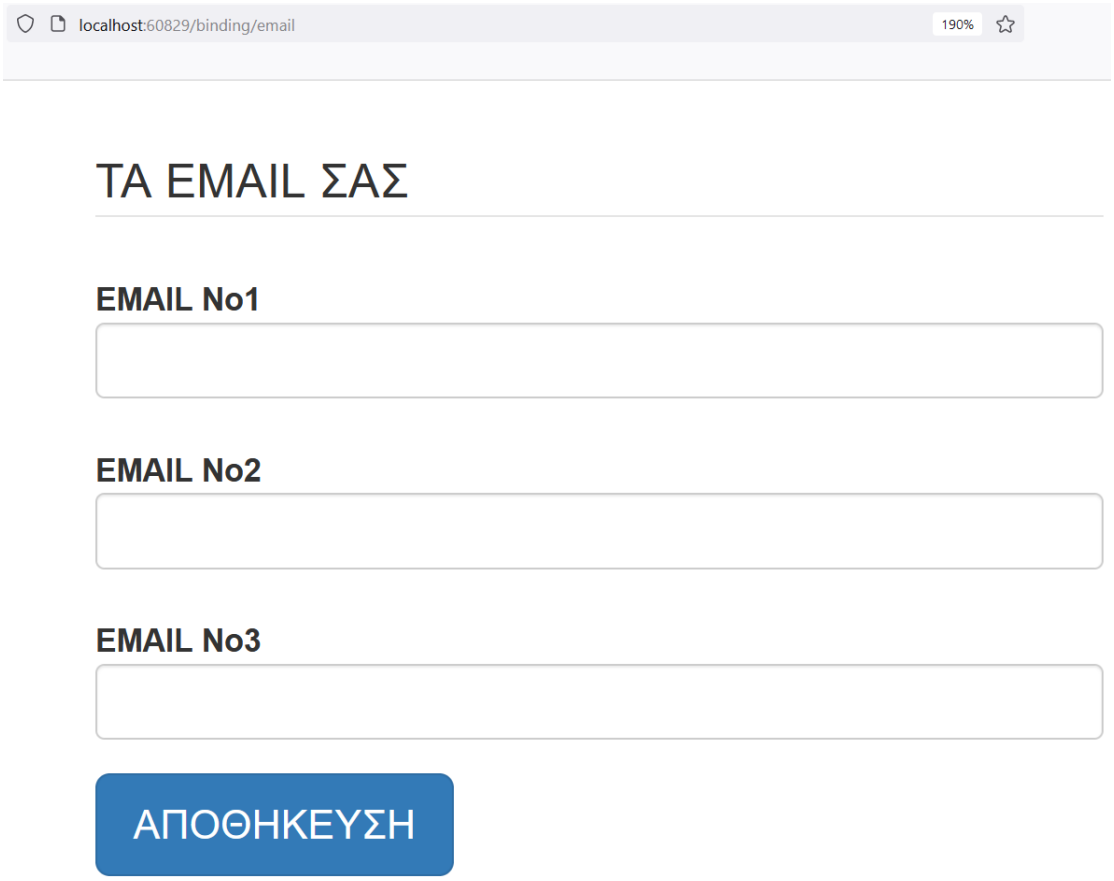
Index.cshtml
Η σελίδα παρακάτω εκτελεί τη δράση async .

```
<h1>HOME.INDEX</h1>

<a href="@Url.Action("Async", "Home")">Δοκιμάστε Ασύγχρονες Δράσεις</a>
```

Η παρακάτω ιστοσελίδα εισάγει και σώζει τρία email.
Email.cshtml

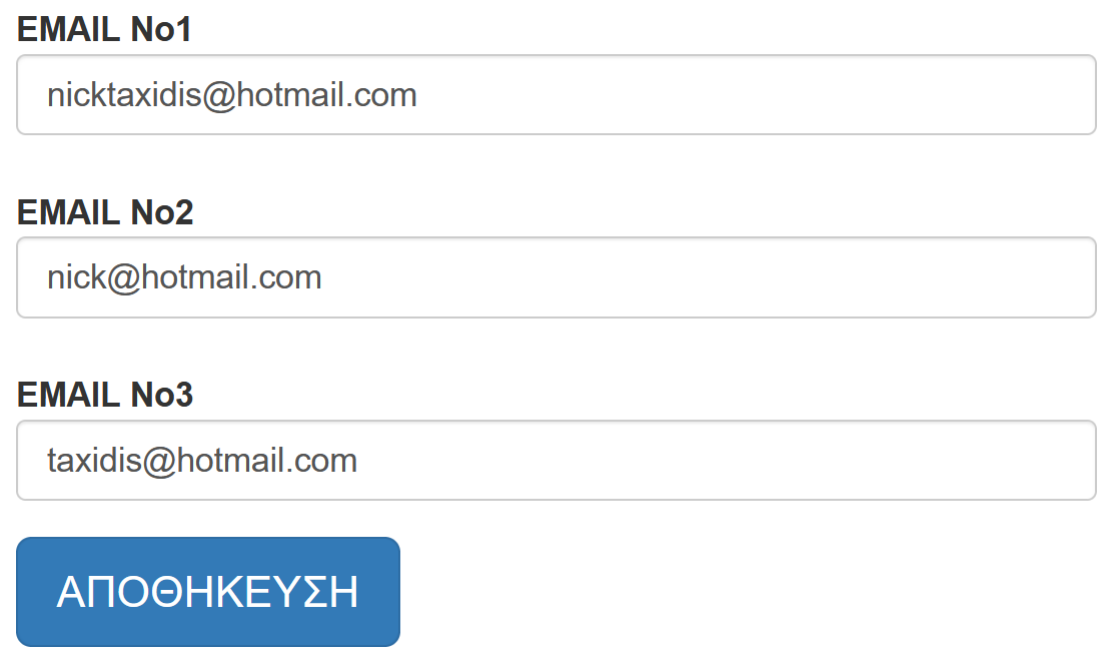
```
<div class="col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3" style="margin-
top: 30px">

    <form class="form-horizontal" method="post" action="@Url.Action("email",
"binding")">
        <fieldset>
            <legend>ΤΑ EMAIL ΣΑΣ</legend>

            <!-- EMAIL 1 -->
            <div class="form-group">
                <label class="col-md-4 control-label" for="email1">EMAIL
No1</label>
                <div class="col-md-4">
                    <input name="email" id="email1" class="form-control input-
md" type="text">
                </div>
            </div>

            <!-- EMAIL 2 -->
            <div class="form-group">
                <label class="col-md-4 control-label" for="email2">EMAIL
No2</label>
                <div class="col-md-4">
                    <input name="email" id="email2" class="form-control input-
md" type="text">
                </div>
            </div>

            <!-- EMAIL 3 -->
            <div class="form-group">
                <label class="col-md-4 control-label" for="email3">EMAIL
No3</label>
                <div class="col-md-4">
                    <input name="email" id="email3" class="form-control input-
md" type="text">
                </div>
            </div>
```

```
            <button class="btn btn-primary btn-lg">ΑΠΟΘΗΚΕΥΣΗ</button>
        </fieldset>
    </form>

</div>
```



Εικόνα 31 – Φόρμα email

Ο χρήστης εισάγει τα στοιχεία του και πατά ΑΠΟΘΗΚΕΥΣΗ.

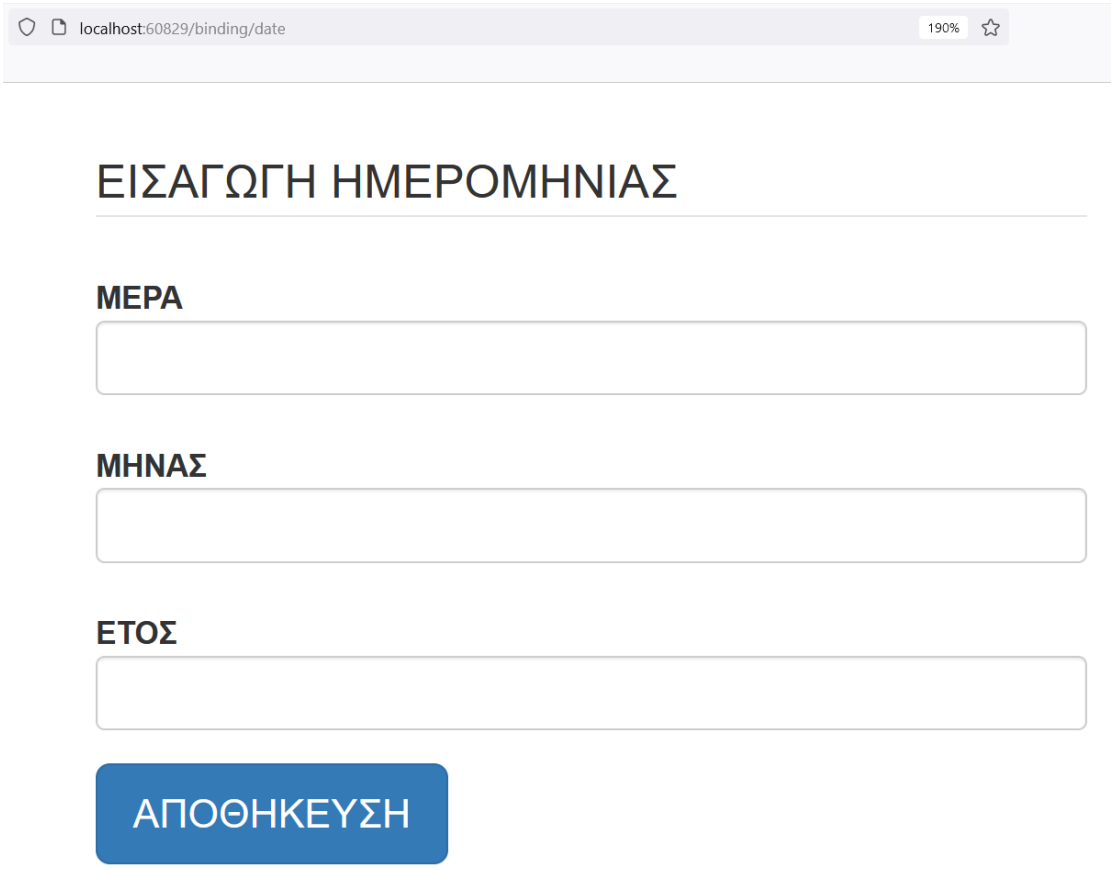


Εικόνα 32 – Εισαγωγή email

Date.cshtml
Η παρακάτω ιστοσελίδα σώζει την ημερομηνία.

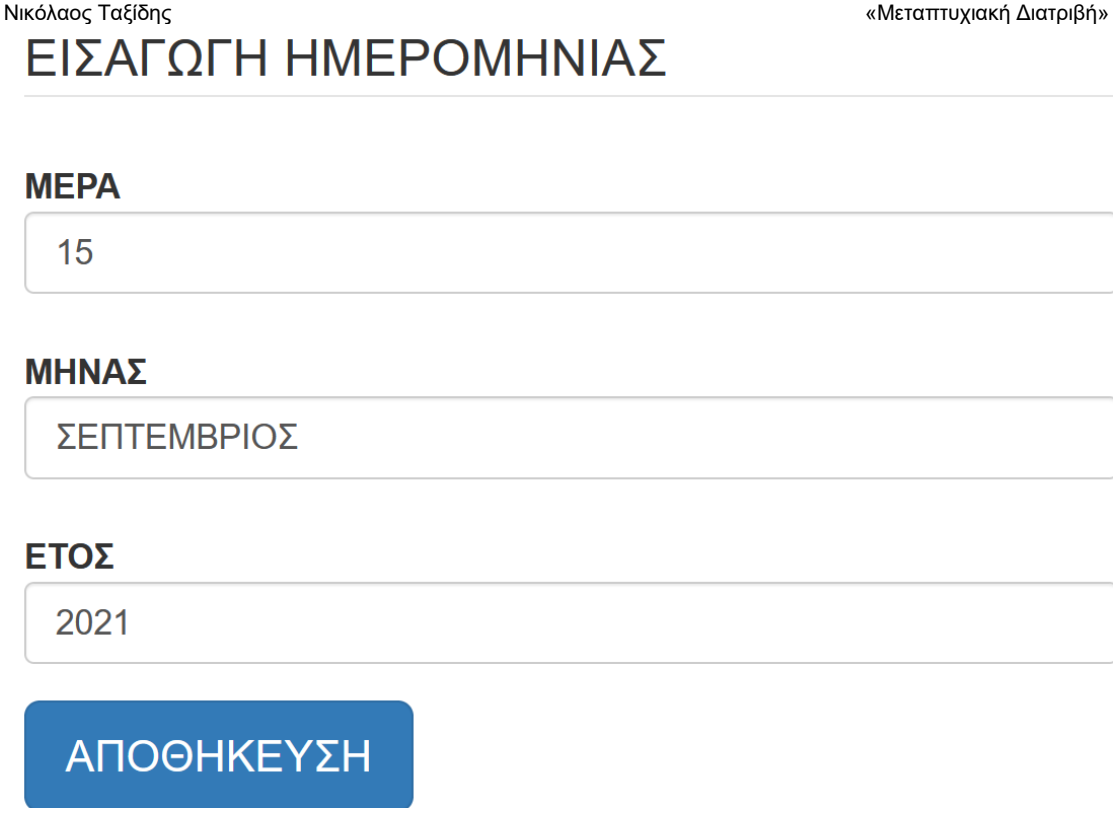
```
<div class="col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3" style="margin-
top: 30px">

    <form class="form-horizontal" method="post" action="@Url.Action("date",
"binding")">
        <fieldset>
            <legend>ΕΙΣΑΓΩΓΗ ΗΜΕΡΟΜΗΝΙΑΣ</legend>

            <!-- DD -->
            <div class="form-group">
                <label class="col-md-4 control-label" for="email1">ΜΕΡΑ</label>
                <div class="col-md-4">
          <input name="day" id="day" class="form-control input-md" type="text">
                </div>
            </div>

            <!-- EMAIL 2 -->
            <div class="form-group">
              <label class="col-md-4 control-label" for="email2">ΜΗΝΑΣ</label>
           <div class="col-md-4">
     <input name="month" id="month" class="form-control input-md" type="text">
                </div>
```

```
            </div>

            <!-- EMAIL 3 -->
            <div class="form-group">
                <label class="col-md-4 control-label" for="email3">ΕΤΟΣ</label>
                <div class="col-md-4">
                    <input name="year" id="year" class="form-control input-md"
type="text">
                </div>
            </div>

            <button class="btn btn-primary btn-lg">ΑΠΟΘΗΚΕΥΣΗ</button>
        </fieldset>
    </form>

</div>
```



Εικόνα 33 - Φόρμα Ημερομηνίας

Ο χρήστης εισάγει την ημερομηνία και πατά ΑΠΟΘΗΚΕΥΣΗ.



Εικόνα 34 – Εισαγωγή Ημερομηνίας

- Συνολικά, τα φίλτρα ενεργειών σχηματίζουν ένα ενσωματωμένο πλαίσιο προσανατολισμένο σε διαστάσεις εντός του ASP.NET Core.
- Όταν πρόκειται να γραφεί ένα φίλτρο ενεργειών, συνήθως κληρονομείτε από το ActionFilterAttribute και απλώς προσθέτει ο χρήστης την επιθυμητή συμπεριφορά.

Βιβλιογραφία

Programming ASP.NET Core, First Edition by Dino Esposito του Dino Esposito

ASP.NET Core in Action του Andrew Lock

Κεφάλαιο 5 - Προβολές (Views) στην ASP.NET Core MVC

Στο μοτίβο Model-View-Controller (MVC), η προβολή (View) χειρίζεται την παρουσίαση δεδομένων της εφαρμογής και την αλληλεπίδραση των χρηστών. Η προβολή είναι ένα πρότυπο HTML με ενσωματωμένη σήμανση ξυραφιού(Razor).

Η σήμανση Razor είναι ένας κώδικας που αλληλεπιδρά με τη σήμανση HTML για την παραγωγή μιας ιστοσελίδας που αποστέλλεται στον πελάτη. Στην ASP.NET Core MVC, οι προβολές είναι αρχεία .cshtml που χρησιμοποιούν τη γλώσσα προγραμματισμού C # στη σήμανση Razor.

ViewEngine

```
//MultiTenantViewLocationExpander.cs
//   Ch05 - ASP.NET MVC Views
//   ViewEngine
//

using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Http.Extensions;
using Microsoft.AspNetCore.Mvc.Razor;

namespace Ch05.ViewEngine.Common
{
    public class MultiTenantViewLocationExpander : IViewLocationExpander
    {
        public void PopulateValues(ViewLocationExpanderContext context)
        {
            var tenant =
context.ActionContext.HttpContext.Request.GetDisplayUrl();
            context.Values["tenant"] = "contoso";  //tenant;
        }

        public IEnumerable<string>
ExpandViewLocations(ViewLocationExpanderContext context, IEnumerable<string>
viewLocations)
        {
            if (!context.Values.ContainsKey("tenant") ||
                string.IsNullOrWhiteSpace(context.Values["tenant"]))
                return viewLocations;

            var overriddenViewNames = viewLocations
                .Select(f => f.Replace("/Views/", "/Views/" +
context.Values["tenant"] + "/"))
                .Concat(viewLocations)
                .ToList();
            return overriddenViewNames;
        }
    }
}
```

Οι φόρμες θέσης προβολής είναι μια στατική ρύθμιση για τη μηχανή προβολής (View Engine). Ορίζει ο χρήστης τις μορφές θέσης προβολής κατά την εκκίνηση της εφαρμογής και παραμένουν ενεργές για όλη τη διάρκεια ζωής τους. Κάθε φορά που μια προβολή πρέπει να ρυθμίζεται, η μηχανή προβολής περνά από τη λίστα των καταχωρημένων τοποθεσιών έως ότου εντοπίσει μια τοποθεσία που περιέχει το επιθυμητό πρότυπο. Εάν δεν βρεθεί το πρότυπο που ζητείται , εμφανίζεται μια εξαίρεση.

Αντίθετα, αν πρέπει να καθοριστεί η διαδρομή προς την προβολή δυναμικά ανά αίτηση η βάση κώδικα είναι πάντα ίδια και είναι πάντα το ίδιο σύνολο λογικών προβολών, αλλά σε κάθε χρήστη μπορεί να δοθεί μια συγκεκριμένη έκδοση της προβολής, ίσως με διαφορετικό στυλ ή με διαφορετική διάταξη. Μια κοινή προσέγγιση για αυτόν τον τύπο εφαρμογής είναι

ο καθορισμός της συλλογής προεπιλεγμένων προβολών και, στη συνέχεια, η δυνατότητα στους χρήστες να προσθέσουν προσαρμοσμένες προβολές.

Αν ο χρήστης Contoso μεταβαίνει στο view index.cshtml και αναμένει να δει το Views / Contoso / Home / index.cshtml αντί για την προεπιλεγμένη προβολή στο Views / Home / index.cshtml, χρειάζονται οι επεκτάσεις τοποθεσίας προβολής που είναι ένας νέος τύπος στοιχείου που έχει δημιουργηθεί για την δυναμική επίλυση προβολών.

Η επέκταση θέσης προβολής είναι μια κλάση που εφαρμόζει τη διεπαφή IViewLocationExpander. Στο PopulateValues, έχει πρόσβαση ο χρήστης στο περιβάλλον HTTP και επιλέγει την τιμή κλειδιού που θα καθορίσει τη διαδρομή προβολής που θα χρησιμοποιηθεί. Η τιμή κλειδιού που θα χρησιμοποιηθεί για τον προσδιορισμό της διαδρομής αποθηκεύεται στο περιβάλλον επέκτασης τοποθεσίας προβολής. Στο ExpandViewLocations, λαμβάνετε η τρέχουσα λίστα μορφών τοποθεσίας προβολής, με δυνατότητα επεξεργασίας ανάλογα με το τρέχον πλαίσιο και επιστρέφετε. Η επεξεργασία της λίστας συνήθως σημαίνει την εισαγωγή πρόσθετων μορφών τοποθεσίας προβολής και συγκεκριμένου περιβάλλοντος.

Στο επάνω μέρος της λίστας έχουν προστεθεί μορφές τοποθεσίας ειδικά για τον ενοικιαστή(Tenant), πράγμα που σημαίνει ότι οποιαδήποτε παράκαμψη της προβολής θα υπερισχύει οποιασδήποτε προεπιλεγμένης προβολής.

```
// Startup.cs

public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            services
                .AddMvc()
                .AddRazorOptions(options =>
                {
                    // {0} - Action Name
                    // {1} - Controller Name
                    // {2} - Area Name
                    options.ViewLocationFormats.Clear();
                    options.ViewLocationFormats.Add("/Views/{1}/{0}.cshtml");

options.ViewLocationFormats.Add("/Views/Shared/{0}.cshtml");

options.ViewLocationFormats.Add("/Views/Shared/Layouts/{0}.cshtml");

options.ViewLocationFormats.Add("/Views/Shared/PartialViews/{0}.cshtml");
                    options.ViewLocationExpanders.Add(new
MultiTenantViewLocationExpander());
                });
```

// SpyTemplateEngine.cs

```
//   Ch05 - ASP.NET MVC Views
//   ViewEngine
//

using Microsoft.AspNetCore.Mvc.Razor.Extensions;
using Microsoft.AspNetCore.Razor.Language;

namespace Ch05.ViewEngine.Common
{
    public class SpyTemplateEngine : MvcRazorTemplateEngine
    {
        public SpyTemplateEngine(RazorEngine engine, RazorProject project)
            : base(engine, project)
        {
        }

        public override RazorCSharpDocument GenerateCode(RazorCodeDocument
codeDocument)
        {
            var csharpDocument = base.GenerateCode(codeDocument);
            var generatedCode = csharpDocument.GeneratedCode;

            // Κοίτα τον δημιουργημένο κώδικα.

            return csharpDocument;
        }
    }

}// Δημιουργεί τον κώδικα που προσδιορίζει ο Razor κώδικας.
```

Εδώ γεννιέται ο κώδικας που προσδιορίζει τον Razor κώδικα.

Ο generatedCode θα κάνει διαθέσιμο το csharpDocument στον κώδικα που δημιουργείται δυναμικά για την προβολή Razor.

```
//   Dog.cs
//   Ch05 - ASP.NET MVC Views
//   ViewEngine
//

namespace Ch05.ViewEngine.Common
{
    public class Dog
    {
        public string onoma { get; set; }

        public override string ToString()
        {
            return onoma;
        }
    }
}
```

```
//   HomeController.cs
//   Ch05 - ASP.NET MVC Views
//   ViewEngine
//

using Ch05.ViewEngine.Common;
using Microsoft.AspNetCore.Http.Extensions;
using Microsoft.AspNetCore.Mvc;

namespace Ch05.ViewEngine.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            ViewData["Url"] = Request.GetDisplayUrl();

            var azor = new Dog {onoma = "Azor"};
            ViewBag.MyDogName = azor;
            ViewBag.Url = Request.GetDisplayUrl();

            return View();
        }

        public IActionResult Test()
        {
            return View();
        }
    }
}
```

Η μέθοδος προβολής πακετάρει το όνομα του προτύπου Razor, της κύριας προβολής και του μοντέλου προβολής για να επιστρέψει ένα μεμονωμένο αντικείμενο που εφαρμόζει τη διεπαφή IActionResult.

Το ViewBag είναι μια άλλη ιδιότητα που ορίζεται στη βασική κλάση ελεγκτή της οποίας το περιεχόμενο μεταφέρεται αυτόματα στην κατηγορία προβολής. Το ViewBag διαφέρει από το ViewData επειδή επιτρέπει άμεση πρόσβαση προγραμματισμού σε ιδιότητες, αποφεύγοντας έτσι την τυπική πρόσβαση στο λεξικό που υποστηρίζεται από το ViewData.

Μπορείτε να πληκτρολογήσετε οποιοδήποτε όνομα ιδιότητας δίπλα στην αναφορά αντικειμένου ViewBag και ο μεταγλωττιστής C # θα το εκτελέσει.

Home/Index.cshtml

```
@using Microsoft.AspNetCore.Mvc.ViewFeatures
<h1>ViewData and ViewBag</h1>

<hr />

@ViewData["Url"]

<hr/>
```

```
@ViewBag.Url<br/>
@ViewBag.MyDogName<br/>
```

Εδώ γίνεται εμφάνιση των δεδομένων στον Browser.

Στην Οθόνη εμφανίζεται ο αριθμός του localhost και το όνμα του Dog που είναι Azor .

Εικόνα 35 – Εμφάνιση του localhost

ExtraViews

Το μοντέλο viewmodelbase έχει τα πεδία και τον constructor παρακάτω:

ViewModelBase.cs

```
public class ViewModelBase
    {
        public ViewModelBase(string title = "")
        {
            Title = title;
            ErrorMessage = "";
            StatusCode = 0;
        }

        public string Title { get; set; }
        public string ErrorMessage { get; set; }
        public int StatusCode { get; set; }
    }
```

Το μοντελο homeviewmodel έχει τα πεδια παρακατω:

```
public class HomeViewModel : ViewModelBase
    {
        public HomeViewModel(string title) : base(title)
        {

        }

        public string Today { get; set; }
        public DateTime Current { get; set; }
    }
```

HomeService.cs

Η υπηρεσία homeservice θέτει στο μοντέλο την σημερινή ημερομηνία.

```
  public class HomeService : ApplicationServiceBase, IHomeService
    {
        public HomeViewModel GetHomeViewModel()
        {
            var model = new HomeViewModel("Χρώματα και Ημερομηνίες")
            {
                Current = DateTime.Today
            };
            model.Today = model.Current.ToString("dddd, <b>d MMM</b> yyyy");
            return model;
        }
    }
```

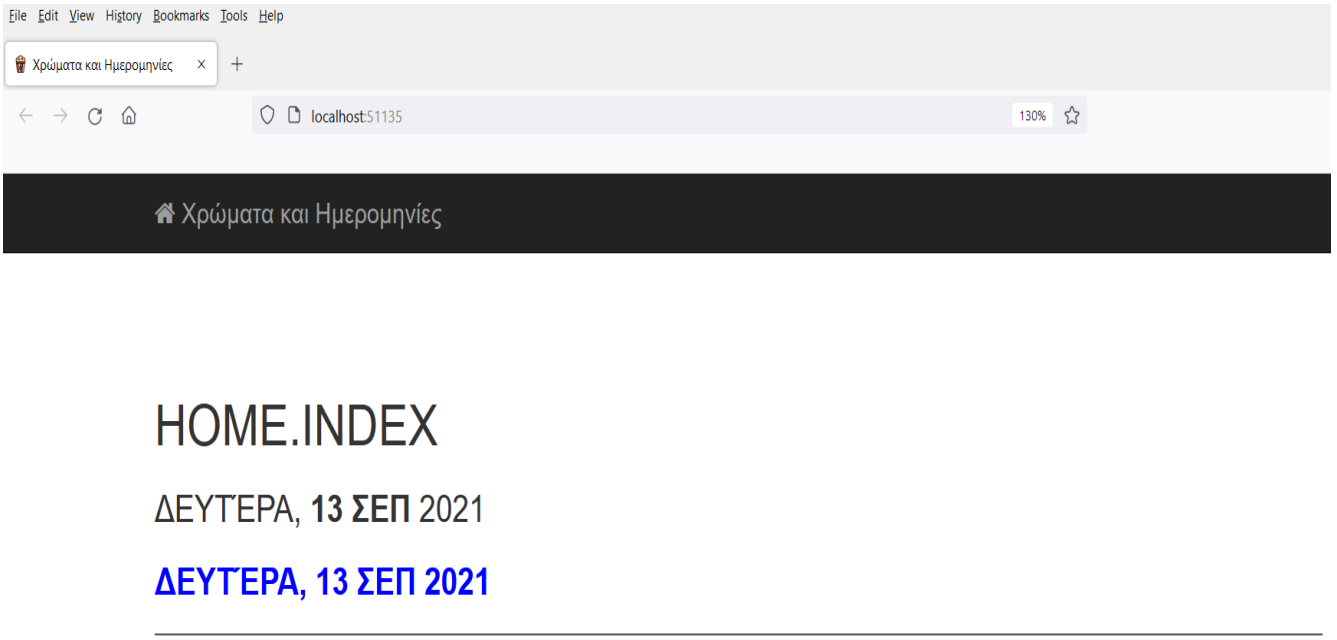
Η δράση index προβάλλει την ημερομηνία και την ιδιότητα χρώμα στην ιστοσελίδα.

HomeController.cs

```
public class HomeController : Controller
  {
     private readonly IHomeService _homeService;


     public HomeController(IHomeService service)
     {
        _homeService = service;
     }


     public IActionResult Index()
     {
        var model = _homeService.GetHomeViewModel();
      //  ViewData["Xrwma"] = "portokali";
     //    ViewBag.Color = "kyano";
        return View(model);
     }
  }
```

Η σελίδα Index.cshtml είναι η παρακάτω.

```
@model Ch05.JustViews.Models.HomeViewModel

@{
    var random = DateTime.Now.Second % 2 > 0 ? "blue" : "yellow";
    var week = Model.Current.ToString("dddd, ");
    var day = Model.Current.ToString("d MMM ");
    var year = Model.Current.ToString("yyyy");
}
<h1>Home.Index</h1>

<h3>@Html.Raw(Model.Today)</h3>
<h3>@week <b style="color:red">@day</b> <span
style="color:@random">@year</span></h3>

<hr />
<h3>@ViewData["Xrwma"]</h3>
<h3>@ViewBag.Colr</h3>
```



Εικόνα 36 – Χρώματα και Ημερομηνίες με Μπλε χρώμα

Όταν έτρεξε το πρόγραμμα η κάτω ημερομηνία εμφανίστηκε σε μπλε χρώμα.

Εικόνα 37 – Χρώματα και Ημερομηνίες με Κίτρινο χρώμα

Όταν ξαναέτρεξε το πρόγραμμα η κάτω ημερομηνία εμφανίστηκε σε κίτρινο χρώμα.

Το χρώμα της κάτω ημερομηνίας αλλάζει με τυχαίο τρόπο κάθε φορά που ανανεώνετε η ιστοσελίδα.

Βιβλιογραφία

Programming ASP.NET Core, First Edition by Dino Esposito του Dino Esposito

ASP.NET Core in Action του Andrew Lock

Κεφάλαιο 6 - RAZOR ΣΕΛΙΔΕΣ(PAGES)

Στην ASP.NET Core MVC, οι προβολές Στην κλασική ASP.NET MVC, δεν υπάρχει τρόπος αναφοράς ενός προτύπου ξυραφιού μέσω μιας απευθείας διεύθυνσης URL.

Μια διεύθυνση URL μπορεί να χρησιμοποιηθεί για τη σύνδεση μιας στατικής σελίδας HTML ή της εξόδου HTML που οργανώνεται με μια μέθοδο δράσης ελεγκτή. Η ASP.NET Core δεν αποτελεί εξαίρεση.

Ωστόσο, ξεκινώντας με την ASP.NET Core 2.0, διατίθεται μια νέα δυνατότητα - Σελίδες ξυραφιού (Razor) - που επιτρέπουν στον χρήστη να καλέσει ένα πρότυπο Razor απευθείας μέσω της διεύθυνσης URL χωρίς καμία διαμεσολάβηση ελεγκτή.

Ανακαλύπτοντας το σκεπτικό πίσω από τις Razor Σελίδες θα γίνει αντιληπτό ότι μερικές φορές, ωστόσο, διατίθενται μέθοδοι ελεγκτή που απλώς εξυπηρετούν κάποια αρκετά στατική σήμανση.

Ένα τυπικό παράδειγμα είναι οι σελίδες About Us ή Contact Us ενός τυπικού ιστότοπου.

Tag Helpers

```
//HomeController.cs
//   PROGRAMMING ASP.NET CORE
//
//
//   Ch06 - The Razor Syntax
//   TagHelpers
//

using Ch06.TagHelpers.Application;
using Microsoft.AspNetCore.Mvc;

namespace Ch06.TagHelpers.Controllers
{
    public class HomeController : Controller
    {
        private readonly HomeService _homeService;

        public HomeController(HomeService service)
        {
            _homeService = service;
        }

        public IActionResult Index()
        {
            var model = _homeService.GetHomeViewModel();
            return View(model);
        }

        public IActionResult Room()
        {
            var model = _homeService.GetRoomViewModel();
            return View(model);
        }
    }
}
```

Λειτουργεί με το HomeService Model και επιστρέφει στην σελίδα Index ένα HomeView Model και στη σελίδα Room ένα RoomView Model.

```
// ViewModelBase.cs
//   PROGRAMMING ASP.NET CORE
//   Ch06 - The Razor Syntax
//   TagHelpers
//

namespace Ch06.TagHelpers.Models
{
    public class ViewModelBase
    {
        public ViewModelBase(string title = "")
        {
            titlos = title;
            mhnymalathoys = "";
            kwdikoskatastasis = 0;
        }

        public string titlos { get; set; }
        public string mhnymalathoys { get; set; }
        public int kwdikoskatastasis { get; set; }
    }
}
```

Αποτελείται από τα πεδία τίτλος, μήνυμα λάθους και τον ακέραιο κωδικό κατάστασης που είναι ακέραιος.

```
//RoomViewModel.cs
//   PROGRAMMING ASP.NET CORE
//
//   Ch06 - The Razor Syntax
//   TagHelpers
//

using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace Ch06.TagHelpers.Models
{
    public class RoomViewModel : ViewModelBase
    {
        public RoomViewModel(string title) : base(title)
        {
            RoomTypes = new List<string>();
        }

        public IList<string> RoomTypes { get; set; }
        public RoomCategories CurrentRoomType { get; set; }
    }

    public enum RoomCategories
    {
        [Display(Name = "MH KATHORISMENO")]
        None,
        Single,
        [Display(Name = "POLY MEGALO DWMATIO")]
        Double
    }
}
```

Αποτελείται από το RoomTypes που είναι μια λίστα από Strings και το CurrentRoomType που είναι ένας τύπος απαρίθμησης με τις τιμές None, Single, Double.

```
// HomeViewModel.cs
//   PROGRAMMING ASP.NET CORE
//
//
//   Ch06 - The Razor Syntax
//   TagHelpers
//

namespace Ch06.TagHelpers.Models
{
    public class HomeViewModel : ViewModelBase
    {
        public HomeViewModel(string title) : base(title)
        {

        }
    }
}
```

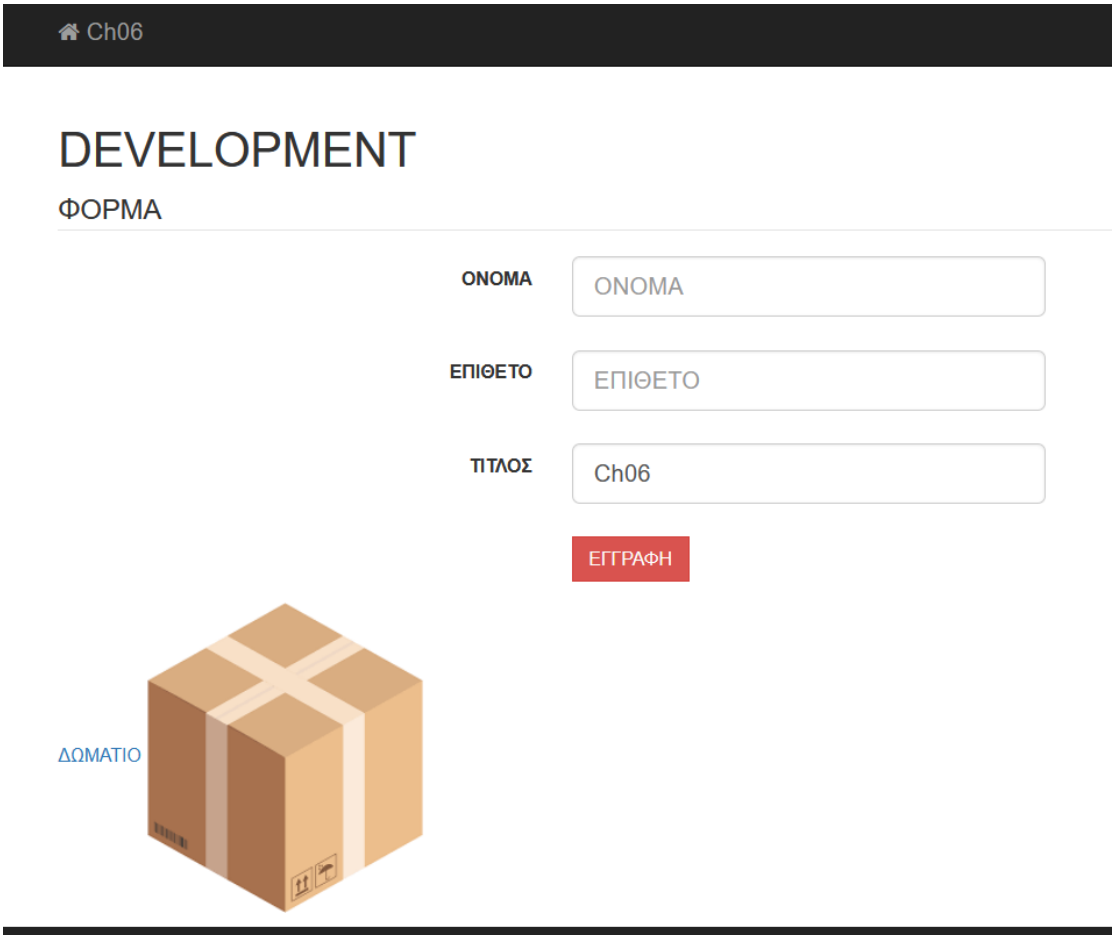
Αυτό είναι ένα μοντέλο που επεκτείνει το `ViewModelBase`.

Index.cstml

```
@using Microsoft.AspNetCore.Hosting
@model Ch06.TagHelpers.Models.HomeViewModel
@inject IHostingEnvironment Host


<h1>@Host.EnvironmentName</h1>
<form class="form-horizontal"
      method="post"
      a1ction="#"
      asp-action="Register"
      asp-controller="Account"
      asp-hello="hello">
    <fieldset>

        <!-- Form Name -->
        <legend>ΦΟΡΜΑ</legend>

        <!-- Text input-->
        <div class="form-group">
            <label class="col-md-4 control-label" for="firstname">ONOMA</label>
            <div class="col-md-4">
                <input type="text" id="firstname" name="firstname"
                       class="form-control input-lg"
                       placeholder="ONOMA">
                <span class="help-block"></span>
```

```
            </div>
        </div>

        <!-- Text input-->
        <div class="form-group">
            <label class="col-md-4 control-label"
for="lastname">ΕΠΙΘΕΤΟ</label>
            <div class="col-md-4">
                <input type="text" id="lastname" name="lastname"
                       class="form-control input-lg"
                       placeholder="ΕΠΙΘΕΤΟ">
                <span class="help-block"></span>
            </div>
        </div>

        <!-- Text input-->
        <div class="form-group">
            <label class="col-md-4 control-label" asp-
for="titlos">ΤΙΤΛΟΣ</label>
            <div class="col-md-4">
                <input class="form-control input-lg" asp-for="titlos"
placeholder="ΤΙΤΛΟΣ">
                <span class="help-block"></span>
            </div>
        </div>

        <!-- Button -->
        <div class="form-group">
            <label class="col-md-4 control-label"
for="registerButton"> </label>
            <div class="col-md-4">
                <button id="registerButton" class="btn btn-danger">
                    ΕΓΓΡΑΦΗ
                </button>
            </div>
        </div>

    </fieldset>
</form>

<a href="@Url.Action("room", "home")">ΔΩΜΑΤΙΟ</a>

<environment names="Development">
    <link rel="stylesheet" href="~/css/site1.css" />
    <link rel="stylesheet" href="~/css/site2.css" />
</environment>
<environment names="Staging,Production">
    <link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true"
/>
</environment>

<img src="~/content/images/app-logo.png"
     asp-append-version="true" />
```

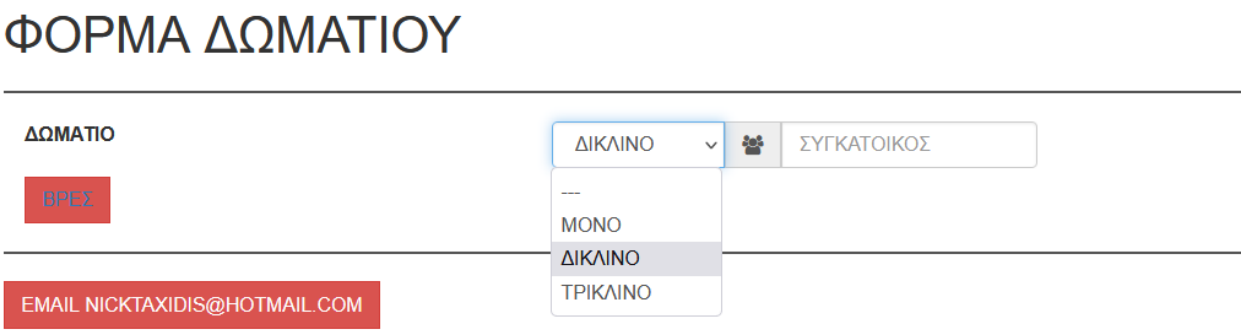
Είναι η αρχική σελίδα όπου ο χρήστης εισάγει τα στοιχεία του.



Εικόνα 38 – Φόρμα Εισαγωγής Στοιχείων Χρήστη

Εικόνα 39 – Εισαγωγή Στοιχείων Χρήστη

Ο χρήστης εισάγει τα στοιχεία του και πατάει ΕΓΓΡΑΦΗ και έτσι γίνεται η εγγραφή.



Εικόνα 40 – Φόρμα Εισαγωγής Στοιχείων Δωματίου

Αν ο χρήστης πατήσει ΔΩΜΑΤΙΟ θα του εμφανιστεί η παρπάνω φόρμα, όπου εισάγει το δωμάτιο που θέλει, ανάλογα με τον αριθμό των κλινών και το όνομα του συγκατοίκου του και πατάει ΒΡΕΣ, ώστε να βρεθεί το συγκεκριμένο δωμάτιο ή μπορεί να πατήσει το email του διαχειριστή της ιστοσελίδας για τυχόν απορία.

Room.cshtml

```
@using Ch06.TagHelpers.Models
@using Microsoft.AspNetCore.Hosting
@using Microsoft.Extensions.Options
@model Ch06.TagHelpers.Models.RoomViewModel
@{
    var email = "nicktaxidis@hotmail.com";
    var subject = "Μιλώντας για ASP.NET Core";
}

<form asp-action="@Url.Action("Hello", "Form")">
    @Html.AntiForgeryToken()
    <h1>ΦΟΡΜΑ ΔΩΜΑΤΙΟΥ</h1>
</form>

<div class="form-group">
    @*<label class="col-md-4 control-label" for="hotel_roomtype">Room</label>
    <div class="col-md-4">
        </div>
    </div>*@
</div>
<hr/>

<div class="clearfix"></div>
```

```
<div class="form-group">
    <label class="col-md-4 control-label" for="hotel_roomtype">ΔΩΜΑΤΙΟ</label>
    <div class="col-md-4">
        <div class="input-group col-xs-12">
            <select id="room1" name="room1"
                    class="form-control"
                    asp-for="@Model.CurrentRoomType"
                    asp-
items="@Html.GetEnumSelectList(typeof(RoomCategories))">
            </select>

            <span class="input-group-addon">
                <i class="fa fa-users"></i>
            </span>
            <input type="text" id="Roommate1" name="Roommate1"
                   class="form-control text-capitalize"
                   placeholder="ΣΥΓΚΑΤΟΙΚΟΣ">
        </div>
    </div>
</div>

<div class="form-group">
    <label class="col-md-4 control-label" for="registerButton"> </label>
    <div class="col-md-4">
        <button id="registerButton" class="btn btn-danger">
            <a href="https://www.google.com">ΒΡΕΣ</a>
        </button>
    </div>
</div>

<div class="clearfix"></div>
<hr />

<email to="@email" subject="@subject" class="btn btn-danger">
    ΓΕΙΑ ΣΗΜΕΡΑ ΕΙΝΑΙ @DateTime.Today
</email>
```

BootTagHelpers

ViewModelBase.cs

Χρησιμοποίει το μοντέλο ViewModelBase όπως παρακάτω:

```
  public class ViewModelBase
    {
        public ViewModelBase(string title = "")
        {
            titlos = title;
            minimalathoys = "";
            kwdikoskatastasis = 0;
        }

        public string titlos { get; set; }
        public string minimalathoys { get; set; }
        public int kwdikoskatastasis { get; set; }
    }
```

HomeController.cs

Χρησιμοποιεί τον κάτωθι ελεγκτή που προβάλλει το HomeViewModel στην ιστοσελίδα:

```
  public class HomeController : Controller
    {
        private readonly HomeService _homeService;

        public HomeController(HomeService service)
        {
            _homeService = service;
        }

        public IActionResult Index()
        {
            var model = _homeService.GetHomeViewModel();
            return View(model);
        }
    }
```

ModalContentTagHelpers.cs

```
    [HtmlTargetElement("mheader", ParentTag = "content")]
    [HtmlTargetElement("mfooter", ParentTag = "content")]
    [HtmlTargetElement("mbody", ParentTag = "content")]
    public class ModalContentItemTagHelper : TagHelper
    {
        public override async Task ProcessAsync(
            TagHelperContext context, TagHelperOutput output)
        {
            var modalContext = context.Items[typeof(ModalContext)] as
ModalContext;
```

```
            if (modalContext == null)
                throw new ArgumentException();

// Αξιολόγηση του περιεχομενου του Razor του σώματος του στοιχείου.
            var body = (await output.GetChildContentAsync()).GetContent();
            body = body.Trim();

            // Προετοιμασία output
            var originalTagName = output.TagName;
            var className = GetBootstrapClassFromTagName(originalTagName);
            output.TagName = "div";
            output.Attributes.SetAttribute("class", className);

            // Στήριξη AUTO-CLOSE στο header και footer
            if (modalContext.AutoClose)
            {
                if (string.Equals(originalTagName, ModalContext.HeaderTag,
StringComparison.InvariantCultureIgnoreCase))
                {
                    var button = "<button type=\"button\" class=\"close\" data-
dismiss=\"modal\">&times;</button>";
                    output.Content.AppendHtml(button);
                    output.Content.AppendHtml(body);
                }
                if (string.Equals(originalTagName, ModalContext.FooterTag,
StringComparison.InvariantCultureIgnoreCase))
                {
                    var button = "<button type=\"button\" class=\"btn btn-
primary\" data-dismiss=\"modal\">OK</button>";
                    output.Content.AppendHtml(body);
                    output.Content.AppendHtml(button);
                }
            }
        }

        private static string GetBootstrapClassFromTagName(string tagName)
        {
            if (string.Equals(tagName, ModalContext.HeaderTag,
StringComparison.InvariantCultureIgnoreCase))
                return "modal-header";
            if (string.Equals(tagName, ModalContext.FooterTag,
StringComparison.InvariantCultureIgnoreCase))
                return "modal-footer";
            if (string.Equals(tagName, ModalContext.BodyTag,
StringComparison.InvariantCultureIgnoreCase))
                return "modal-body";
            return "";
        }
    }
```

ModalContentTagHelper.cs

```
[HtmlTargetElement("content", ParentTag = "modal")]
    public class ModalContentTagHelper : TagHelper
    {
        public override async Task ProcessAsync(
            TagHelperContext context, TagHelperOutput output)
        {
            // Αξιολόγηση του περιεχομενου του Razor του σώματος του στοιχείου.
```

```
            var body = (await output.GetChildContentAsync()).GetContent();
            body = body.Trim();

            // Αντικατάσταση <toggle> με <button>
            output.TagName = "div";
            var modalContext = context.Items[typeof(ModalContext)] as
ModalContext;
            if (modalContext == null)
                throw new ArgumentException();

            // Προετοιμασία output
            output.Attributes.SetAttribute("class", "modal");
            output.Attributes.SetAttribute("id", modalContext.Id);
            output.Content.AppendHtml("<div class='modal-dialog'><div
class='modal-content'>");
            output.Content.AppendHtml(body);
            output.Content.AppendHtml("</div></div>");
        }
    }
```

ModalContext.cs

```
public class ModalContext
    {
        public const string HeaderTag = "mheader";
        public const string BodyTag = "mbody";
        public const string FooterTag = "mfooter";

        public string Id { get; set; }
        public bool AutoClose { get; set; }
    }
```

ModalTagHelper.cs

```
[HtmlTargetElement("modal")]
    public class ModalTagHelper : TagHelper
    {
        public override void Process(
            TagHelperContext context, TagHelperOutput output)
        {
            var autoClose = false;
            if (output.Attributes.ContainsName("autoclose"))

Boolean.TryParse(output.Attributes["autoclose"].Value.ToString(), out
autoClose);


            // Δημιουργία του context (γενικού πλαισίου) για τα child elements
            var modalContext = new ModalContext
            {
                Id = output.Attributes["id"].Value.ToString(),
                AutoClose = autoClose
            };
            context.Items[typeof(ModalContext)] = modalContext;
```

```
            // Αντικατάσταση <modal> με <div>
            output.TagName = "div";
            output.Attributes.Remove(context.AllAttributes["id"]);
            output.Attributes.Remove(context.AllAttributes["autoclose"]);
        }
    }
```

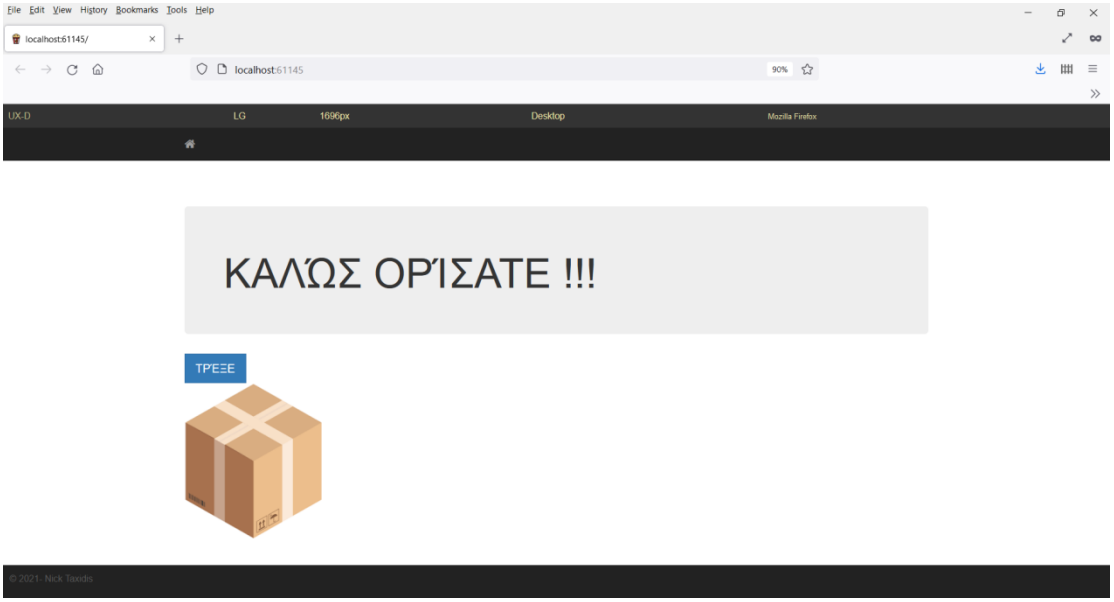
ToggleTagHelper.cs

```
[HtmlTargetElement("toggle", ParentTag = "modal")]
    public class ToggleTagHelper : TagHelper
    {
        public override void Process(
            TagHelperContext context, TagHelperOutput output)
        {
            // Αντικατάσταση <toggle> με <button>
            output.TagName = "button";
            var modalContext = context.Items[typeof(ModalContext)] as
ModalContext;
            if (modalContext == null)
                throw new ArgumentException();

            // Προετοιμασία output
            output.Attributes.SetAttribute("data-toggle", "modal");
            output.Attributes.SetAttribute("data-target", "#" +
modalContext.Id);
        }
    }
```

Η ιστοσελίδα Index.html:

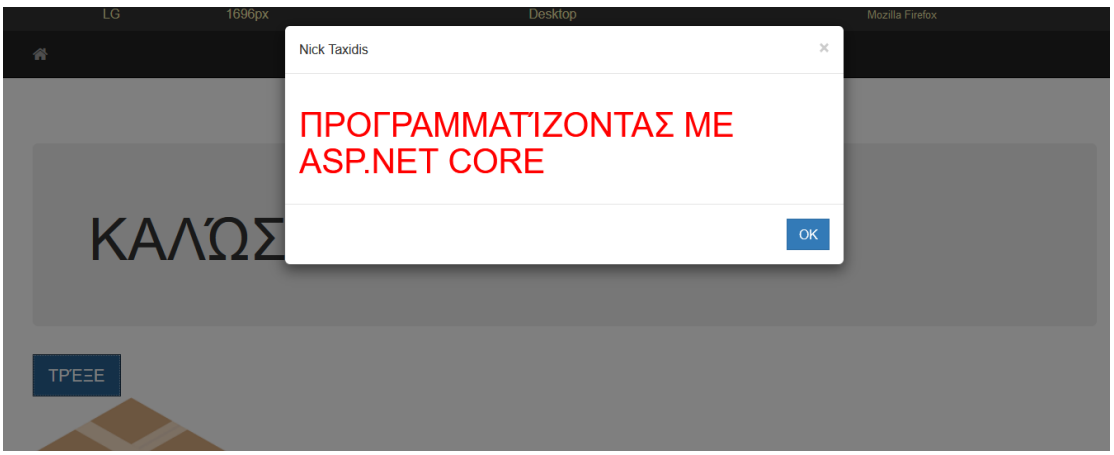
```
@model Ch06.BootstrapTagHelpers.Models.HomeViewModel


<div class="jumbotron">
    <h1>Καλώς Ορίσατε !!!</h1>
</div>
<modal id="myModal" autoclose="true">
    <toggle class="btn btn-primary btn-lg">Τρέξε</toggle>
    <content>
        <mheader>Nick Taxidis</mheader>
        <mbody>
            <h1> <span style="color:red">Προγραμματίζοντας με ASP.NET
Core</span></h1>
        </mbody>
        <mfooter></mfooter>
    </content>
</modal>

<environment names="Development">
    <link rel="stylesheet" href="~/css/site1.css" />
    <link rel="stylesheet" href="~/css/site2.css" />
</environment>
<environment names="Staging,Production">
    <link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true"
/>
</environment>
```

```
<img src="~/content/images/app-logo.png"
     asp-append-version="true" />
```



Εικόνα 41 – Αρχική Σελίδα

Πατώντας το κουμπί ΤΡΈΞΕ έχουμε την απόκριση παρακάτω:



Εικόνα 42 – ΠΡΟΓΡΑΜΜΑΤΊΖΟΝΤΑΣ ΜΕ ASP.NET CORE

Βιβλιογραφία

Programming ASP.NET Core, First Edition by Dino Esposito του Dino Esposito

ASP.NET Core in Action του Andrew Lock

Κεφάλαιο 7 - Ζητήματα σχεδιασμού

Η ASP.NET Core έχει το δικό της πλαίσιο DI που αρχικοποιείται αμέσως κατά την εκκίνηση της εφαρμογής. Ας δούμε τα πιο χαρακτηριστικά σημεία του.

Όταν το κοντέινερ διατίθεται στον κώδικα εφαρμογής, περιέχει ήδη μερικές διαμορφωμένες εξαρτήσεις.
IApplicationBuilder : Παρέχει τους μηχανισμούς για τη διαμόρφωση του αγωγού αιτήματος της εφαρμογής.
ILoggerFactory : Παρέχει το μοτίβο για τη δημιουργία στοιχείων καταγραφής.
IHostingEnvironment : Παρέχει πληροφορίες σχετικά με το περιβάλλον φιλοξενίας ιστού στο οποίο εκτελείται μια εφαρμογή.

Σε μια εφαρμογή ASP.NET Core, μπορεί ο χρήστης να εισάγει οποιονδήποτε από τους παραπάνω τύπους σε οποιοδήποτε έγκυρο σημείο έγχυσης κώδικα χωρίς κάποια προκαταρκτική διαμόρφωση. Ωστόσο, για να μπορέσει ο χρήστης να κάνει την έγχυση οποιουδήποτε άλλου
τύπου, πρέπει πρώτα να κάνει ένα βήμα εγγραφής.

Config

Τα ακόλουθα μοντέλα χρησιμοποιούνται:

ViewModelBase.cs

```
public class ViewModelBase
    {
        public ViewModelBase()
        {
            Orismata = new GeneralSettings();
        }

        public string TitlosEfarmogis { get; set; }
        public GeneralSettings Orismata { get; set; }
    }
```

GeneralSettings.cs

```
public class GeneralSettings
    {
        public GeneralSettings()
        {
            selidopoiisi = new PagingSettings();
            xroniadikaiwmatwn = new List<int>();
        }

        public IList<int> xroniadikaiwmatwn { get; set; }
        public PagingSettings selidopoiisi { get; set; }
        public int XronoZoni { get; set; }
    }

    public class PagingSettings
    {
        public int selidasmegethos { get; set; }
    }
```

Ακολουθούν οι ελεγκτές:

HomeController.cs

Προβάλλει στοιχεία για την σελιδοποίηση:

```
public class HomeController : Controller
    {
        private readonly IConfigurationRoot _diamorfosi;

        public HomeController(IConfigurationRoot diamorfosi)
```

```
        {
            _diamorfosi = diamorfosi;
        }
        public IActionResult Index()
        {
            ViewData["PageSize"] =
_diamorfosi.GetValue<int>("GeneralSettings:selidopoiisi:selidasmegethos");
            return View();
        }
    }
```

ConfigController.cs

```
public class ConfigController : Controller
    {
        private readonly GeneralSettings _orismata;

        public ConfigController(IOptionsSnapshot<GeneralSettings> orismata)
        {
            _orismata = orismata.Value;
        }
        public IActionResult Index()
        {
            ViewData["PageSize"] = _orismata.selidopoiisi.selidasmegethos;
            return View();
        }
    }
```

GeneralSettings.cs

```
private readonly GeneralSettings _orismata = new GeneralSettings();

        public MonitorController(IOptionsMonitor<GeneralSettings> monitor)
        {
            monitor.OnChange(trexontaorismata =>
                _orismata.selidopoiisi.selidasmegethos =
100+trexontaorismata.selidopoiisi.selidasmegethos);
        }
        public IActionResult Index()
        {
            ViewData["PageSize"] = _orismata.selidopoiisi.selidasmegethos;
            return View();
        }
    }
```

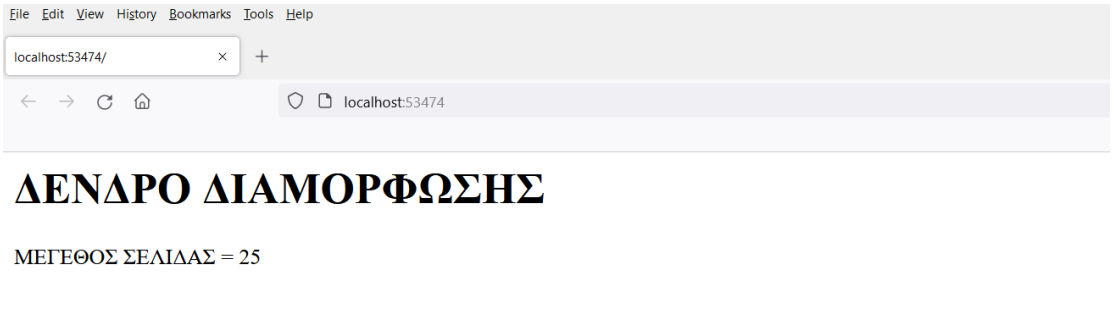
RandomController.cs

```
public class RandomController : Controller
    {
        private readonly IRandomCustomerService _ypiresia;
```

```
        public RandomController(IRandomCustomerService ypiresia)
        {
            _ypiresia = ypiresia;
        }
    }
```

Η σελίδα home/index

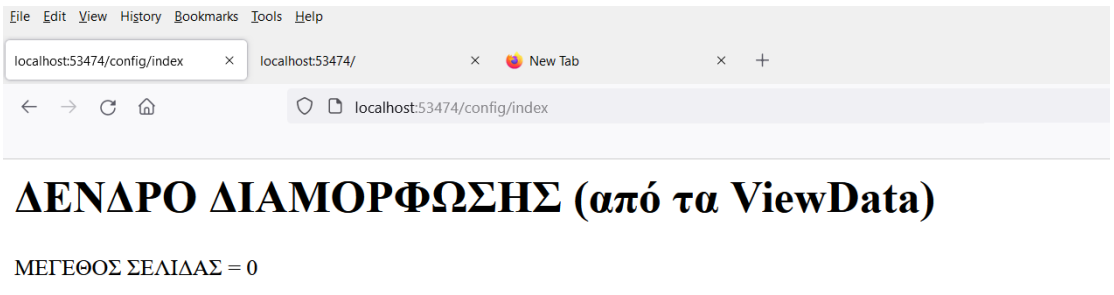
```
@using Microsoft.Extensions.Configuration
<h1>ΔΕΝΔΡΟ ΔΙΑΜΟΡΦΩΣΗΣ</h1>
@model Ch07.Config.Models.ViewModelBase
@inject IConfigurationRoot Configuration

@{
    var time = DateTime.Now;
}

ΜΕΓΕΘΟΣ ΣΕΛΙΔΑΣ = @Configuration["GeneralSettings:Paging:PageSize"]
```



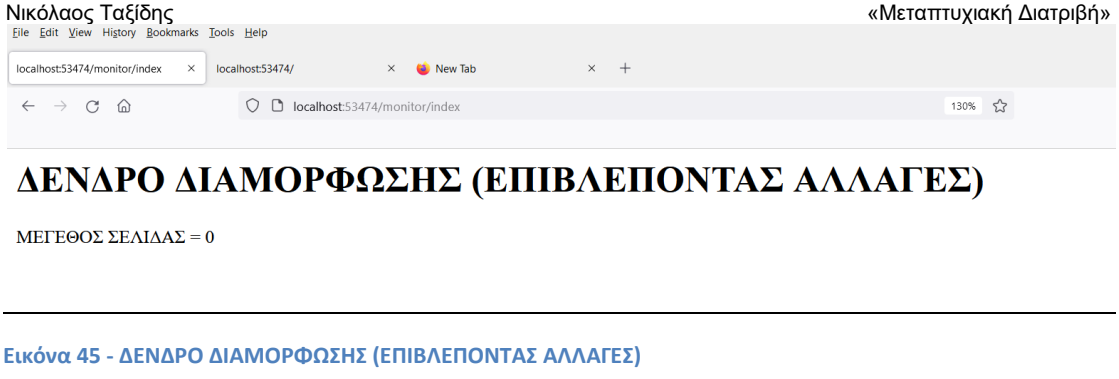
Εικόνα 43 – ΔΕΝΔΡΟ ΔΙΑΜΟΡΦΩΣΗΣ

Η σελίδα config/index

```
@using Microsoft.Extensions.Configuration
<h1>ΔΕΝΔΡΟ ΔΙΑΜΟΡΦΩΣΗΣ (από τα ViewData)</h1>

ΜΕΓΕΘΟΣ ΣΕΛΙΔΑΣ = @ViewData["PageSize"]
```



Εικόνα 44 – ΔΕΝΔΡΟ ΔΙΑΜΟΡΦΩΣΗΣ (από τα ViewData)



Εικόνα 45 - ΔΕΝΔΡΟ ΔΙΑΜΟΡΦΩΣΗΣ (ΕΠΙΒΛΕΠΟΝΤΑΣ ΑΛΛΑΓΕΣ)

Η σελίδα monitor/index

```
@using Microsoft.Extensions.Configuration
<h1>ΔΕΝΔΡΟ ΔΙΑΜΟΡΦΩΣΗΣ (ΕΠΙΒΛΕΠΟΝΤΑΣ ΑΛΛΑΓΕΣ)</h1>

ΜΕΓΕΘΟΣ ΣΕΛΙΔΑΣ = @ViewData["PageSize"]
```

Η σελίδα Layout

```
@using Ch07.Config.Models
@model Ch07.Config.Models.ViewModelBase
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@Model.TitlosEfarmogis</title>
</head>
<body>
<div>
    <h1> ΠΡΟΓΡΑΜΜΑΤΙΖΟΝΤΑΣ ΜΕ ASP.NET CORE </h1>
    <hr/>
    @RenderBody()
</div>
</body>
</html>
```

Η σελίδα error

```
@using Ch07.Config.Models
@model Ch07.Config.Models.ViewModelBase

<h1>ΥΠΗΡΞΕ ΛΑΘΟΣ</h1>
```

Excep

Τα ακόλουθα μοντέλα χρησιμοποιούνται:

ViewModelBase.cs

```
public class ViewModelBase
    {
        public string TitlosEfarmogis { get; set; }
    }
```

ErrorViewModel.cs

```
public class ErrorViewModel : ViewModelBase
    {
        public Exception Error { get; set; }
    }
```

Οι κάτωθι ελεγκτές:
HomeController.cs

```
  public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Throw()
        {
            throw new ArgumentException("Skopima riptwmeni exception");
        }
    }
```

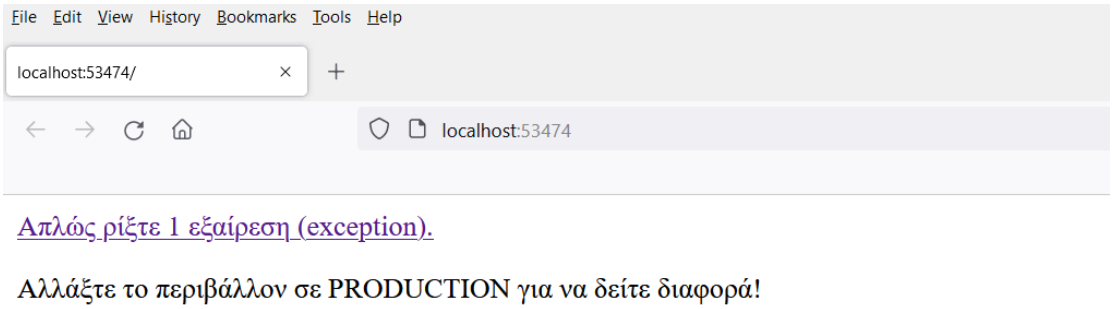
Ο επόμενος προβάλλει το error viewmodel.
AppController.cs

```
  public class AppController : Controller
    {
        public IActionResult Error([Bind(Prefix = "id")] int statusCode = 0)
        {
            // Μεταβείτε στην κατάλληλη σελίδα
            //αλλαγή (statusCode)
            //{
            //    case 404:
            //        return Redirect(...);
            //        ...
            //}

            var model = new ErrorViewModel();

           // Ανάκτηση πληροφοριών σφάλματος σε περίπτωση εσωτερικών σφαλμάτων.
            var error = HttpContext.Features.Get<IExceptionHandlerFeature>();
            if (error == null)
                return View(model);
```

```
   // ΣΗΜΑΝΤΙΚΌ!!!
  //  Ορίστε τον κωδικό κατάστασης σε 200 εάν θέλετε να δείτε τη σελίδα.
 // Διαφορετικά, ο τρέχων κωδικός κατάστασης (πιθανότατα, 500) θα σταλεί με το
σωστό HTML της προβολής που καλείται παρακάτω.
Response.StatusCode = 200;

// Χρησιμοποιήστε τις πληροφορίες που είναι αποθηκευμένες στο αντικείμενο
εξαίρεσης(exception) που εντοπίστηκε.
            model.Error = error.Error;
            return View(model);
        }
    }
```

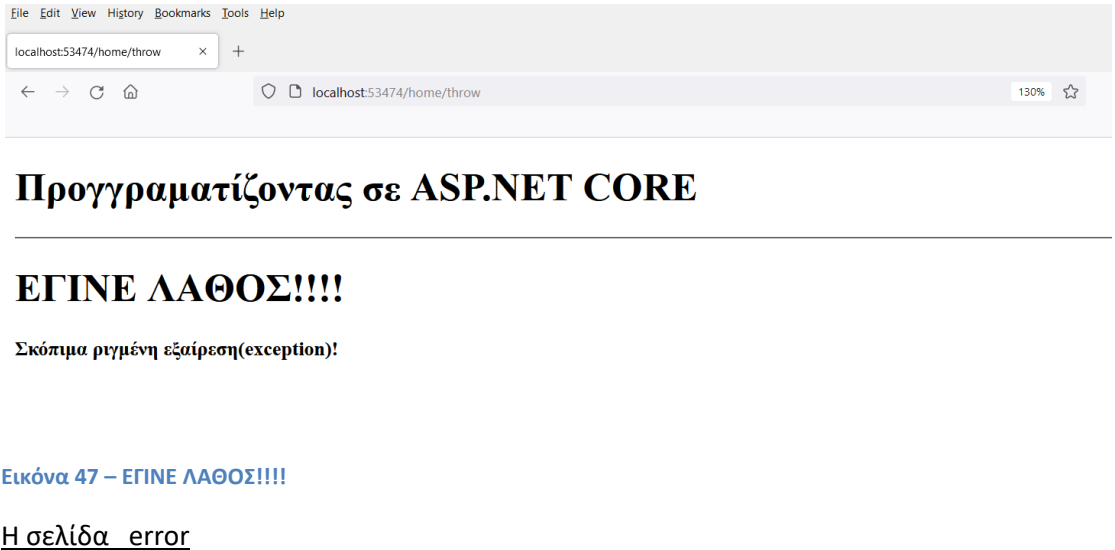
Η σελίδα home/index

```
@using Microsoft.Extensions.Configuration

<a href="@Url.Action("throw", "home")"> Απλώς ρίξτε 1 εξαίρεση (exception).
</a>


<p> Αλλάξτε το περιβάλλον σε  PRODUCTION για να δείτε διαφορά!</p>
```



Εικόνα 46 – Εμφάνιση ρίψης εξαίρεσης

Πατώντας το link επάνω στην ιστοσελίδα εμφανίζεται η επόμενη.



Εικόνα 47 – ΕΓΙΝΕ ΛΑΘΟΣ!!!!

Η σελίδα error

```
@model Ch07.Except.Models.ErrorViewModel

<h1>ΕΓΙΝΕ ΛΑΘΟΣ!!!!</h1>

@if (Model.Error != null)
{
    <h4>@Model.Error.Message</h4>
}
```

UserSecret

MyAppSecretConfig.cs

```
public class MyAppSecretConfig
    {
        public string ConnectionString { get; set; }
    }
```

Χρησιμοποιείται ο κάτωθι ελεγκτής:

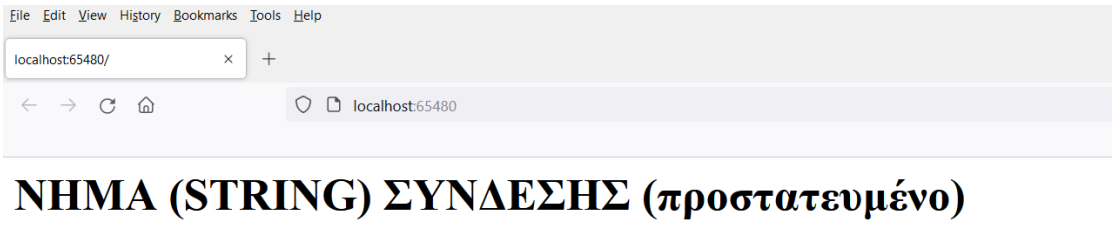
HomeController.cs

```
public class HomeController : Controller
    {
        protected MyAppSecretConfig Mystika;

        public HomeController(IOptions<MyAppSecretConfig> config, IMiscService
misc)
        {
           Mystika = config.Value;
        }

        public IActionResult Index()
        {
            ViewData["config"] = Mystika;
            return View();
        }
    }
```

Η σελίδα home/index

```
@using Ch07.UserSecrets.Common
@{
    var mystika = ViewData["config"] as MyAppSecretConfig;
}

< h1> ΝΗΜΑ (STRING) ΣΥΝΔΕΣΗΣ (προστατευμένο)</h1>
<h4>
    @if (mystika == null)
    {
        <text>Δεν διαβάζεται!</text>
    }
    else
    {
        @mystika.ConnectionString
    }
</h4>
```

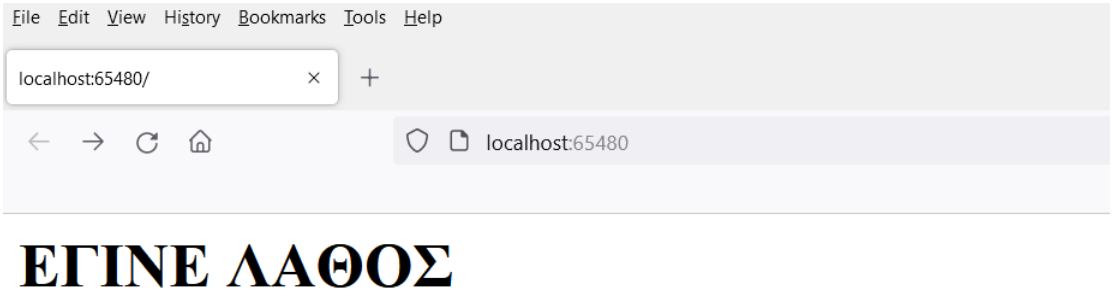
Προβάλλει:



Εικόνα 48 – ΝΗΜΑ ΣΥΝΔΕΣΗΣ

Η σελίδα error

```
<h1>ΕΓΙΝΕ ΛΑΘΟΣ</h1>
```



Εικόνα 49 – ΜΗΝΥΜΑ ΛΑΘΟΥΣ

Βιβλιογραφία

Programming ASP.NET Core, First Edition by Dino Esposito του Dino Esposito

ASP.NET Core in Action του Andrew Lock

Κεφάλαιο 8 - Ασφάλεια της εφαρμογής

Η ασφάλεια των διαδικτυακών εφαρμογών έχει πολλές πτυχές.

Πρώτα απ’ όλα, σε ένα διαδικτυακό σενάριο, η ασφάλεια σχετίζεται με την πράξη της διασφάλισης του απορρήτου των δεδομένων που ανταλλάσσονται.

Δεύτερον, σχετίζεται με την αποφυγή παραβίασης των δεδομένων διασφαλίζοντας έτσι ότι η ακεραιότητα των πληροφοριών διατηρείται καθώς ταξιδεύει από άκρο σε άκρο.

Μια άλλη πτυχή της ασφάλειας ιστού είναι η αποτροπή της εισαγωγής κακόβουλου κώδικα στην εφαρμογή που εκτελείται.

Τέλος, η ασφάλεια σχετίζεται με το κτίσιμο των εφαρμογών (και ενοτήτων μιας εφαρμογής) στις οποίες έχουν πρόσβαση μόνο επικυρωμένοι και εξουσιοδοτημένοι χρήστες.

Σε αυτό το κεφάλαιο, θα δειχθεί η εφαρμογή του ελέγχου ταυτότητας χρήστη στην ASP.NET Core και η διερεύνηση του νέου API που βασίζεται στην πολιτική για την αντιμετώπιση της εξουσιοδότησης χρήστη.

Autho

Οι κάτωθι ελεγκτές:

HomeController.cs

```
public class HomeController : Controller
    {
        // ACTIONS
        public IActionResult Index()
        {
            var model = new IndexViewModel("");
            return (IActionResult)View(model);
        }
    }
```

Εδώ γίνεται Validation των Credentials του χρήστη και Authentication.

AccountController.cs

```
public class AccountController : Controller
    {

       [HttpGet]
```

Εδώ ο κώδικας επιστρέφει την εμφάνιση της εισόδου.

```
        public IActionResult Login(LoginViewModel eisodos)

        {
            if (eisodos == null)
                eisodos = new LoginViewModel();

            string td = TempData["Login-Error"] as string;
            if (td != null)
            {
                eisodos.minimalathoys = td;
            }

            return (IActionResult)View(eisodos);
        }
```

Κάνει τον έλεγχο εισόδου και κάνει ανάλογη ανακατεύθυνση.

```
        [HttpPost]
        [ActionName("login")]
        public async Task<IActionResult> TryLogIn(LoginViewModel eisodos)
        {
            // Κάνε Επικύρωση (Validate) των credentials(διαπιστευτηρίων).
            if (!ValidateCredentials(eisodos.onomaxristi, eisodos.synthima))
            {
                TempData["Login-Error"] = "Invalid credentials";
                return RedirectToAction("login", "account");
            }
```

```
            var actualRole = (eisodos.onomaxristi == "nick" ? "Admin" :
"Guest");

            // Δημιούργησε το authentication cookie(αυθεντικοπίησης).
            var claims = new[]
            {
                new Claim("DogName", eisodos.onomaxristi),
                new Claim(ClaimTypes.Role, actualRole),
                new Claim("Picture", ""),
            };
            var identity = new ClaimsIdentity(claims,
                CookieAuthenticationDefaults.AuthenticationScheme,
                "DogName",
                ClaimTypes.Role);
            await HttpContext.SignInAsync(
                CookieAuthenticationDefaults.AuthenticationScheme,
                new ClaimsPrincipal(identity));
            return Redirect(eisodos.epistrofisUrl ?? "~/");
        }
```

Κάνει αποσύνδεση και ανακατευθύνει στην εμφάνιση οθόνης της ρίζας.

```
        public async Task<IActionResult> Logout()
        {
           await HttpContext.SignOutAsync(
                CookieAuthenticationDefaults.AuthenticationScheme);
            return Redirect("~/");
        }
```

Επιστρέφει το μήνυμα λάθους για λάθος πιστοποιητικά εισόδου.

```
        [HttpGet]
        public IActionResult Forbidden(string returnUrl)
        {
            var model = new ViewModelBase("");
            model.minimalathoys = $" {returnUrl}";
            return View(model);
        }


        public async Task T()
        {
            var props = new AuthenticationProperties
            {
                 RedirectUri = Url.Action("External", "Account")

            };
            await HttpContext.ChallengeAsync( props);
        }

        public async Task<IActionResult> External()
        {
            var authenticateResult = await
HttpContext.AuthenticateAsync("TEMP");
            var principal = authenticateResult.Principal;
            var newClaims = principal.Claims.Append(new Claim("Foo", "Foo"));
            var newIdentity = new ClaimsIdentity(newClaims,
authenticateResult.Ticket.AuthenticationScheme);
```

```
            var newPrincipal = new ClaimsPrincipal(newIdentity);

            // Χρησιμοποίησε την principal.
            // (όπως, δημιούργησε μια νεα principal με περισσότερη πληροφορία)
            await
HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme,
newPrincipal);
            await HttpContext.SignOutAsync("TEMP");
            return Redirect("~/");
        }
```

Κάνει Validate των πιστοποιητικών εισόδου.

```
        private bool ValidateCredentials(string username, string password)
        {
            if (string.IsNullOrWhiteSpace(username) ||
                string.IsNullOrWhiteSpace(password))
                return false;

            return username.Equals(password,
StringComparison.CurrentCultureIgnoreCase);
        }
    }
}
```

Επιτρέπει την πρόσβαση στην ιδιωτική περιοχή.

SecretController.cs

```
[Authorize]
    public class SecretController : Controller
    {
        // ΕΝΕΡΓΕΙΕΣ
        public IActionResult Index()
        {
            var model = new IndexViewModel("");
            return (IActionResult)View(model);
        }
```

Για τον ρόλο του διαχειριστή επιτρέπει την είσοδο.

```
        [Authorize(Roles = "Admin")]
        public IActionResult Admin1()
        {
            var model = new IndexViewModel("");
            return (IActionResult)View(model);
        }
```

Για τον ρόλο προσκεκλημένου επιτρέπει την είσοδο.

```
        public IActionResult Guest1()
        {
            var model = new IndexViewModel("");
            return (IActionResult)View(model);
        }
    }
```

Τα κάτωθι μοντέλα χρησιμοποιούνται:

ViewModelBase.cs σχετική με τα Views.

```
public class ViewModelBase
    {
        public ViewModelBase(string title = "")
        {
            titlos = title;
            minimalathoys = "";
            kwdikos = 0;
        }

        public string titlos { get; set; }
        public string minimalathoys { get; set; }
        public int kwdikos { get; set; }
    }
```

IndexViewModel.cs

```
public class IndexViewModel : ViewModelBase
    {
        public IndexViewModel(string title) : base(title)
        {

        }
    }
```

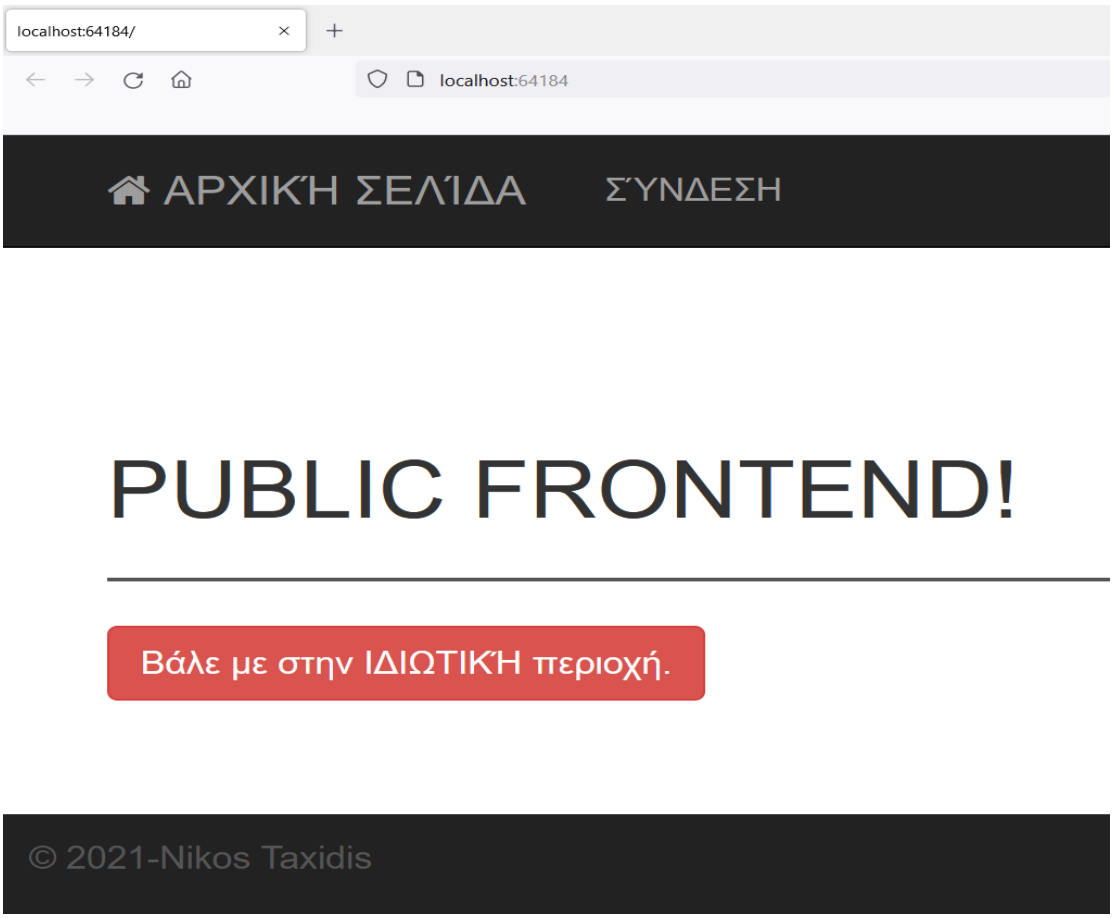
LoginViewModel.cs για το Login.

```
public class LoginViewModel : ViewModelBase
    {
        public LoginViewModel()
        {
            thymisoyme  = true;
        }

        public string onomaxristi { get; set; }
        public string synthima { get; set; }
        public bool thymisoyme { get; set; }
        public string epistrofisUrl { get; set; }
    }
```

Οι κάτωθι οθόνες χρησιμοποιούνται:

Index.cshtml

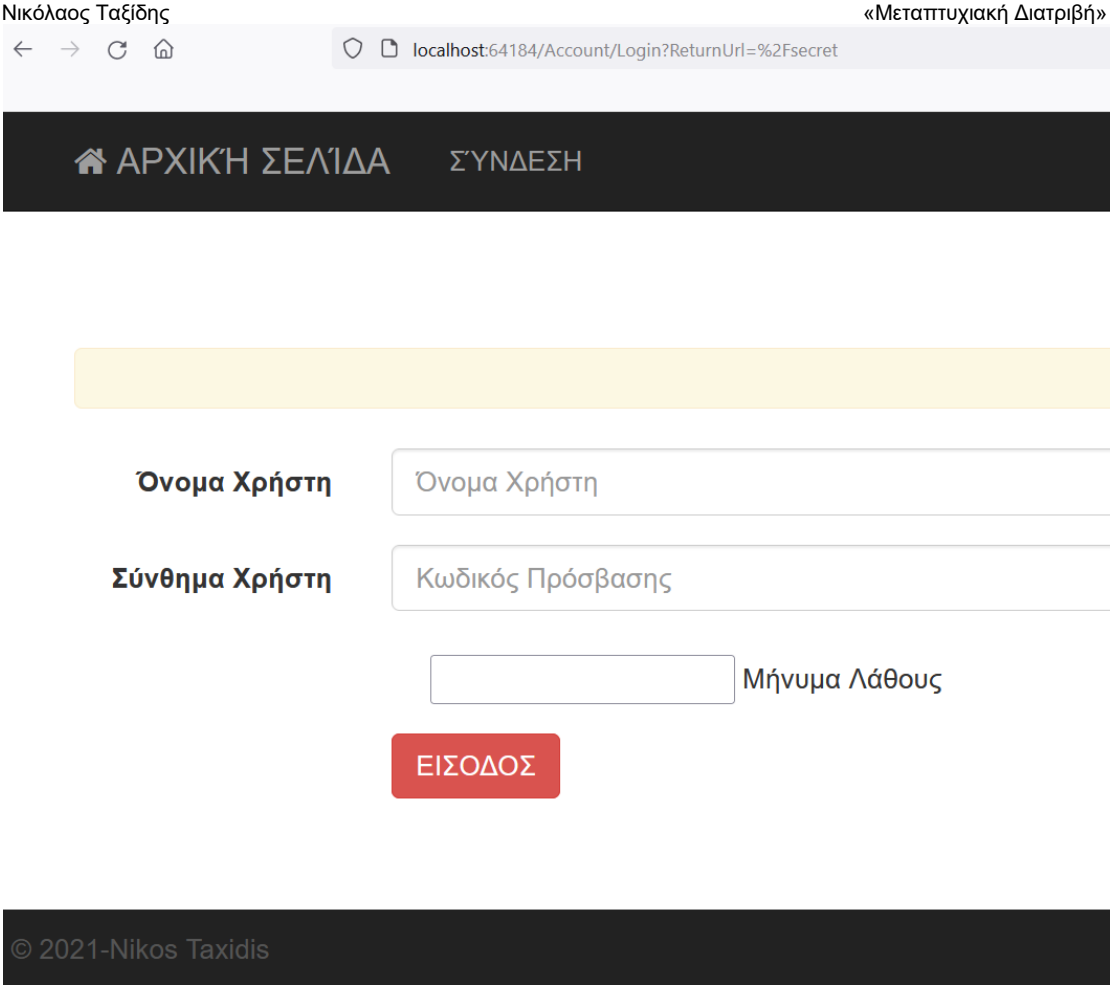
```
@model Ch08.Autho.Models.IndexViewModel

<h1>
    PUBLIC FRONTEND!
</h1>
<hr />

<a role="button" class="btn btn-danger" href="~/secret">
    Βάλε με στην ΙΔΙΩΤΙΚΉ περιοχή.
</a>
```



Εικόνα 50 – ΑΡΧΙΚΉ ΣΕΛΊΔΑ

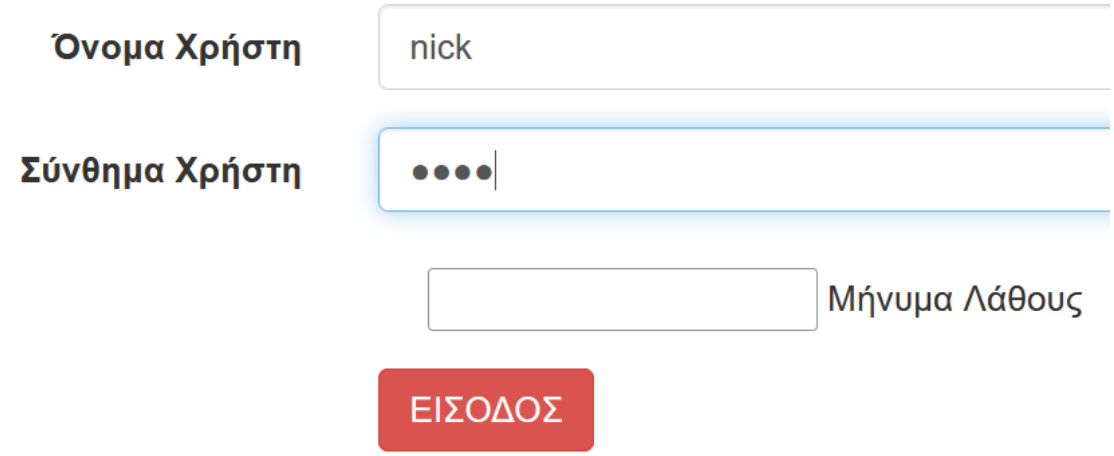
Login.cshtml

```
@model Ch08.Autho.Models.LoginViewModel

<div class="alert alert-warning">
    @Model.minimalathoys
</div>

<form class="form-horizontal" method="post" asp-controller="Account" asp-
action="login">
    <input type="hidden" name="returnurl" value="@Model.epistrofisUrl" />

    <div class="form-group">
        <label asp-for="onomaxristi" class="col-sm-2 control-label">Όνομα
Χρήστη</label>
        <div class="col-sm-10">
            <input type="text" class="form-control"
                   id="username" name="onomaxristi"
                   placeholder="Όνομα Χρήστη">
        </div>
    </div>
    <div class="form-group">
        <label asp-for="synthima" class="col-sm-2 control-label">Σύνθημα
Χρήστη</label>
        <div class="col-sm-10">
            <input type="password" class="form-control"
                   id="password" name="synthima"
                   placeholder="Κωδικός Πρόσβασης">
        </div>
    </div>
    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-10">
            <div class="checkbox">
                <label>
                                    @*
                        @Html.CheckBox("RememberMe")
        <input type="checkbox" name="RememberMe"> Remember me
        <input type="hidden" name="RememberMe" value="false" />
                                    *@
                    <input asp-for="minimalathoys"/> Μήνυμα Λάθους
                </label>
            </div>
        </div>
    </div>
    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-10">
            <button type="submit"
                    class="btn btn-danger text-uppercase">
                ΕΙΣΟΔΟΣ
            </button>
        </div>
    </div>
</form>
```



Εικόνα 51 – Φόρμα Εισαγωγής Στοιχείων Χρήστη

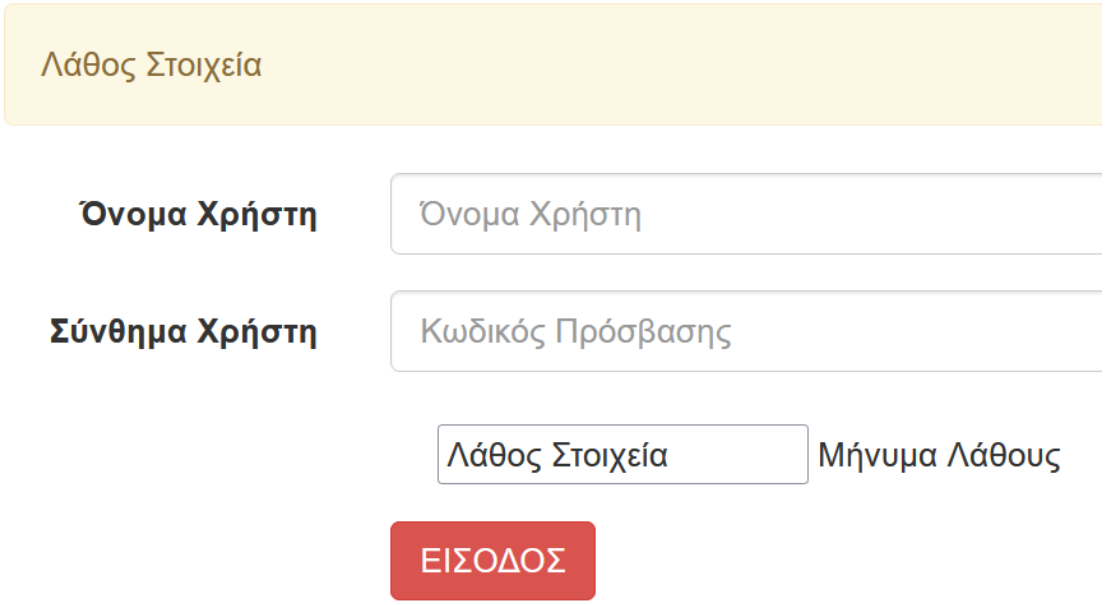
Ο χρήστης εισάγει τα στοιχεία του και πατάει ΕΙΣΟΔΟΣ.



Εικόνα 52 – Εισαγωγή Στοιχείων του Χρήστη στη Φόρμα

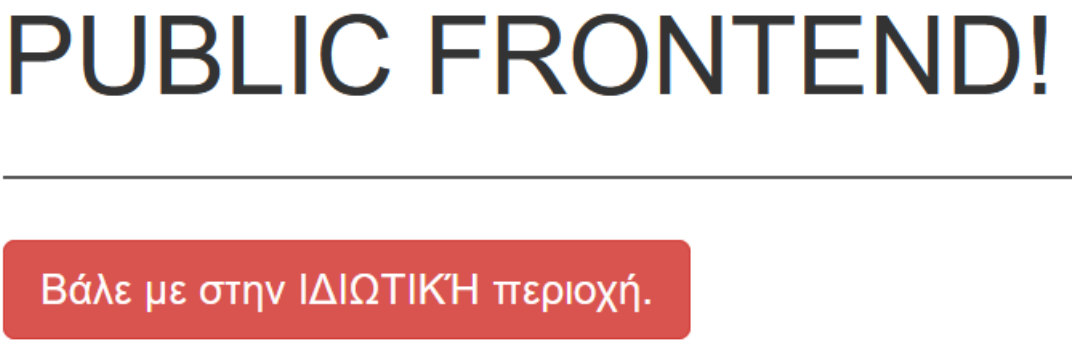
Ο χρήστης εισάγει λάθος στοιχεία και πατάει ΕΙΣΟΔΟΣ.

Έτσι εμφανίζεται μήνυμα λάθους «Λάθος Στοιχεία».



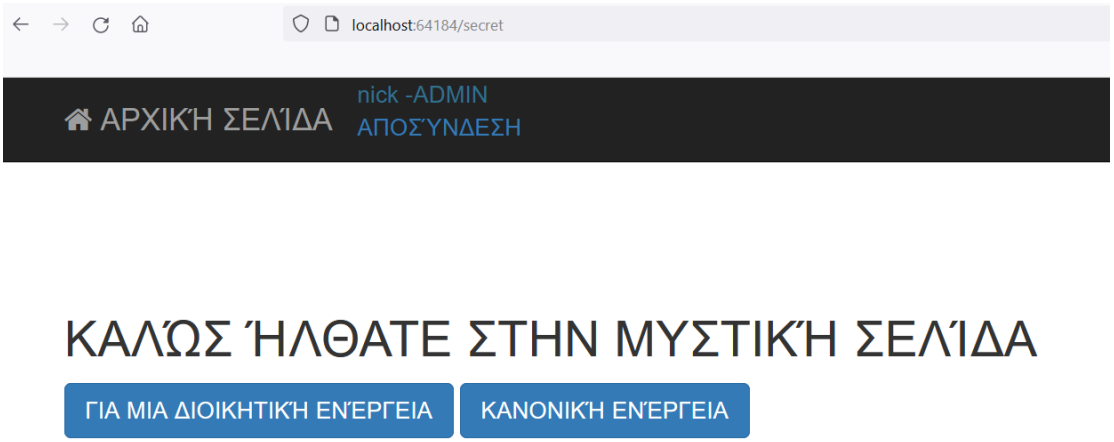
Εικόνα 53 – Ο χρήστης εισήγαγε Λάθος Στοιχεία

Ο χρήστης εισέρχεται στην ιστοσελίδα όπου είναι και ADMIN (Διαχειριστής).



Εικόνα 54 – ΑΡΧΙΚΉ ΣΕΛΊΔΑ Σύνδεσης

Ο χρήστης πατάει Βάλε με στην ΙΔΙΩΤΙΚΉ περιοχή, και εμφανίζεται η αρχική σελίδα.



Εικόνα 55 – Η Μυστική Σελίδα

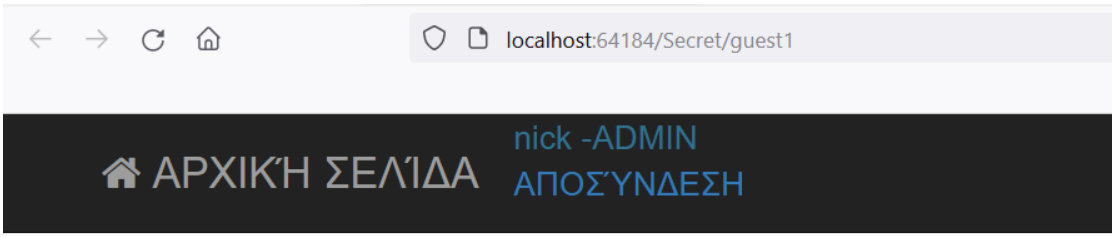
Index.cshtml

```
@model Ch08.Autho.Models.IndexViewModel

<h2>
    ΚΑΛΏΣ ΉΛΘΑΤΕ ΣΤΗΝ ΜΥΣΤΙΚΉ ΣΕΛΊΔΑ
</h2>

<a role="button"
   class="btn btn-primary"
   asp-controller="Secret"
   asp-action="admin1">
    ΓΙΑ ΜΙΑ ΔΙΟΙΚΗΤΙΚΉ ΕΝΈΡΓΕΙΑ
</a>

<a role="button"
   class="btn btn-primary"
   asp-controller="Secret"
   asp-action="guest1">
    ΚΑΝΟΝΙΚΉ ΕΝΈΡΓΕΙΑ
</a>
```

Ο χρήστης πατάει ΚΑΝΟΝΙΚΉ ΕΝΈΡΓΕΙΑ για μια απλή ενέργεια και εμφανίζεται η αρχική σελίδα GUEST1.

guest1.cshtml

```
@model Ch08.Autho.Models.IndexViewModel

<h2>
    GUEST1 ΕΝΈΡΓΕΙΑ
</h2>
```

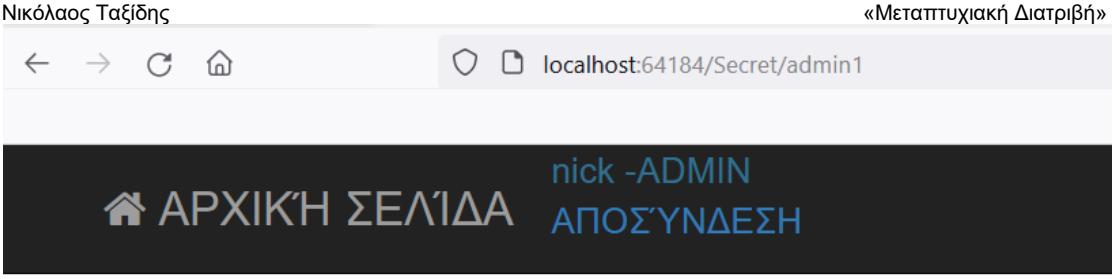


Εικόνα 56 – GUEST1 ΕΝΈΡΓΕΙΑ

Μετά ο χρήστης πατάει ΓΙΑ ΜΙΑ ΔΙΟΙΚΗΤΙΚΉ ΕΝΈΡΓΕΙΑ για μια ενέργεια που μόνο ο ADMIN μπορεί να εκτελέσει και εμφανίζεται η αρχική σελίδα ADMIN1.

admin1.cshtml

```
@model Ch08.Autho.Models.IndexViewModel

<h2 class="text-danger">
    ADMIN1 ΕΝΈΡΓΕΙΑ
</h2>
```



ADMIN1 ΕΝΈΡΓΕΙΑ

Εικόνα 57 - ADMIN1 ΕΝΈΡΓΕΙΑ

Ο χρήστης μπορεί ανά πάσα στιγμή να αποσυνδεθεί πατώντας το κουμπί ΑΠΟΣΥΝΔΕΣΗ.

_Layout.cshtml

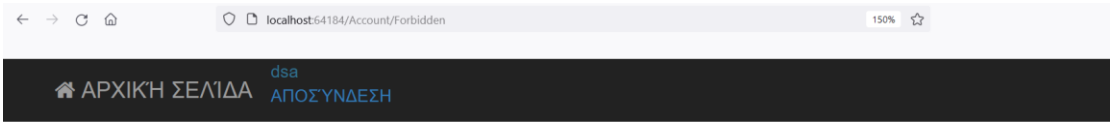
```
@model Ch08.Autho.Models.ViewModelBase
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="~/Content/Styles/font-awesome.min.css" />
    <link rel="stylesheet" href="~/Content/Styles/bootstrap.min.css" />
    <link rel="stylesheet" href="~/Content/Styles/Site/site.css" />
    <script src="~/Content/Scripts/jquery.min.js"></script>
    <script src="~/Content/Scripts/tether.min.js"></script>
    <script src="~/Content/Scripts/bootstrap.min.js"></script>
    <title>@Model.titlos</title>
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a href="@Url.Action("index", "home")" class="navbar-brand">
                    <i class="fa fa-home"></i>
                    ΑΡΧΙΚΉ ΣΕΛΊΔΑ
                </a>
```

```
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li>
                        @if (this.User.Identity.IsAuthenticated)
                        {
                            <span class="text-info">@User.Identity.Name</span>
      <span class="text-info">@(User.IsInRole("Admin") ?" -ADMIN" :"")</span>
                            <div>
             <a href="@Url.Action("logout", "account")">ΑΠΟΣΥΝΔΕΣΗ</a>
                            </div>
                        }
                        else
                        {
                    <a href="@Url.Action("login", "account")">ΣΥΝΔΕΣΗ</a>
                        }
                    </li>
                </ul>
            </div>
        </div>
    </div>
    <div class="container">
        @RenderBody()
        <footer>
            <p>&copy; 2021-Nikos Taxidis</p>
        </footer>
    </div>
</body>
</html>
```

Error.cshtml

```
@using Ch08.Autho.Models
@model Ch08.Autho.Models.ViewModelBase
<h1>Λάθος <span>@(Model.kwdikos >0 ? Model.kwdikos.ToString() :"")</span></h1>
```

Σε περίπτωση που εισέλθει ένας χρήστης που δεν είναι διαχειριστής τότε σε περίπτωση που προσπαθήσει να εισέλθει στην σελίδα της ενέργειας ADMIN1, τότε θα του εμφνιστεί μήνυμα λάθους.



Εικόνα 58 – ΜΗΝΥΜΑ ΛΑΘΟΥΣ

Forbidden.cshtml

```
@model Ch08.Autho.Models.ViewModelBase

<h1>ΔΕΝ ΔΙΚΑΙΟΎΣΤΕ ΝΑ ΜΠΕΊΤΕ ΣΤΗΝ ΙΣΤΟΣΕΛΊΔΑ!</h1>
<h4>@Model.minimalathoys</h4>
```

Authorz

Τα παρακάτω μοντέλα χρησιμοποιούνται:

ViewModelBase.cs

```
  public class ViewModelBase
    {
        public ViewModelBase(string title = "")
        {
            titlos = title;
            minimalathoys = "";
            kwdikos = 0;
        }

        public string titlos { get; set; }
        public string minimalathoys { get; set; }
        public int kwdikos { get; set; }
    }
```

LoginViewModel.cs

```
public class LoginViewModel : ViewModelBase
    {
        public LoginViewModel()
        {
            thymisoyme = true;
        }

        public string onomaxristi { get; set; }
        public string synthima { get; set; }
        public bool thymisoyme { get; set; }
        public string epistrofisUrl { get; set; }
    }
```

IndexViewModel.cs

```
public class IndexViewModel : ViewModelBase
    {
        public IndexViewModel(string title) : base(title)
        {

        }
    }
```

Και οι κάτωθι ελεγκτές:

HomeController.cs

```
public class HomeController : MyControllerBase
    {
        private ILogger Logger { get; set; }

        public HomeController(IOptions<GlobalConfig> config)
        {
            Configuration = config.Value;
        }
```

Επιστρέφει την ιδιωτική περιοχή.

```
        // ΕΝΕΡΓΕΙΕΣ
        public IActionResult Index()
        {
            var model = new IndexViewModel(Configuration.titlos);
            return View(model);
        }
    }
```

Ο ελεγκτής παρακάτω διαχειρίζεται την διαδικασία εισόδου Login

AccountController.cs

```
public class AccountController : MyControllerBase
    {
        public AccountController(IOptions<GlobalConfig> config)
        {
            Configuration = config.Value;
        }
```

Εμφανίζει την οθόνη εισόδου:

```
        [HttpGet]
        public IActionResult Login(LoginViewModel data)
        {
            if (data == null)
                data = new LoginViewModel();

            var td = TempData["Login-Error"] as string;
            if (td != null)
            {
                data.minimalathoys = td;
            }

            return View(data);
        }
```

Ελέγχει την είσοδο στην ιδιωτική περιοχή.

```
        [HttpPost]
        [ActionName("login")]
        public async Task<IActionResult> TryLogIn(LoginViewModel data)
        {
            // Κάνε validate(επικύρωση) των credentials.
            if (!ValidateCredentials(data.onomaxristi, data.synthima))
            {
                Debug.WriteLine("not validated ......");
```

```
                TempData["Login-Error"] = "Invalid credentials";
                return RedirectToAction("login", "account");
            }

            var actualRole = (data.onomaxristi == "nick" ? "Admin" : "Guest");

            // Δημιούργησε το authentication cookie.
            var claims = new[] {
                new Claim(ClaimTypes.Name, data.onomaxristi),
                new Claim(ClaimTypes.Role, actualRole) };

            if (data.onomaxristi == "leo")
            {
                claims = new[] {
                new Claim(ClaimTypes.Name, data.onomaxristi),
                new Claim(ClaimTypes.Role, "Admin"),
                new Claim("Hair", "Brown") };
            }

            var identity = new ClaimsIdentity(claims,
                CookieAuthenticationDefaults.AuthenticationScheme,
                ClaimTypes.Name,
                ClaimTypes.Role);
            await HttpContext.SignInAsync(
                CookieAuthenticationDefaults.AuthenticationScheme,
                new ClaimsPrincipal(identity));

            return Redirect(data.epistrofisUrl ?? "/");
        }
```

Δείχνει την εμφάνιση οθόνης μετά την αποσύνδεση.

```
        public async Task<IActionResult> Logout()
        {
           await HttpContext.SignOutAsync(
                CookieAuthenticationDefaults.AuthenticationScheme);
            return Redirect("/");
        }
```

Δείχνει την εμφάνιση οθόνης μετά από λάθος διαπιστευτήρια.

```
        [HttpGet]
        public IActionResult Forbidden(string returnUrl)
        {
            var model = new ViewModelBase(Configuration.titlos) {minimalathoys
= $"from {returnUrl}"};
            return View(model);
        }
```

Ελέγχει την εγκυρότητα διαπιστευτηρίων εισόδου.

```
        private bool ValidateCredentials(string username, string password)
        {

            Debug.WriteLine("σύνδεση......");
            if (string.IsNullOrWhiteSpace(username) ||
                string.IsNullOrWhiteSpace(password))
            {
                Debug.WriteLine("άδειο χωρίς σύνδεση......");
```

```
                return false;
            }
            else
            {

                Debug.WriteLine("δεν είναι άδειο υπάρχει η σύνδεση......");


            }
            return username.Equals(password,
StringComparison.CurrentCultureIgnoreCase);
        }
    }
```

MyControllerBase.cs

```
public class MyControllerBase : Controller
    {
        protected GlobalConfig Configuration { get; set; }
    }
```

Ο ελεγκτής παρακάτω διαχειρίζεται την πρόσβαση στις διάφορες περιοχές εργασιών:

SecretController.cs

```
[Authorize(Policy = "MustBeRegisteredUser")]
    public class SecretController : MyControllerBase
    {
        private readonly IAuthorizationService _egkrisi;

        public SecretController(
            IOptions<GlobalConfig> config,
            IAuthorizationService egkrisi)
        {
            Configuration = config.Value;
            _egkrisi = egkrisi;
        }


        public IActionResult Index()
        {
            var model = new IndexViewModel(Configuration.titlos);
            return View(model);
        }

        [Authorize(Policy = "MustBeAdministrator")]
        public IActionResult Admin1()
        {
            var model = new IndexViewModel(Configuration.titlos);
            return View(model);
        }

        [Authorize(Policy = "MustBeAdministratorWithBlondHair")]
        public IActionResult Admin2()
        {
            var model = new IndexViewModel(Configuration.titlos);
```

```
            return View(model);
        }

        public IActionResult Guest1()
        {
            var model = new IndexViewModel(Configuration.titlos);
            return View(model);
        }

        public IActionResult Guest2()
        {
            // ΠΡΟΣΟΧΗ: Η μέθοδος έχει πολλά overloads (αντί για policy name,
            //       μπορείς να περάσεις στη λίστα των requirements, ένα single
requirement, ένα policy object)
            if (_egkrisi
                .AuthorizeAsync(User, "GuestWithSpecialInitials")
                .Result
                .Succeeded)
            {
                var model = new IndexViewModel(Configuration.titlos);
                return View(model);
            }

            return new ChallengeResult();
        }
    }
```

Ακολουθούν οι html:

Index.cshtml

```
@using Ch08.Authorz.Common
@using Microsoft.Extensions.Options;
@inject IOptions<GlobalConfig> Settings

<h1>
    PUBLIC FRONTEND!
    @Settings.Value.titlos
</h1>
<hr />

<a role="button" class="btn btn-danger" href="~/secret">
    Πάρε με στην ιδιωτική περιοχή!
 </a>
```

Login.cshtml

```
@using Ch08.Authorz.Models
@model Ch08.Authorz.Models.LoginViewModel

<div class="alert alert-warning">
    @Model.minimalathoys
</div>
```

```
<form class="form-horizontal" method="post" asp-controller="Account" asp-
action="login">
    <input type="hidden" name="returnurl" value="@Model.epistrofisUrl" />

    <div class="form-group">
        <label asp-for="onomaxristi" class="col-sm-2 control-label">Όνομα
Χρήστη</label>
        <div class="col-sm-10">
            <input type="text" class="form-control"
                   id="onomaxristi" name="onomaxristi"
                   placeholder="Βάλε όνομα">
        </div>
    </div>
    <div class="form-group">
        <label asp-for="synthima" class="col-sm-2 control-label"> Συνθηματικό
Χρήστη</label>
        <div class="col-sm-10">
            <input type="password" class="form-control"
                   id="synthima" name="synthima"
                   placeholder="Βάλε συνθηματικό">
        </div>
    </div>
    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-10">
            <div class="checkbox">
                <label>
                    @*<input type="checkbox"> Remember me*@
                    <input asp-for="thymisoyme" /> Να με θυμάσαι!
                </label>
            </div>
        </div>
    </div>
    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-10">
            <button type="submit"
                    class="btn btn-danger text-uppercase">
           ΕΙΣΟΔΟΣ
            </button>
        </div>
    </div>
</form>
```

Forbidden.cshtml

```
@using Ch08.Authorz.Models
@model Ch08.Authorz.Models.ViewModelBase

<h1>ΔΕΝ ΈΧΕΙΣ ΤΗΝ ΔΙΚΑΙΟΔΟΣΊΑ ΝΑ ΔΕΙΣ ΑΥΤΉΝ ΤΗΝ ΣΕΛΊΔΑ!</h1>
<h4>@Model.minimalathoys</h4
```

/secret/Index.cshtml

```
@model Ch08.Authorz.Models.IndexViewModel
```

```
<h2>
    ΚΑΛΩΣ ΗΡΘΑΤΕ ΣΤΗΝ ΜΥΣΤΙΚΗ ΣΕΛΙΔΑ!
</h2>

<a role="button"
   class="btn btn-primary"
   asp-controller="Secret"
   asp-action="admin1">
    Προσπάθησε να μπεις αν είσαι ADMIN.
</a>

<a role="button"
   class="btn btn-primary"
   asp-controller="Secret"
   asp-action="admin2">
    Προσπάθησε να μπεις αν είσαι ΚΑΦΕ ADMIN.
</a>

<a role="button"
   class="btn btn-primary"
   asp-controller="Secret"
   asp-action="guest1">
    Απλή εργασία.
</a>

<a role="button"
   class="btn btn-primary"
   asp-controller="Secret"
   asp-action="guest2">
    Εργασία μόνο για ειδικούς χρήστες.
</a>
```

Secret/admin1.cshtml

```
@model Ch08.Authorz.Models.IndexViewModel

<h2 class="text-danger">
    ADMIN1 εργασία
</h2>
```

Secret/admin2.cshtml

```
@model Ch08.Authorz.Models.IndexViewModel

<h2 class="text-warning">
    ΑΥΤΗ Η ΣΕΛΙΔΑ ΕΙΝΑΙ ΜΟΝΟ ΓΙΑ ΚΑΦΕ ADMINS
</h2>
<h4>
```

```
    @{
        var claim = User.FindFirst("Hair").Value;
    }
    @claim
</h4>
```

Secret/guest1.cshtml

```
@model Ch08.Authorz.Models.IndexViewModel

<h2>
    GUEST1 εργασία
</h2>
```

Secret/guest2.cshtml

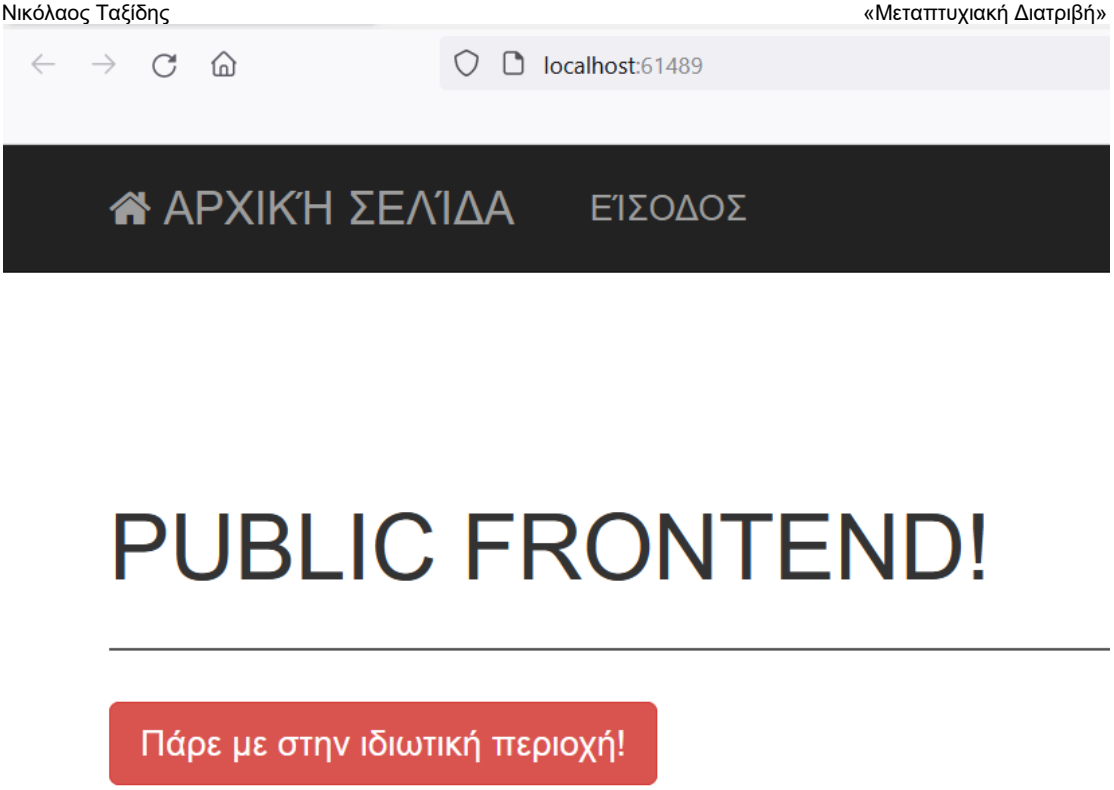
```
@model Ch08.Authorz.Models.IndexViewModel

<h2>
    GUEST2 εργασία
</h2>
```

_Layout.cshtml

```
@model Ch08.Authorz.Models.ViewModelBase

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="~/Content/Styles/font-awesome.min.css" />
    <link rel="stylesheet" href="~/Content/Styles/bootstrap.min.css" />
    <link rel="stylesheet" href="~/Content/Styles/Site/site.css" />
    <script src="~/Content/Scripts/jquery.min.js"></script>
    <script src="~/Content/Scripts/tether.min.js"></script>
    <script src="~/Content/Scripts/bootstrap.min.js"></script>
    <title>@Model.titlos</title>
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a href="@Url.Action("index", "home")" class="navbar-brand">
                    <i class="fa fa-home"></i>
                    ΑΡΧΙΚΉ ΣΕΛΊΔΑ
                </a>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
```

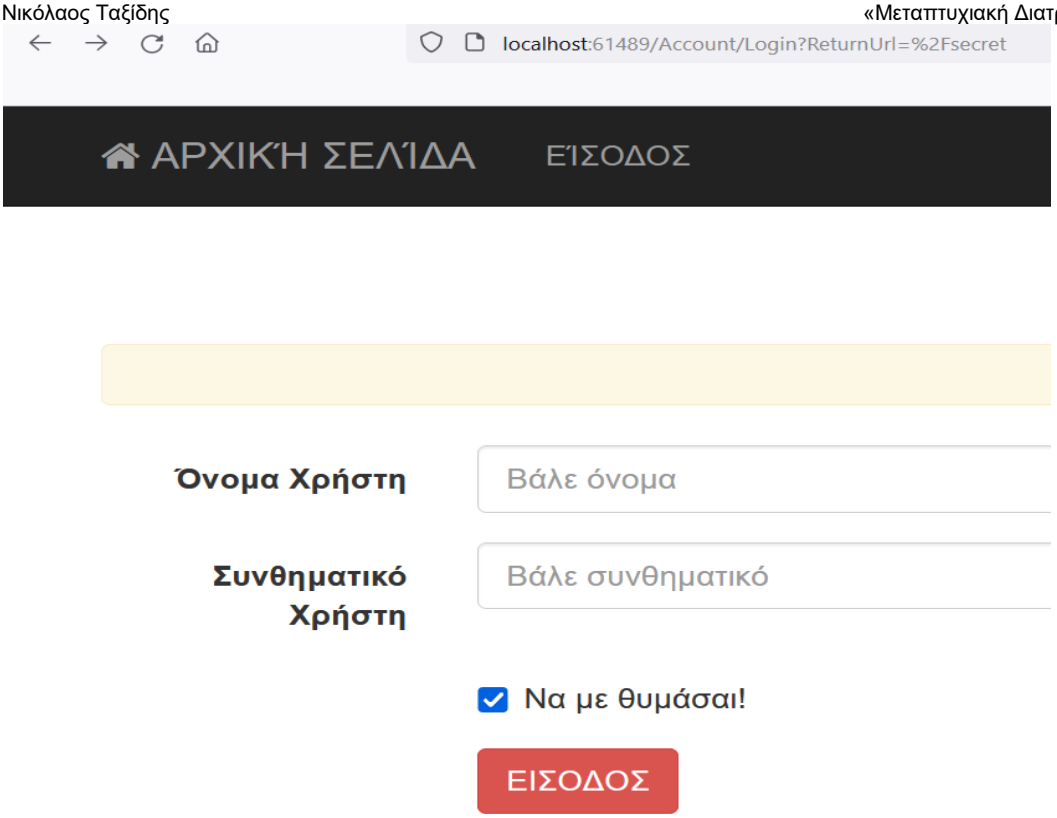
```
                    <li>
                        @if (this.User.Identity.IsAuthenticated)
                        {
                            <span class="text-info">@User.Identity.Name</span>
        <span class="text-info">@(User.IsInRole("Admin") ?" -ADMIN" :"")</span>
                            <div>
                             <a href="@Url.Action("logout", "account")">ΈΞΟΔΟΣ</a>
                            </div>
                        }
                        else
                        {
                         <a href="@Url.Action("login", "account")">ΕΊΣΟΔΟΣ</a>
                        }
                    </li>
                </ul>
            </div>
        </div>
    </div>

    <div class="container">
        @RenderBody()

        <footer>
            <p>&copy; 2021- NICK TAXIDIS</p>
        </footer>
    </div>
</body>
</html>
```

Αρχική σελίδα(Index.cshtml):



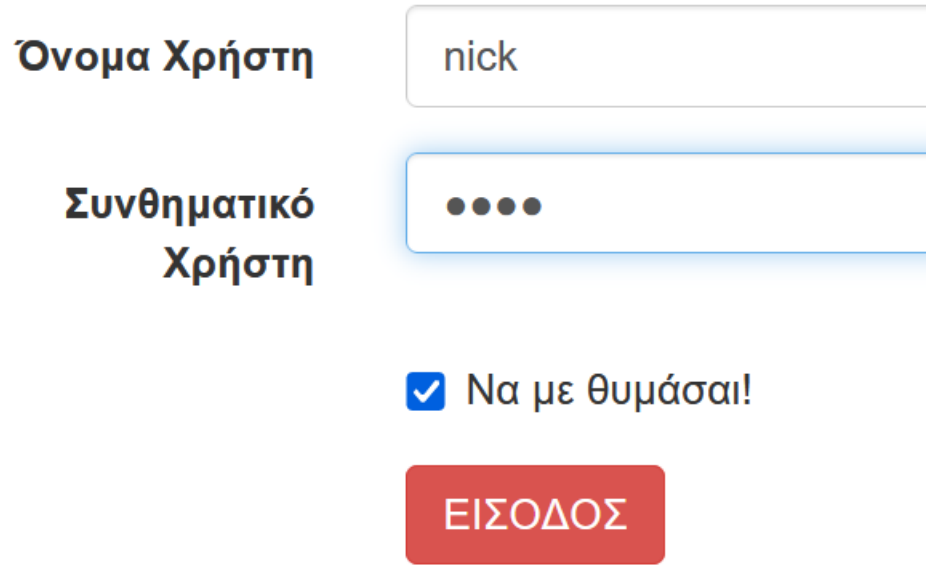
Εικόνα 59 – ΑΡΧΙΚΉ ΣΕΛΊΔΑ

Πατώντας το κουμπί : `Πάρε με στην ιδιωτική περιοχή!` στον χρήστη εμφανίζεται :



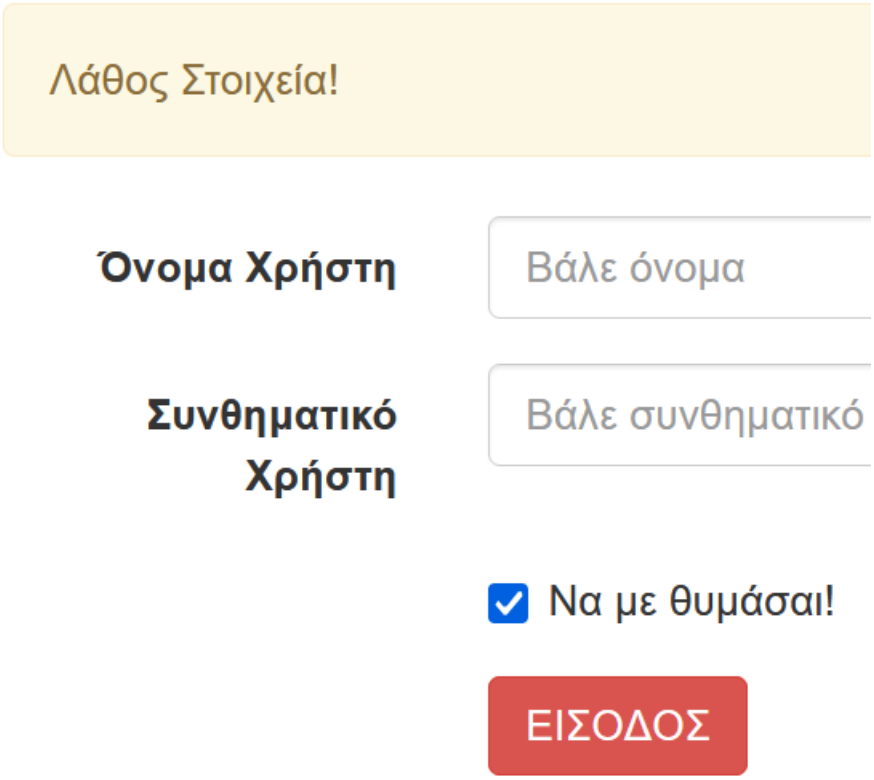
Εικόνα 60 – Φόρμα Εισαγωγής Στοιχείων Χρήστη

Ο χρήστης πρέπει να βάλει τα στοιχεία του.



Εικόνα 61 – Εισαγωγή Λάθος Στοιχείων από Χρήστη

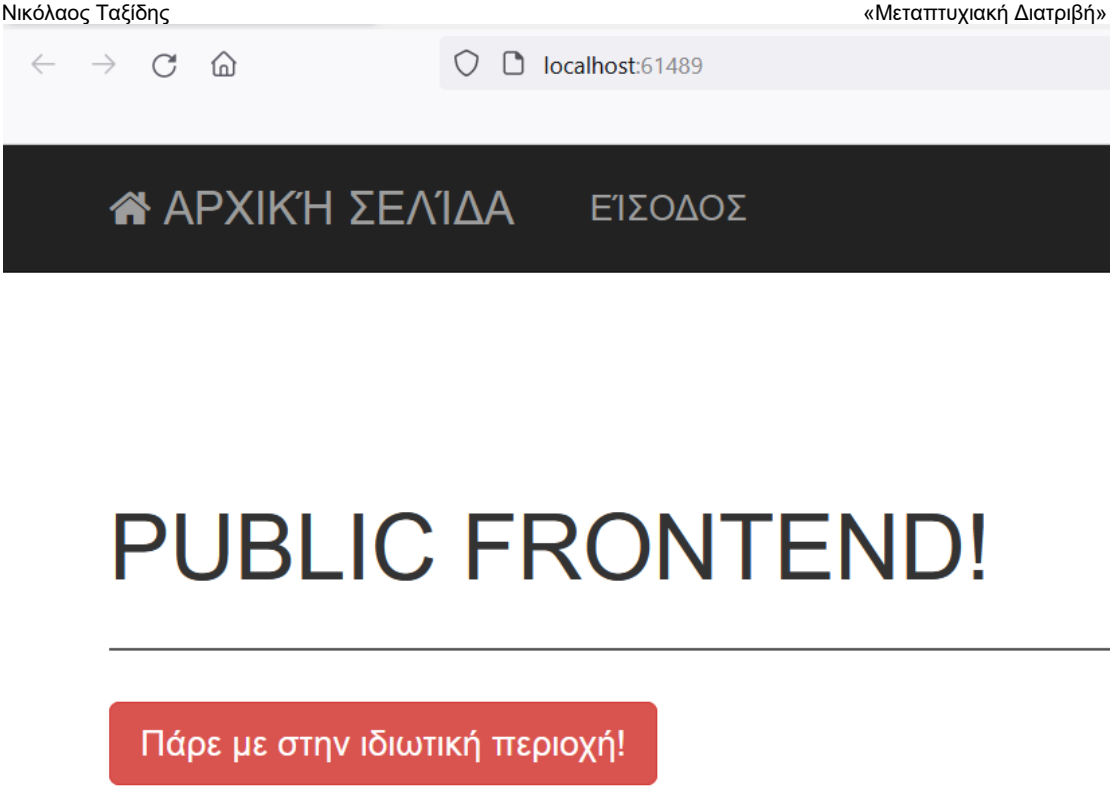
Ο χρήστης εισήγαγε λάθος στοιχεία.



Εικόνα 62 – Ο Χρήστης εισήγαγε Λάθος Στοιχεία

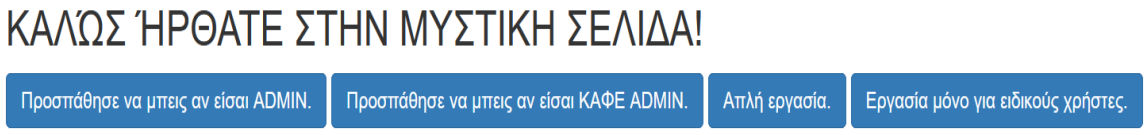
Όταν εισήγαγε τα σωστά στοιχεία του εμφανίστηκε η σελίδα:

Μετά την είσοδο έχουμε:



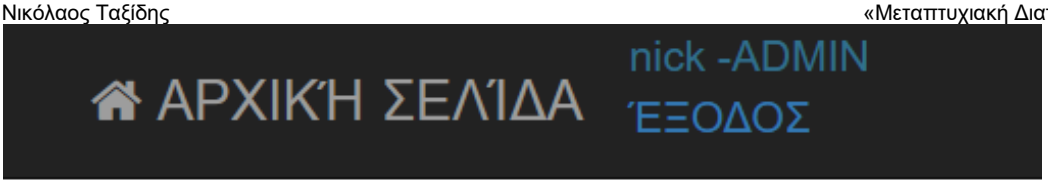
Εικόνα 63 – ΑΡΧΙΚΉ ΣΕΛΊΔΑ του Χρήστη

Οπότε ο χρήστης πατά το κουμπί : `Πάρε με στην ιδιωτική περιοχή!` στον χρήστη εμφανίζεται η Μυστική Σελίδα:



Εικόνα 64 – ΚΑΛΏΣ ΉΡΘΑΤΕ ΣΤΗΝ ΜΥΣΤΙΚΗ ΣΕΛΙΔΑ

Επιλέγοντας το πρώτο κουμπί έχουμε:

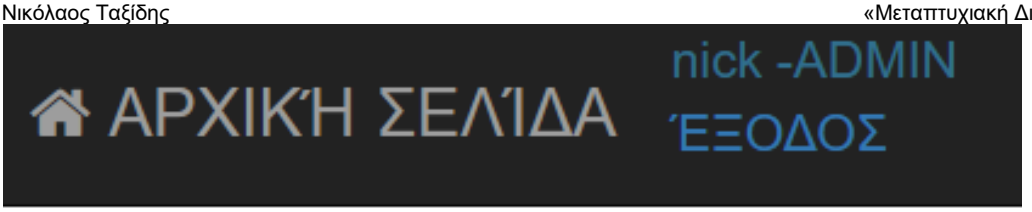


ADMIN1 ΕΡΓΑΣΊΑ

Εικόνα 65 – ADMIN1 ΕΡΓΑΣΙΑ

Επειδή ο χρήστης είναι Admin (Διαχειριστής) μπορεί να εισέλθει στην συγκεκριμένη σελίδα.

Επιλέγοντας το τρίτο κουμπί έχουμε:



Εικόνα 66 – GUEST1 ΕΡΓΑΣΊΑ

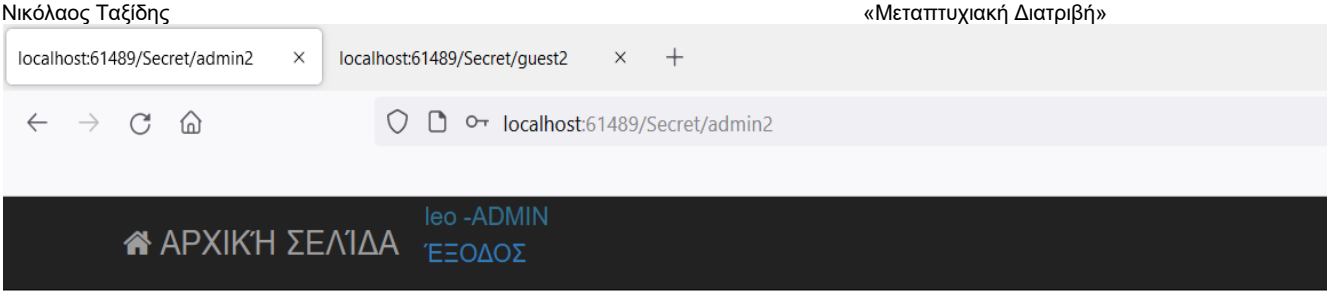
Επιλέγοντας το 4ο κουμπί έχουμε:

Εικόνα 67 – GUEST2 ΕΡΓΑΣΊΑ

Επιλέγοντας το 2ο κουμπί έχουμε, αν ο χρήστης δεν είναι εξουσιοδοτημένος:

Εικόνα 68 – ΜΗΝΥΜΑ ΛΑΘΟΥΣ

Επιλέγοντας το 2ο κουμπί έχουμε, μόνο αν ο χρήστης είναι εξουσιοδοτημένος:



ΑΥΤΗ Η ΣΕΛΙΔΑ ΕΙΝΑΙ ΜΟΝΟ ΓΙΑ ΚΑΦΕ ADMINS

Brown

Εικόνα 69 – ΣΕΛΙΔΑ ΚΑΦΕ ADMIN

Βιβλιογραφία

Programming ASP.NET Core, First Edition by Dino Esposito του Dino Esposito

ASP.NET Core in Action του Andrew Lock

Κεφάλαιο 9 - Πρόσβαση σε Δεδομένα Εφαρμογής

Σε αυτό το κεφάλαιο, θα γίνει μια προσπάθεια κατανόησης της πρόσβασης δεδομένων σε σύγχρονες εφαρμογές ιστού και ανάπτυξη ενός σχετικά γενικού μοτίβου για ένα αποτελεσματικό back-end εφαρμογής.

Η Ανθεκτικότητα (Persistence) είναι σαφώς μέρος του back-end της εφαρμογής και, ανεξάρτητα από το επίπεδο αφαίρεσης(abstraction) που ο κάθε χρήστης σκοπεύει να έχει στην κορυφή του πλάνου του, η Ανθεκτικότητα είναι προφανώς φτιαγμένη από ένα πλαίσιο για την ανάγνωση και την εγγραφή δεδομένων από κάποιο ανθεκτικό αποθηκευτικό χώρο, πιθανώς τοποθετημένο σε έναν απομακρυσμένο διακομιστή σε κάποια πλατφόρμα cloud .

Όλο και περισσότερες εφαρμογές χρησιμοποιούν καταστήματα NoSQL και αρκετές χρησιμοποιούν δύο ξεχωριστές στοίβες για την αντιμετώπιση δεδομένων με εντολή και ερώτημα(query). Η πρόσβαση δεδομένων στο πλαίσιο μιας σύγχρονης εφαρμογής ASP.NET Core δεν είναι απλώς ένα θέμα διερεύνησης των λεπτομερειών μιας βιβλιοθήκης πρόσβασης δεδομένων, αλλά ο σχεδιασμός πρώτα από οτιδήποτε άλλο.

Επομένως, θα γίνει μια προσπάθεια αποτύπωσης της ουσίας ενός μοτίβου γενικού σκοπού για το back-end της εφαρμογής, εμπνευσμένο από το γνωστό μοτίβο Layered Architecture του DDD(Domain-driven Design) και στη συνέχεια θα εξεταστούν οι επιλογές πρόσβασης δεδομένων για πραγματικές αναγνώσεις και

AdoNet

Εδώ γίνεται η επεξεργασία της βάσης.
dbcreate.cs

```
public class dbcreate
    {
        public bool CheckDatabase(string databaseName)
        {
            // Αφού είναι string, χρησιμοποίησε var
            var connString = "Data Source=(localdb)\\MSSQLLocalDB;Initial
Catalog=master;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False";
   // Προσοχή: Είναι καλύτερα να πάρεις connection string από το αρχείο config.

            var cmdText = "select count(*) from master.dbo.sysdatabases where
name=@database";
            using (var sqlConnection = new SqlConnection(connString))
            {
                using (var sqlCmd = new SqlCommand(cmdText, sqlConnection))
                {
Χρησιμοποιούνται τις parameters για να ασφαλίσεις από την Sql

InjectionsqlCmd.Parameters.Add("@database",
System.Data.SqlDbType.NVarChar).Value = databaseName;

                    // Άνοιξε την connection όσο πιο αργά γίνεται.
                    sqlConnection.Open();
// Η count(*) θα επιστρέφει πάντοτε έναν int, etsi έτσι είναι ασφαλής η χρήση
της Convert.ToInt32
                    return Convert.ToInt32(sqlCmd.ExecuteScalar()) == 1;
                }
            }

        }

        public void dbcreratefunc()
        {
            if (!CheckDatabase("ProgCore"))
            {
                string myConnectionString = "Data
Source=(localdb)\\MSSQLLocalDB;Initial Catalog=master;Integrated
Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False"; //ConnectionStrings.progconn;

                using (var connection = new SqlConnection(myConnectionString))
                {
                    connection.Open();
                    var command = connection.CreateCommand();
                    command.CommandText = "CREATE DATABASE ProgCore";
                    command.ExecuteNonQuery();
                    connection.Close();
                }
            }
        }
```

```
        public void dbcreatetables()
        {
            string myConnectionString = ConnectionStrings.progconn;

            using (SqlConnection con = new SqlConnection(myConnectionString))
            {

                try
                {
                    //
                    // Άνοιξε την SqlConnection.
                    //
                    con.Open();
                    //
    // Παρακάτω χρησιμοποιείται μια SqlCommand βασισμένη στην SqlConnection.
                    //
                    using (SqlCommand command = new SqlCommand("CREATE TABLE
Customers(FirstName char(50),LastName char(50),Country char(25), phone
char(50),address char(50));", con))
                        command.ExecuteNonQuery();
                    con.Close();
                }
                catch (Exception ex)
                {
                   Debug.WriteLine(ex.Message);
                }
            }


        }



        public void dbcreaterecs(int m)
        {

            string myConnectionString = ConnectionStrings.progconn;

            using (SqlConnection con = new SqlConnection(myConnectionString))
            {

                try
                {    //
                     // Άνοιξε την SqlConnection.
                    //
                    con.Open();
```

```
      // Παρακάτω χρησιμοποιείται μια SqlCommand βασισμένη στην SqlConnection.
```

```
                    for (int j = 0; j < m; j++)
                    {
                        string fname = "name" + j.ToString();
                        string lname = "lastname" + j.ToString();
                        string country = "greece";
                        string phonec = "210-888" + j.ToString();
                        string addressc="address"+j.ToString();
                        string query = "INSERT INTO Customers (FirstName,
LastName, Country, phone, address) VALUES ( '" + fname + "', '" + lname + "',
'" + country + "', '" + phonec + "', '" + addressc +"' )";

                        SqlCommand myCommand = new SqlCommand(query, con);

                        myCommand.ExecuteNonQuery();
                    }
                    con.Close();
                }
                catch (Exception ex)
                {
                    Debug.WriteLine(ex.Message);
                }
            }
        }
    }
```

Ακολουθεί το HomeService

HomeService.cs

```
public class HomeService
    {
        private readonly SomeRepository _repo;

        public HomeService(SomeRepository repo)
        {
            _repo = repo;
        }

        public HomeViewModel GetHomeViewModel()
        {
            dbcreate mydb = new dbcreate();
            if (!mydb.CheckDatabase("ProgCore"))
            {

                mydb.dbcreratefunc();

                mydb.dbcreatetables();
                mydb.dbcreaterecs(20);

            }
            var model = new HomeViewModel {eggrafes = _repo.GetRecords()};
            return model;
        }
    }
```

Η SomeRepository κάνει μια διαχείρηση της βάσης.

SomeRepository.cs

```
  public class SomeRepository
    {
        public DataTable GetRecords()
        {
            var conn = new SqlConnection {ConnectionString =
ConnectionStrings.progconn };
            Debug.WriteLine(ConnectionStrings.progconn);
            conn.Open();
            var cmd =
                new SqlCommand("SELECT TOP 4 FirstName, LastName, Country FROM
customers", conn)
                {
                    CommandType = CommandType.Text
                };
            var table = new DataTable("Results");
            var adapter = new SqlDataAdapter(cmd);
            adapter.Fill(table);
            conn.Close();
            //conn.Open();

            return table;
        }
    }
```

Ακολουθεί ο κύριος ελεγκτής που επιστρέφει τις εγγραφές πελατών στην ιστοσελίδα.

HomeController.cs

```
  public class HomeController : Controller
    {
        private readonly HomeService _ypiresia;
        public HomeController(HomeService ypiresia)
        {
            _ypiresia = ypiresia;
        }

        public IActionResult Index()
        {
            var model = _ypiresia.GetHomeViewModel();
            return View(model);
        }
    }
```

HomeViewModel .cs

```
public class HomeViewModel
    {
        public DataTable eggrafes { get; set; }
    }
```

Ακολουθεί το Index.cshtml

```
@using System.Data
@model Ch09.AdoNet.Models.HomeViewModel

<h1>RECORDS</h1>
<table class="table table-condensed">
    @foreach (DataRow row in Model.eggrafes.Rows)
    {
        <tr>
            <td>@row["FirstName"]</td>
            <td>@row["LastName"]</td>
            <td>@row["Country"]</td>
        </tr>
    }
</table>
```

Το Index προβάλλει τις παρακάτω εγγραφές:

ΕΓΓΡΑΦΕΣ

name0	lastname0	greece
name1	lastname1	greece
name2	lastname2	greece
name3	lastname3	greece
name4	lastname4	greece
name5	lastname5	greece
name6	lastname6	greece
name7	lastname7	greece
name8	lastname8	greece
name9	lastname9	greece

Εικόνα 70 – Πίνακας Εγγραφών Χρηστών

Efcore

Το homeservice:

HomeService.cs

```
  public class HomeService
    {
        private readonly RecordRepository _repo;

        public HomeService(RecordRepository repo)
        {
            _repo = repo;
        }

        public HomeViewModel GetHomeViewModel()
        {

            var model = new HomeViewModel {Records = _repo.GetRecords()};
            return model;
        }
    }
```

Η Recordservice κάνει τα Records:

RecordService.cs

```
public class RecordService
    {
        private readonly RecordRepository _repo;

        public RecordService(RecordRepository repo)
        {
            _repo = repo;
        }

        public RecordViewModel GetNewRecordViewModel()
        {
            var model = new RecordViewModel
            {
                Customer = {FirstName = "Nick"}
            };
            return model;
        }

        public void SaveRecord(string fn, string ln)
        {
            var customer = new Customer();
            customer.FirstName = fn;
            customer.LastName = ln;
            _repo.Save(customer);
        }
    }
```

Εδώ δημιουργείται η βάση της εφαρμογής.

YourDatabase.cs

```
public class YourDatabase : DbContext
    {
        public static string ConnectionString = "";

        public DbSet<Customer> Customers { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
        {
            optionsBuilder.UseSqlServer(ConnectionString);
        }
    }
```

Το RecordRepository class:

RecordRepository.cs

```
public class RecordRepository
    {
        public IList<Customer> GetRecords()
        {
            using (var db = new YourDatabase())
            {
                var list = (from c in db.Customers select c).ToList();
                return list;
            }
        }

        public void Save(Customer customer)
        {
            using (var db = new YourDatabase())
            {
                var existing = (from c in db.Customers
                    where c.Id == customer.Id
                    select c).SingleOrDefault();
                if (existing == null)
                {
                    db.Customers.Add(customer);
                }
                else
                {
                    existing.FirstName = customer.FirstName;
                    existing.LastName = customer.LastName;
                }
                db.SaveChanges();
            }
        }
    }
```

Το μοντέλο customer

Customer.cs

```
  public class Customer : EntityBase
    {
        [Key]
        public int Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }

        [NotMapped]
        public bool Done { get; set; }
    }
```

Η βοηθητική κλάση dbcreate που ελέγχει την δημιουργία βάσης:

dbcreate.cs

```
public class dbcreate
    {

        public void dbcreratefunc()
        {

            string myConnectionString = "Data
Source=(localdb)\\MSSQLLocalDB;Initial Catalog=master;Integrated
Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False"; //ConnectionStrings.progconn;

            using (var connection = new SqlConnection(myConnectionString))
            {
                connection.Open();
                var command = connection.CreateCommand();
                command.CommandText = "CREATE DATABASE ProgCore6";
                command.ExecuteNonQuery();
                connection.Close();
            }
        }

        public void dbcreatetables()
        {
            string myConnectionString = "Data
Source=(LocalDB)\\MSSQLLocalDB;Initial Catalog=progcore6;Integrated
Security=True";

            using (SqlConnection con = new SqlConnection(myConnectionString))
            {

                try
                {
                    //
                    // Άνοιξε την SqlConnection.
                    //
                    con.Open();
```

```
                    //
// Ο παρακάτω κώδικας χρησιμοποιεί μια SqlCommand βασιμένη στην SqlConnection.
                    //
                    using (SqlCommand command = new SqlCommand("CREATE TABLE
Customers(Id int, FirstName char(50),LastName char(50),Done bool);", con))
                        command.ExecuteNonQuery();
                    con.Close();
                }
                catch (Exception ex)
                {
                    Debug.WriteLine(ex.Message);
                }
            }

        }

        public void dbcreaterecs(int m)
        {
            string myConnectionString = "Data
Source=(LocalDB)\\MSSQLLocalDB;Initial Catalog=progcore6;Integrated
Security=True"; ;

            using (SqlConnection con = new SqlConnection(myConnectionString))
            {
                try
                {
                   //
              // Άνοιξε την SqlConnection.
                    //
                    con.Open();
                    //
// Ο παρακάτω κώδικας χρησιμοποιεί μια SqlCommand βασιμένη στην SqlConnection.
                    //
                    for (int j = 0; j < m; j++)
                    {
                        string fname = "name" + j.ToString();
                        string lname = "lastname" + j.ToString();
                        string country = "greece";
                        string phonec = "210-888" + j.ToString();
                        string addressc = "address" + j.ToString();
                        string query = "INSERT INTO Customers (FirstName,
LastName, Country, phone, address) VALUES ( '" + fname + "', '" + lname + "',
'" + country + "', '" + phonec + "', '" + addressc + "' )";

                        SqlCommand myCommand = new SqlCommand(query, con);

                        myCommand.ExecuteNonQuery();
                    }
                    con.Close();
                }
                catch (Exception ex)
                {
                    Debug.WriteLine(ex.Message);
                }
            }
        }
    }
```

Το μοντέλο Entity Base:

EntityBase.cs

```
  public class EntityBase
    {
        public EntityBase()
        {
            Enabled = true;
            Modified = DateTime.UtcNow;
        }

        public bool Enabled { get; set; }
        public DateTime? Modified { get; set; }
    }
```

Ακολουθούν οι κάτωθι ελεγκτές:

Ο ελεγκτής Home controller προβάλλει τις εγγραφές:

HomeController.cs

```
public class HomeController : Controller
    {
        private readonly HomeService _service;
        public HomeController(HomeService service)
        {
            _service = service;
        }

        public IActionResult Index()
        {
            var model = _service.GetHomeViewModel();
            return View(model);
        }
    }
```

Ο recordController διαχειρίζεται την προσθεση εγγραφής:

RecordController.cs

```
  public class RecordController : Controller
    {
        private readonly RecordService _service;
        public RecordController(RecordService service)
        {
            _service = service;
        }

        [HttpGet]
        public IActionResult New()
        {
            var model = _service.GetNewRecordViewModel();
            return View(model);
        }
```

```
        [HttpPost]
        public IActionResult New(string firstname, string lastname)
        {
            _service.SaveRecord(firstname, lastname);
            return RedirectToAction("index", "home");
        }
    }
```

Το μοντέλο HomeViewModel κάνει την λίστα των Customers:

HomeViewModel.cs

```
public class HomeViewModel
    {
        public IList<Customer> Records { get; set; }
    }
```

RecordViewModel.cs

```
public class RecordViewModel
    {
        public RecordViewModel()
        {
            Customer = new Customer();
        }

        public Customer Customer { get; set; }
    }
```

Προβολή Εγγραφών Index.cshtml

```
@model Ch09.EfCore.Models.HomeViewModel

<h1>ΕΓΓΡΑΦΕΣ<a href="@Url.Action("new", "record")" class="btn btn-danger">
ΠΡΟΣΘΕΣΕ</a></h1>
<table class="table table-condensed">
    @foreach (var customer in Model.Records)
    {
        <tr>
            <td>@customer.FirstName</td>
            <td>@customer.LastName</td>
        </tr>
    }
</table>
```

ΕΓΓΡΑΦΕΣ ΠΡΟΣΘΕΣΕ

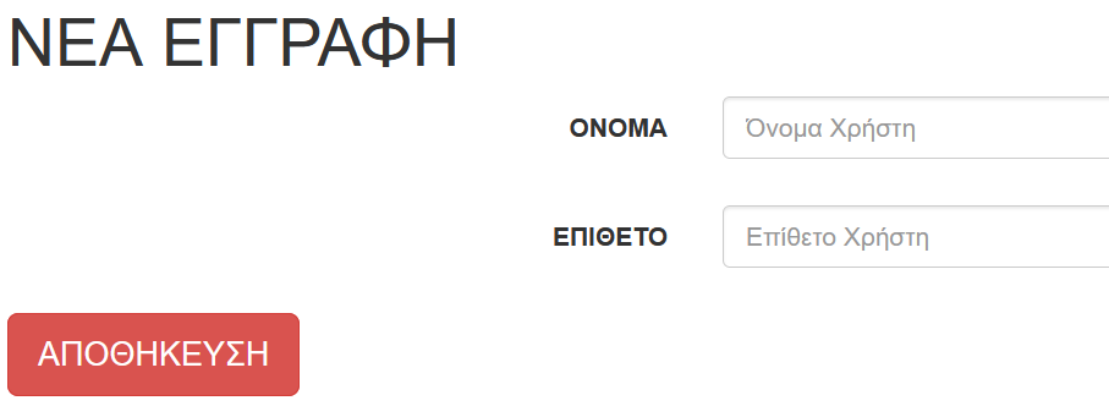
Nick	taxidis
tassos	werty
Cinamon	Roller
rick	astley
edy	edws
Lemon	Pie
ricky	martin
Apple	Pie
Peach	Pie
Cherry	Tree

Εικόνα 71 – Πίνακας Ονομάτων Χρηστών

Εισαγωγή καινούριας εγγραφής record.cshtml

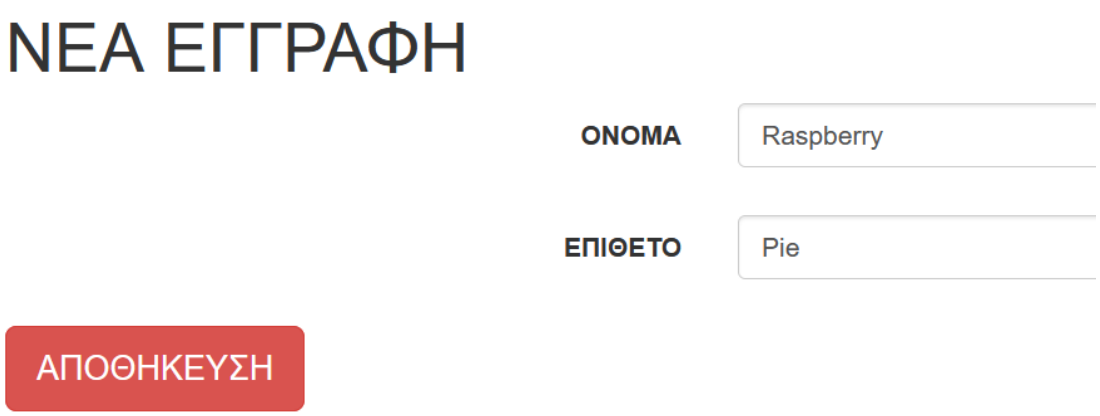
```
@model Ch09.EfCore.Models.RecordViewModel

<h1>ΝΕΑ ΕΓΓΡΑΦΗ</h1>

<form class="form-horizontal" method="post">
    <fieldset>
        <!-- FIRST -->
        <div class="form-group">
            <label class="col-md-4 control-label" for="firstname">ΟΝΟΜΑ</label>
            <div class="col-md-8">
                <input type="text" class="form-control"
                       id="firstname"
                       name="firstname"
                       value="@Model.Customer.FirstName"
                       placeholder="Όνομα Χρήστη">
                <span class="help-block"> </span>
            </div>
        </div>
        <!-- LAST -->
        <div class="form-group">
            <label class="col-md-4 control-label"
for="lastname">ΕΠΙΘΕΤΟ</label>
            <div class="col-md-8">
                <input type="text" class="form-control"
                       id="lastname"
                       name="lastname"
                       value="@Model.Customer.LastName"
                       placeholder="Επίθετο Χρήστη">
                <span class="help-block"> </span>
            </div>
        </div>

        <button class="btn btn-lg btn-danger">ΑΠΟΘΗΚΕΥΣΗ</button>
    </fieldset>
</form>
```

Ο χρήστης θα κάνει μια νέα εγγραφή.



Εικόνα 72 – Φόρμα Εισαγωγής νέου Χρήστη

Ο χρήστης εισάγει τα στοιχεία της νέας εγγραφής και πατάει ΑΠΟΘΗΚΕΥΣΗ.



Εικόνα 73 – Ο Χρήστης εισήγαγε τα στοιχεία του στην Φόρμα

Έτσι εμφανίζεται στην λίστα η νέα εγγραφή στη λίστα εγγραφών:

ΕΓΓΡΑΦΕΣ ΠΡΟΣΘΕΣΕ

Nick	taxidis
tassos	werty
Cinamon	Roller
rick	astley
edy	edws
Lemon	Pie
ricky	martin
Apple	Pie
Peach	Pie
Cherry	Tree
Raspberry	Pie

Εικόνα 74 – Πίνακας με τα ονόματα των Χρηστών

Βιβλιογραφία

Programming ASP.NET Core, First Edition by Dino Esposito του Dino Esposito

ASP.NET Core in Action του Andrew Lock

Κεφάλαιο 10 - Ζητήματα σχεδιασμού

Το API σημαίνει (Application Programming Interface) είναι μια διεπαφή που επιτρέπει σε πολλές εφαρμογές να επικοινωνούν μεταξύ τους. Χρησιμοποιούμε πολλά API καθημερινά, όπως για να λαμβάνουμε την τρέχουσα ενημέρωση για τον καιρό της περιοχής μας στο τηλέφωνό μας, τις τρέχουσες τιμές των μετοχών μιας εταιρείας στην οποία επενδύσαμε και ούτω καθεξής.

Ακόμα και το λειτουργικό σύστημα (OS) του υπολογιστή μας αποτελείται από ένα μεγάλο αριθμό API. Για παράδειγμα, κάνοντας διπλό κλικ σε ένα φάκελο, το λειτουργικό σύστημα καλεί ένα συγκεκριμένο API που εμφανίζει το περιεχόμενο του φακέλου.

Καλούμε ένα API έτσι ώστε να εκτελεί την εργασία για την οποία έχει σχεδιαστεί. Όταν το API ολοκληρώσει την εργασία, θα επιστρέψει την κατάσταση της ολοκληρωμένης εργασίας, σε περίπτωση σφάλματος επιστρέφεται ο κατάλληλος κωδικός σφάλματος και το μήνυμα.

Το ερώτημα που τίθεται είναι πώς να καλέσετε ένα API.

- Μπορούμε να καλέσουμε ένα API χρησιμοποιώντας το κατάλληλο λογισμικό που έχει σχεδιαστεί για να λειτουργεί με το API, για παράδειγμα ένα λειτουργικό σύστημα, ένα πρόγραμμα περιήγησης ή μια εφαρμογή στο iPhone σας. Εάν το API έχει μια διεύθυνση URL, τότε αυτή η διεύθυνση URL μπορεί να κληθεί με πρωτόκολλα HTTP.

Τα Web APIs λειτουργούν με μεθόδους που καλούνται από πρωτόκολλα HTTP. Αυτές οι μέθοδοι είναι τα GET, PUT, POST και DELETE από τα πιο συνηθισμένα.

ApiControllers

Το Web API μπορεί να έχει μεθόδους δράσης τύπου GET / POST / PUT / PATCH / DELETE / HEAD. Με βάση την εισερχόμενη αίτηση, το Web API αποφασίζει ποια μέθοδος δράσης θα εκτελεστεί.

Παράδειγμα:

Το αίτημα τύπου HTTP GET θα αντιμετωπιστεί με τη μέθοδο δράσης τύπου GET.

Το αίτημα τύπου HTTP POST θα αντιμετωπιστεί με τη μέθοδο δράσης τύπου POST.

Κατά αυτόν τον τρόπο κινούμαστε και για τις υπόλοιπες μεθόδους δράσης.

Από προεπιλογή, η ASP.NET CORE κάνει τα εξής:

1. Στέλνει τα δεδομένα στον πελάτη ως συμβολοσειρά εάν η μέθοδος ενέργειας επιστρέψει μια συμβολοσειρά. Ορίζει επίσης την κεφαλίδα Content-Type της απόκρισης ως απλό κείμενο.
2. Στέλνει τα δεδομένα στον πελάτη ως JSON εάν ο τύπος επιστροφής της μεθόδου δράσης είναι οποιοσδήποτε άλλος, αλλά όχι ως συμβολοσειρά όπως int, datetime, αντικείμενο, απλός τύπος, σύνθετος τύπος, κλπ. Ορίζει επίσης την κεφαλίδα Content-Type της απόκρισης ως εφαρμογή JSON.

Οι μέθοδοι δράσης του Web API Controller έχουν εφαρμοστεί σε ορισμένα χαρακτηριστικά HTTP. Επομένως, επικαλούνται μόνο τη συγκεκριμένη μέθοδο HTTP γνωστή ως VERBS.

Παραδείγματα VERBS είναι : GET, POST, PUT, PATCH, DELETE και HEAD.

Οι κάτωθι ελεγκτές χρησιμοποιούνται:

HomeController.cs

```
public class HomeController : Controller
   {
      public string Index()
      {
         return " Το API τρέχει...";
      }
   }
```

ReservationController.cs

```
[ApiController]
   [Route("api/[controller]")]
   public class ReservationController : ControllerBase
   {
      private IRepository repository;

      private IWebHostEnvironment webHostEnvironment;

      public ReservationController(IRepository repo, IWebHostEnvironment environment)
      {
         repository = repo;
         webHostEnvironment = environment;
      }

      Παράγει το "application/xml"
      [HttpGet]
      public IEnumerable<Reservation> Get() => repository.Reservations;

      [HttpGet("{id}")]
      public ActionResult<Reservation> Get(int id)
      {
         if (id == 0)
            return BadRequest("Η εγγραφή δεν υπάρχει.");
         return Ok(repository[id]);
      }

      [HttpPost]
      public Reservation Post([FromBody] Reservation res) =>
      repository.AddReservation(new Reservation
      {
         Onoma = res.Onoma,
         Arxikithesi = res.Arxikithesi,
         Telikithesi = res.Telikithesi
      });
```

```
    [HttpPut]
    public Reservation Put([FromForm] Reservation res) => repository.UpdateReservation(res);

    [HttpPatch("{id}")]
    public StatusCodeResult Patch(int id, [FromBody] JsonPatchDocument<Reservation> patch)
    {
        var res = (Reservation)((OkObjectResult)Get(id).Result).Value;
        if (res != null)
        {
            patch.ApplyTo(res);
            return Ok();
        }
        return NotFound();
    }

    [HttpDelete("{id}")]
    public void Delete(int id) => repository.DeleteReservation(id);//Διαγραφή κράτησης

    bool Authenticate()
    {
        var allowedKeys = new[] { "Secret@123", "Secret#12", "SecretABC" };
        StringValues key = Request.Headers["Key"];
        int count = (from t in allowedKeys where t == key select t).Count();
        return count == 0 ? false : true;
    }

    [HttpPost("UploadFile")]
    public async Task<string> UploadFile([FromForm] IFormFile file)//Ανέβασμα αρχείου
    {
        string path = Path.Combine(webHostEnvironment.WebRootPath, "Images/" + file.FileName);
        using (var stream = new FileStream(path, FileMode.Create))
        {
            await file.CopyToAsync(stream);
        }
        return "https://localhost:44324/Images/" + file.FileName;
    }

    [HttpPost("PostXml")]
    [Consumes("application/xml")]
    public Reservation PostXml([FromBody] System.Xml.Linq.XElement res)
    {
        return repository.AddReservation(new Reservation
        {
            Onoma = res.Element("Onoma").Value,//Εδώ μπαίνει το όνομα του χρήστη
            Arxikithesi = res.Element("Arxikithesi").Value,//Εδώ μπαίνει το αεροδρόμιο αναχώρησης του
χρήστη.
            Telikithesi = res.Element("Telikithesi").Value//Εδώ μπαίνει το αεροδρόμιο άφιξης του χρήστη.
        });
    }//Εμφάνιση Reservation

    [HttpGet("ShowReservation.{format}"), FormatFilter]
    public IEnumerable<Reservation> ShowReservation() => repository.Reservations;
}
```

Μοντέλα που χρησιμοποιούνται:

IRepository.cs

```
public interface IRepository
```

```
  {
    IEnumerable<Reservation> Reservations { get; }
    Reservation this[int id] { get; }
    Reservation AddReservation(Reservation reservation);
    Reservation UpdateReservation(Reservation reservation);
    void DeleteReservation(int id);
  }
```

Παρακάτω είναι η λίστα με τις αρχικές εγγραφές, που περιέχουν τους 4 πρώτους χρήστες.

Repository.cs

```
 public class Repository: IRepository

  {
    private Dictionary<int, Reservation> items;

    public Repository()
    {
      items = new Dictionary<int, Reservation>(); //Λίστα με αρχικούς χρήστες
      new List<Reservation> {
          new Reservation {Id=1, Onoma = "John", Arxikithesi = "Dublin",
Telikithesi="Singapore" },
                new Reservation {Id=2, Onoma = "George", Arxikithesi =
"Brussels", Telikithesi="Miami" },
                new Reservation {Id=3, Onoma = "Robert", Arxikithesi =
"Manchester", Telikithesi="Rome" },
                new Reservation {Id=4, Onoma = "James", Arxikithesi = "Quebec",
Telikithesi="Tokyo" }
        }.ForEach(r => AddReservation(r));
    }

    public Reservation this[int id] => items.ContainsKey(id) ? items[id] : null;

    public IEnumerable<Reservation> Reservations => items.Values;
```

Το IEnumerable είναι η βασική διεπαφή για όλες τις μη γενικές συλλογές που μπορούν να απαριθμηθούν. Για τη γενική έκδοση αυτής της διεπαφής, ανατρέξτε στο System.Collections.Generic.IEnumerable <T>.

```
    public Reservation AddReservation(Reservation reservation)// Κάνε 1 νέα κράτηση.
    {
      if (reservation.Id == 0)
      {
        int key = items.Count;
        while (items.ContainsKey(key)) { key++; };
        reservation.Id = key;
      }
      items[reservation.Id] = reservation;
      return reservation;
    }

    public void DeleteReservation(int id) => items.Remove(id);// Κάνε διαγραφή 1 κράτησης.

    public Reservation UpdateReservation(Reservation reservation) => AddReservation(reservation);
  }// Κάνε ενημέρωση 1 κράτησης.
```

Το μοντέλο της κράτησης:

Reservation.cs

```
public class Reservation
   {
      public int Id { get; set; }
      public string Onoma { get; set; }
      public string Arxikithesi { get; set; }
      public string Telikithesi { get; set; }
   }
```

Εδώ βλέπουμε το id, το όνομα , το αρχικό αεροδρόμιο και το τελικό αεροδρόμιο του χρήστη.

Εμφάνιση στην οθόνη ότι η εφαρμογή τρέχει, άρα θα μπορέσει να λειτουργήσει και η επόμενη ApiConsume.

Εικόνα 75 – Το API τρέχει...

ApiConsume

Αρχικά ένα Πακέτο Bootstrap , χρησιμοποιήθηκε για το project.

Θα χρειαστώ πακέτο Bootstrap για να δώσω κάποια σωστή διάταξη στο HTML του View. Για αυτό προσθέστε αυτό το πακέτο μέσα στον φάκελο wwwroot ➢ lib του APIConsunment.

Η HttpClient έχει κάποια πολύ σημαντική μέθοδο για να καλέσει μια Web API.

Περιγραφή μεθόδου:

GetAsync Στέλνει ένα αίτημα GET στο καθορισμένο Uri ως ασύγχρονη λειτουργία.

PostAsync Στέλνει ένα αίτημα POST στο καθορισμένο Uri ως ασύγχρονη λειτουργία.

PutAsync Στέλνει ένα αίτημα PUT στο καθορισμένο Uri ως ασύγχρονη λειτουργία.

SendAsync Κάνει Αποστολή αιτήματος HTTP ως ασύγχρονη λειτουργία.

PatchAsync Στέλνει ένα αίτημα PATCH με διακριτικό ακύρωσης ως ασύγχρονη λειτουργία.

DeleteAsync Κάνει Αποστολή αιτήματος ΔΙΑΓΡΑΦΗΣ στο καθορισμένο Uri ως ασύγχρονη λειτουργία.

Εδώ πρέπει να ειπωθεί ότι για να έχετε πρόσβαση σε όλες τις δυνατότητες της εφαρμογής APIConsume.sln, πρέπει να τρέξετε πρώτα την APIControllers.sln και μετά παράλληλα να τρέξετε την APIConsume.sln.

Οι κάτωθι ελεγκτές χρησιμοποιούνται:

Το Web API έχει μια μέθοδο HttpGet που επιστρέφει όλες τις εγγραφές κράτησης στο JSON.
Ο κώδικας αυτής της μεθόδου είναι:

```
[HttpGet]
δημόσιο IEnumerable <Reservation> Get () => repository.Reservations;
```

Για να διαβάσω όλα αυτά τα αρχεία κράτησης, πρέπει να κάνω ένα HTTP GET Type of Request σε αυτήν τη μέθοδο του Web API.

HomeController.cs

```
<public class HomeController : Controller
  {
    public async Task<IActionResult> Index()
    {//Η λίστα των κρατήσεων.
      List<Reservation> reservationList = new List<Reservation>();
      using (var httpClient = new HttpClient())
      {
        using (var response = await httpClient.GetAsync("https://localhost:44324/api/Reservation"))
        {
          string apiResponse = await response.Content.ReadAsStringAsync();
          reservationList = JsonConvert.DeserializeObject<List<Reservation>>(apiResponse);
        }
      }
      return View(reservationList);
  }
```

Εδώ είναι ο κώδικας για την παραλαβή της λίστας κρατήσεων (List<Reservation>), που χρησιμοποιείται στην Index.cshtml του Home, για την εμφάνιση όλων των κρατήσεων.

Χρησιμοποίησα την κλάση HttpClient του χώρου ονομάτων System.Net.Http για να κάνω ένα αίτημα API. Θα παρατηρήσετε ότι το Index Action είναι ένας ασύγχρονος τύπος που επιστρέφει μια εργασία. Αυτό συμβαίνει επειδή η κλάση HttpClient υποβάλλει μόνο ασύγχρονο αίτημα που μπορεί να συμβεί μόνο από μια ασύγχρονη μέθοδο δράσης.

Έκανα το αίτημα HTTP GET στο API στη γραμμή - var response = await httpClient.GetAsync ("https: // localhost: 44324 / api / Reservation"). Παρατηρήστε ότι η διεύθυνση URL της ενέργειας του API παρέχεται στη μέθοδο GetAsync ().

Η Απόκριση(Response) API Ιστού, δηλαδή τα δεδομένα που επιστρέφονται από το API ανακτώνται από τον κώδικα - await response.Content.ReadAsStringAsync () και αποθηκεύονται στη μεταβλητή που ονομάζεται apiResponse.

Είναι ήδη γνωστό ότι η απόκριση API είναι ένας τύπος JSON όλων των αντικειμένων κράτησης. Για αυτό το λόγο μπορώ εύκολα να αποστειρώσω(να κάνω Deserialization) το JSON σε ένα αντικείμενο τύπου λίστας με την χρήση του πακέτου `Newtonsoft.Json` και με τον κώδικα:
reservationList = JsonConvert.DeserializeObject<List<Reservation>>(apiResponse); .

Η αποστείρωση(Deserialization) είναι η διαδικασία αποκωδικοποίησης των δεδομένων που είναι σε μορφή JSON σε εγγενή τύπο δεδομένων.

Εάν στείλω ένα αναγνωριστικό μιας κράτησης στο Web API, τότε το API θα μου στείλει ξανά τα στοιχεία κράτησης αυτού του αναγνωριστικού. Το Web API, το οποίο έχω ήδη δημιουργήσει στον προηγούμενο οδηγό μου, έχει μια μέθοδο HttpGet με μια παράμετρο id. Αυτό που κάνει είναι να διαβάζει ένα δίσκο του οποίου το αναγνωριστικό παρέχεται σε αυτό. Ο κώδικας αυτής της μεθόδου εμφανίζεται παρακάτω.

```
public ViewResult GetReservation() => View();

        [HttpPost]
        public async Task<IActionResult> GetReservation(int id)
        {
            Reservation reservation = new Reservation();
            using (var httpClient = new HttpClient())
            {
                using (var response = await
httpClient.GetAsync("https://localhost:44324/api/Reservation/" + id))
                {

                    if (response.StatusCode == System.Net.HttpStatusCode.OK)
                    {
                        string apiResponse = await
response.Content.ReadAsStringAsync();
                        reservation =
JsonConvert.DeserializeObject<Reservation>(apiResponse);
                    }
                    else
                        ViewBag.StatusCode = response.StatusCode;
                }
            }
            return View(reservation);
        }
```

Εδώ είναι ο κώδικας για την παραλαβή της κράτησης (Reservation) σύμφωνα με το ID της. Αυτή η μέθοδος καλείται στην GetReservation.cshtml.

Τώρα θα καλέσω αυτήν τη μέθοδο του API από το έργο πελάτη(customer). Θυμάστε ότι στην προβολή ευρετηρίου έχω δημιουργήσει έναν σύνδεσμο για να ληφθεί η κράτηση, τώρα θα προσθέσω τη λειτουργικότητά του.

Η κλήση API γίνεται από την κλάση HttpClient() και η απόκριση, η οποία είναι το αντικείμενο Κράτησης στο JSON, αποστειρώνεται(Deserialization) στο αντικείμενο κλάσης Reservation. Το αντικείμενο κλάσης Reservation επιστρέφεται στη συνέχεια στην προεπιλεγμένη προβολή ως μοντέλο.

Εδώ είναι ο κώδικας για την προσθήκη 1 νέας κράτησης (Reservation). Αυτή η μέθοδος καλείται στην AddReservation.cshtml.

```
     public ViewResult AddReservation() => View();

        [HttpPost]
        public async Task<IActionResult> AddReservation(Reservation
reservation)
        {
            Reservation receivedReservation = new Reservation();
            using (var httpClient = new HttpClient())
            {
                StringContent content = new
StringContent(JsonConvert.SerializeObject(reservation), Encoding.UTF8,
"application/json");

                using (var response = await
httpClient.PostAsync("https://localhost:44324/api/Reservation", content))
                {
                    string apiResponse = await
response.Content.ReadAsStringAsync();
                    receivedReservation =
JsonConvert.DeserializeObject<Reservation>(apiResponse);
                }
            }
            return View(receivedReservation);
}
```

Για να προσθέσω μια νέα κράτηση πρέπει να κάνω ένα αίτημα HTTP POST σε αυτήν τη μέθοδο του API ιστού. Θα χρησιμοποιήσω τη μέθοδο PostAsync () της κλάσης HttpClient για να πραγματοποιήσω μια κλήση σε αυτήν τη μέθοδο.

Να σημειωθεί ότι η κάθε κράτηση αποθηκεύεται τοπικά στον υπολογιστή.

Η έκδοση HTTP GET της ενέργειας UpdateReservation απλώς υποβάλλει έναν τύπο GET αιτήματος στο Web API. Παρέχει στο API το αναγνωριστικό μιας κράτησης. Το API θα στείλει ξανά την εγγραφή κράτησης το αναγνωριστικό του οποίου του παρασχέθηκε.

Εδώ είναι ο κώδικας για την ενημέρωση 1 νέας κράτησης (Reservation). Αυτή η μέθοδος καλείται στην UpdateReservation.cshtml.

```
    public async Task<IActionResult> UpdateReservation(int id)
        {
            Reservation reservation = new Reservation();
            using (var httpClient = new HttpClient())
            {
```

```
                using (var response = await
httpClient.GetAsync("https://localhost:44324/api/Reservation/" + id))
                {
                    string apiResponse = await
response.Content.ReadAsStringAsync();
                    reservation =
JsonConvert.DeserializeObject<Reservation>(apiResponse);
                }
            }
            return View(reservation);
        }

[HttpPost]
        public async Task<IActionResult> UpdateReservation(Reservation
reservation)
        {
            Reservation receivedReservation = new Reservation();
            using (var httpClient = new HttpClient())
            {        //Εδώ θα δοθούν τα νέα στοιχεία της κράτησης.
                var content = new MultipartFormDataContent();
                content.Add(new StringContent(reservation.Id.ToString()),
"Id");
                content.Add(new StringContent(reservation.Onoma), "Onoma");
                content.Add(new StringContent(reservation.Arxikithesi),
"Arxikithesi");
                content.Add(new StringContent(reservation.Telikithesi),
"Telikithesi");

                using (var response = await
httpClient.PutAsync("https://localhost:44324/api/Reservation", content))
                {
                    string apiResponse = await
response.Content.ReadAsStringAsync();
                    ViewBag.Result = "Success";
                    receivedReservation =
JsonConvert.DeserializeObject<Reservation>(apiResponse);
                }
            }
            return View(receivedReservation);
}
```

Για να επικαλεστώ αυτήν τη μέθοδο PATCH του API Ιστού, θα πρέπει να χρησιμοποιήσω την κλάση HttpRequestMessage για να προετοιμάσω 3 ιδιότητες. Αυτές οι ιδιότητες είναι:

1. RequestUri - η διεύθυνση URL για το αίτημα PATCH.
2. Μέθοδος - για τον καθορισμό της μεθόδου HTTP ως PATCH.
3. Περιεχόμενο - για να καθορίσετε τον τύπο JSON, Κωδικοποίηση και πολυμέσων.

Εδώ είναι ο κώδικας για την ενημέρωση 1 νέας κράτησης (Reservation) με Patch. Αυτή η μέθοδος καλείται στην UpdateReservationPatch.cshtml.

```
public async Task<IActionResult> UpdateReservationPatch(int id)
```

```
        {
            Reservation reservation = new Reservation();
            using (var httpClient = new HttpClient())
            {
                using (var response = await
httpClient.GetAsync("https://localhost:44324/api/Reservation/" + id))
                {
                    string apiResponse = await
response.Content.ReadAsStringAsync();
                    reservation =
JsonConvert.DeserializeObject<Reservation>(apiResponse);
                }
            }
            return View(reservation);
        }

        [HttpPost]
        public async Task<IActionResult> UpdateReservationPatch(int id,
Reservation reservation)
        {
            using (var httpClient = new HttpClient())
            {
                var request = new HttpRequestMessage
                {
                    RequestUri = new
Uri("https://localhost:44324/api/Reservation/" + id),
                    Method = new HttpMethod("Patch"),
                    Content = new StringContent("[{ \"op\":\"replace\",
\"path\":\"Id\", \"value\":\"" + reservation.Id + "\"},{ \"op\":\"replace\",
\"path\":\"Onoma\", \"value\":\"" + reservation.Onoma + "\"},{
\"op\":\"replace\", \"path\":\"Arxikithesi\", \"value\":\"" +
reservation.Arxikithesi + "\"},{ \"op\":\"replace\", \"path\":\"Telikithesi\",
\"value\":\"" + reservation.Telikithesi + "\"}]", Encoding.UTF8,
"application/json")
                };//Εδώ γίνονται replace(αντικαθίστανται) τα στοιχεία της
κράτησης.
                var response = await httpClient.SendAsync(request);
            }
            return RedirectToAction("Index");
        }
```

Η αρχική κλάσση.

Startup.cs

```
namespace APIConsume
{
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllersWithViews();//Προσθέτει υπηρεσίες για ελεγκτές
στην καθορισμένη IServiceCollection. Αυτή η μέθοδος δεν θα καταχωρίσει τις υπηρεσίες
που χρησιμοποιούνται για σελίδες. AddControllersWithViews (IServiceCollection, Action
<MvcOptions>) Προσθέτει υπηρεσίες για ελεγκτές στην καθορισμένη IServiceCollection.
        }
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
```

```
                app.UseDeveloperExceptionPage();//Καταγράφει σύγχρονες και
ασύγχρονες περιπτώσεις εξαίρεσης από τον αγωγό και δημιουργεί απαντήσεις σφαλμάτων
HTML.
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
                // H default τιμή HSTS είναι 30 μέρες, αλλά μπορεί ο χρήστης να
το αλλάξει αν θελήσει.
                app.UseHsts();
            }
            app.UseHttpsRedirection();//Το HTTPS Redirection Middleware
(UseHttpsRedirection)  χρησιμοποιείται  για ανακατεύθυνση όλων των αιτημάτων HTTP σε
HTTPS.
            app.UseStaticFiles();//Η μέθοδος app.UseStaticFiles () προσθέτει το
μεσαίο λογισμικό(middleware) StaticFiles στον αγωγό αιτήματος. Το UseStaticFiles είναι μια
μέθοδος επέκτασης που περιλαμβάνεται στο μεσαίο λογισμικό StaticFiles, έτσι ώστε να
μπορούμε εύκολα να το διαμορφώσουμε.

            app.UseRouting();//Το UseRouting προσθέτει αντιστοίχιση διαδρομής στον
αγωγό middleware. Αυτό το ενδιάμεσο λογισμικό εξετάζει το σύνολο των τελικών σημείων
που ορίζονται στην εφαρμογή και επιλέγει την καλύτερη αντιστοίχιση με βάση το αίτημα.
            app.UseAuthorization();//Το Authentication Middleware
(UseAuthentication) επιχειρεί να κάνει τον έλεγχο ταυτότητας του χρήστη πριν του
επιτραπεί η πρόσβαση σε ασφαλείς πόρους.
            app.UseEndpoints(endpoints =>
            {//Η UseEndpoints προσθέτει εκτέλεση τελικού σημείου στον αγωγό
middleware. Εκτελεί τον αντιπρόσωπο που σχετίζεται με το επιλεγμένο τελικό σημείο.
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "{controller=Start}/{action=Index}/{id?}");
            });
        }
    }
}
```

Το μοντέλο κράτησης:

Reservation.cs

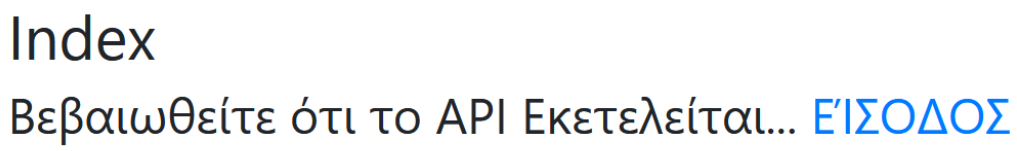
```
public class Reservation
  {
    public int Id { get; set; }
    public string Onoma { get; set; }
    public string Arxikithesi { get; set; }
    public string Telikithesi { get; set; }
  }
```

Το **index.cshtml** του Start.

```
@{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>
```

```
<h2>Βεβαιωθείτε ότι το API Εκτελείται... <a href="/Home"
target="_blank">ΕΊΣΟΔΟΣ</a></h2>
```

Η πρώτη σελίδα που συναντά ο χρήστης μόλις εισέλθει στο site.
Αρχικά προειδοποιείται αν τρέχει η ApiControllers, αν ναι τότε μπορεί να πατήσει το link ΕΊΣΟΔΟΣ για να εισέλθει στις κρατήσεις.



Εικόνα 76 - Index

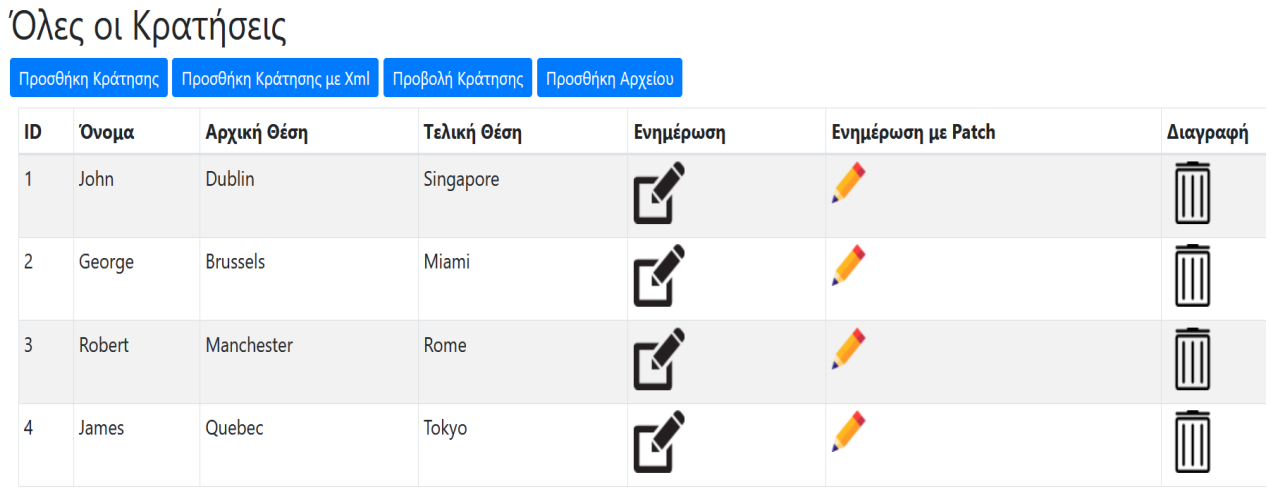
Το **index.cshtml** του Home.

```
@model IEnumerable<Reservation>
@{ Layout = "_Layout"; ViewBag.Title = "Όλες οι Κρατήσεις";}

<h2>Όλες οι Κρατήσεις</h2>
<a asp-action="AddReservation" class="btn btn-sm btn-primary">Προσθήκη
Κράτησης</a>
<a asp-action="AddReservationByXml" class="btn btn-sm btn-primary">Προσθήκη
Κράτησης με Xml</a>
<a asp-action="GetReservation" class="btn btn-sm btn-primary">Προβολή
Κράτησης</a>
<a asp-action="AddFile" class="btn btn-sm btn-primary">Προσθήκη Αρχείου</a>

<table class="table table-sm table-striped table-bordered m-2">
    <thead>
        <tr>
            <th>ID</th>
            <th>Όνομα</th>
            <th>Αρχική Θέση</th>
            <th>Τελική Θέση</th>
            <th>Ενημέρωση</th>
            <th>Ενημέρωση με Patch</th>
            <th>Διαγραφή</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var r in Model)
        {
        <tr>
            <td>@r.Id</td>
            <td>@r.Onoma</td>
            <td>@r.Arxikithesi</td>
            <td>@r.Telikithesi</td>
            <td>
                <a asp-action="UpdateReservation" asp-route-id="@r.Id">
                    <img src="/icon/edit.png" />
                </a>
            </td>
            <td>
                <a asp-action="UpdateReservationPatch" asp-route-id="@r.Id">
```

```
                    <img src="/icon/pencil.png" />
                </a>
            </td>
            <td>
                <form asp-action="DeleteReservation" method="post">
                  <input type="hidden" value="@r.Id" name="ReservationId" />
                    <input type="image" src="/icon/close.png" />
                </form>
            </td>
        </tr>
        }
    </tbody>
</table>
```

Εδώ εμφανίζονται οι αρχικές 4 εγγραφές.

Ο χρήστης έχει την δυνατότητα να επιλέξει να κάνει 1 κράτηση, να κάνει 1 κράτηση μέσω XML, να κάνει προβολή 1 κράτησης, να βάλει 1 αρχείο, να κάνει ενημέρωση 1 κράτησης, να κάνει ενημέρωση 1 κράτησης με Patch ή να διαγράψει 1 κράτηση.



Εικόνα 77 – Οι Αρχικές Κρατήσεις

Αρχικά θα πατήσει το κουμπί Προσθήκη Κράτησης με XML.

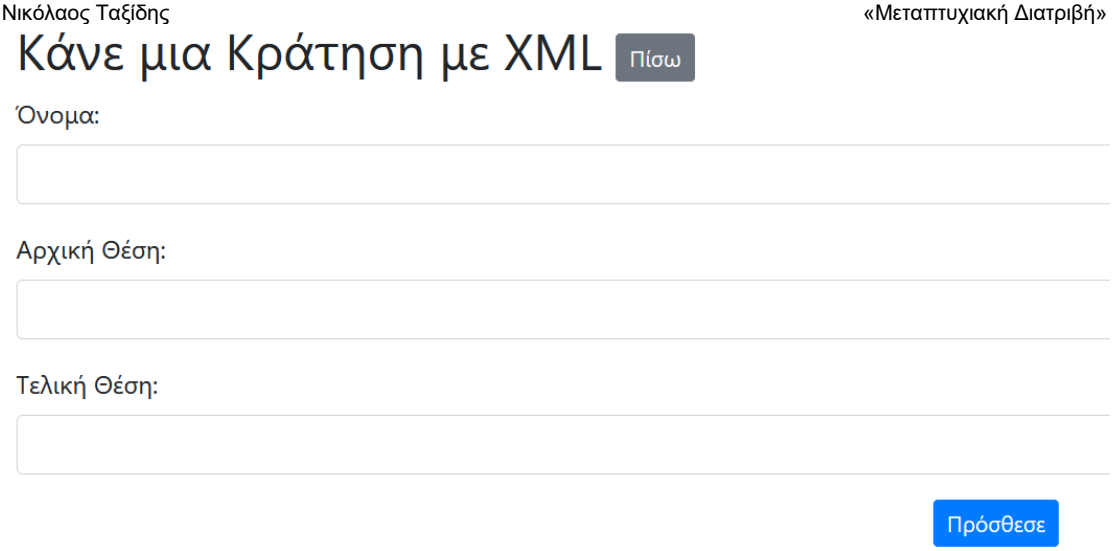
Εδώ είναι ο κώδικας για την προσθήκη 1 νέας κράτησης (Reservation). Αυτή η μέθοδος καλείται στην AddReservationByXml.cshtml.

Προσθήκη κράτησης με XML: AddReservationbyXML.cshtml

```
@model Reservation
@{ Layout = "_Layout"; ViewBag.Title = "Προσθήκη Κράτησης με XML";}

<h2>Κάνε μια Κράτηση με XML <a asp-action="Index" class="btn btn-sm btn-
secondary">Πίσω</a></h2>
<form asp-action="AddReservationByXml" method="post">
    <div class="form-group">
```

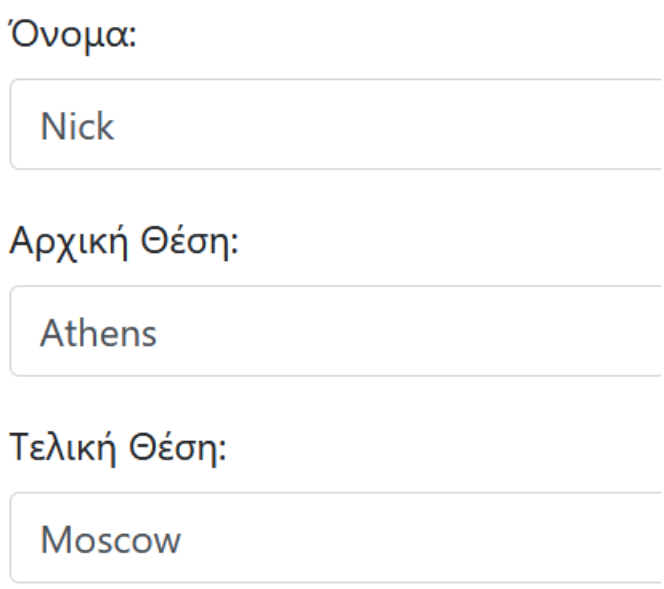
```
        <label for="Onoma">Όνομα:</label>
        <input class="form-control" name="Onoma" />
    </div>
    <div class="form-group">
        <label for="Arxikithesi">Αρχική Θέση:</label>
        <input class="form-control" name="Arxikithesi" />
    </div>
    <div class="form-group">
        <label for="Telikithesi">Τελική Θέση:</label>
        <input class="form-control" name="Telikithesi" />
    </div>
    <div class="text-center panel-body">
        <button type="submit" class="btn btn-sm btn-primary">Πρόσθεσε</button>
    </div>
</form>

<h3 class="alert">@ViewBag.Result</h3>

@if (Model != null)
{
    <h2>Κράτηση</h2>
    <table class="table table-sm table-striped table-bordered m-2">
        <thead>
            <tr>
                <th>ID</th>
                <th>Όνομα</th>
                <th>Αρχική Θέση</th>
                <th>Τελική Θέση</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>@Model.Id</td>
                <td>@Model.Onoma</td>
                <td>@Model.Arxikithesi</td>
                <td>@Model.Telikithesi</td>
            </tr>
        </tbody>
    </table>
}
```

Ο χρήστης πατάει Προσθήκη Κράτησης με XML.



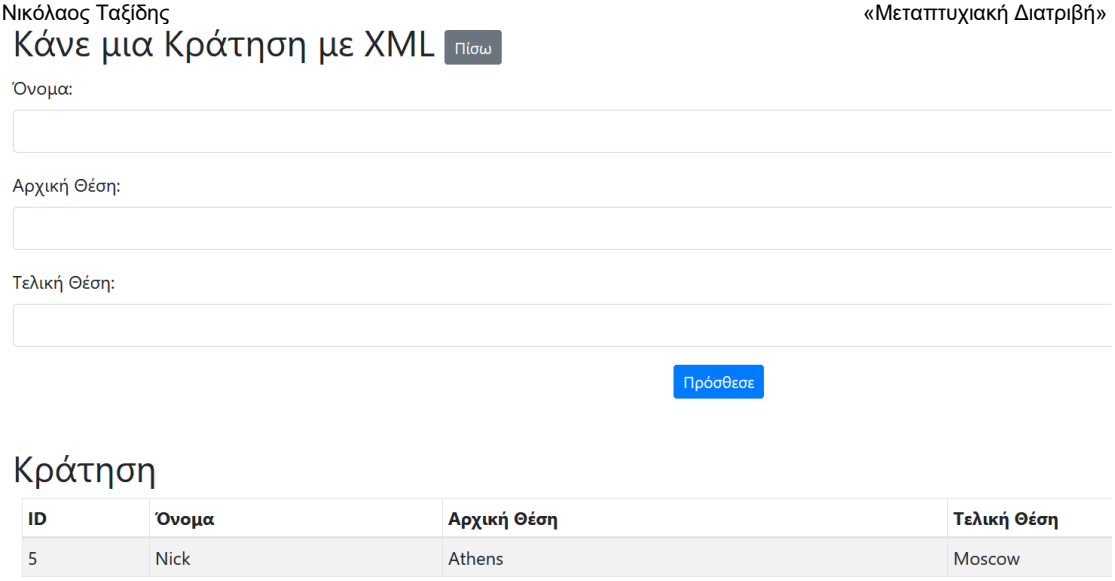
Εικόνα 78 – Κάνε Κράτηση με XML

Βάζει τα στοιχεία της Κράτησής του και πατάει Πρόσθεσε.



Εικόνα 79 – Εισαγωγή Στοιχείων Κράτησης με XML

Έτσι εμφανίζεται η νέα Κράτηση και ο χρήστης πατάει Πίσω.



Εικόνα 80 – Προβολή νέας Κράτησης από XML

Εμφάνιση νέου πίνακα πτήσεων.

Εικόνα 81 – Νέος Πίνακας Κρατήσεων με 5 εγγραφές

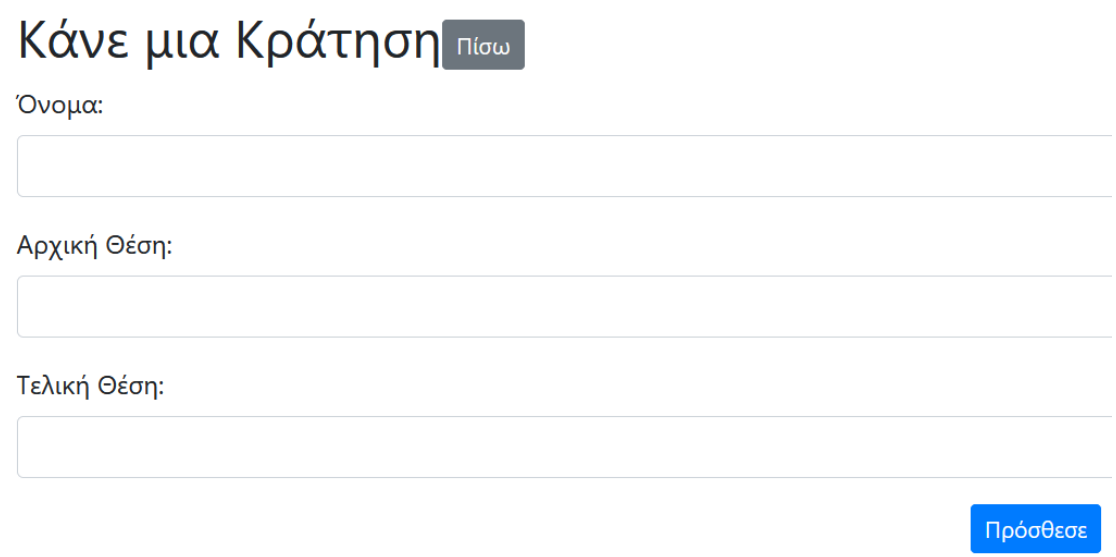
Μπορεί ο χρήστης να κάνει κράτηση και με έναν νέο τρόπο, πατώντας το κουμπί Προσθήκη κράτησης.

Προσθήκη κράτησης: `AddReservation.cshtml`

```
@model Reservation
@{ Layout = "_Layout"; ViewBag.Title = "Προσθήκη Κράτησης";}

<h2>Κάνε μια Κράτηση<a asp-action="Index" class="btn btn-sm btn-
secondary">Πίσω</a></h2>
<form asp-action="AddReservation" method="post">
    <div class="form-group">
        <label for="Onoma">Όνομα:</label>
        <input class="form-control" name="Onoma" />
    </div>
    <div class="form-group">
        <label for="Arxikithesi">Αρχική Θέση:</label>
        <input class="form-control" name="Arxikithesi" />
    </div>
    <div class="form-group">
```

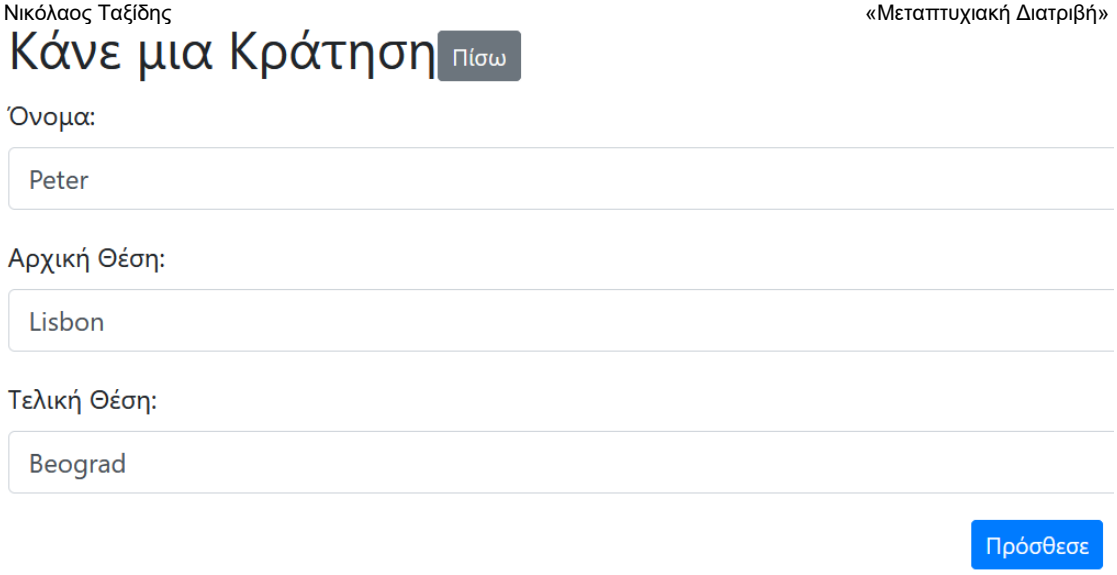
```
        <label for="Telikithesi">Τελική Θέση:</label>
        <input class="form-control" name="Telikithesi" />
    </div>
    <div class="text-center panel-body">
        <button type="submit" class="btn btn-sm btn-primary">Πρόσθεσε</button>
    </div>
</form>

<h3 class="alert">@ViewBag.Result</h3>

@if (Model != null)
{
    <h2>Krathsh</h2>
    <table class="table table-sm table-striped table-bordered m-2">
        <thead>
            <tr>
                <th>ID</th>
                <th>Όνομα</th>
                <th>Αρχική Θέση</th>
                <th>Τελική Θέση</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>@Model.Id</td>
                <td>@Model.Onoma</td>
                <td>@Model.Arxikithesi</td>
                <td>@Model.Telikithesi</td>
            </tr>
        </tbody>
    </table>
}
```

Ο χρήστης πατάει Προσθήκη Κράτησης.



Εικόνα 82 – Κάνε μια Κράτηση

Βάζει τα στοιχεία της Κράτησής του και πατάει Πρόσθεσε.

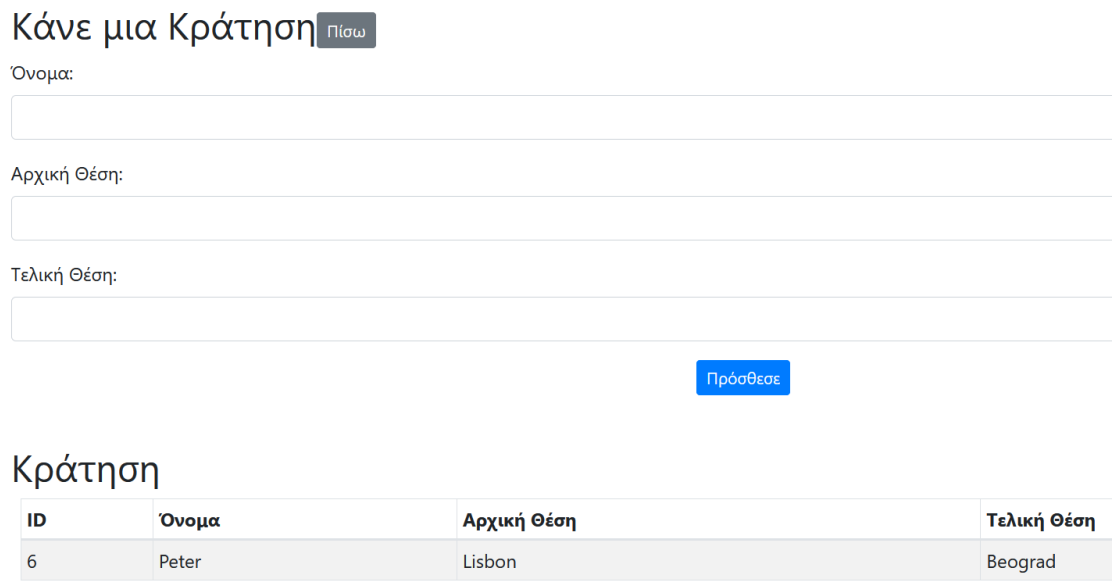


Εικόνα 83 – Εισαγωγή Στοιχείων νέας Κράτησης

Βάζει ο χρήστης τα στοιχεία του, δηλαδή το όνομά του, το αεροδρόμιο που θα πάρει το αεροπλάνο για να φύγει και το αεροδρόμιο που θέλει να πάει. Στη συγκεκριμένη περίπτωση το όνομα είναι Peter, το αεροδρόμιό του Lisbon και ο προορισμός του το Beograd.

Θα πατήσει το κουμπί Πρόσθεσε.

Εμφάνιση νέας κράτησης.



Εικόνα 84 – Προβολή νέας Κράτησης

Ο χρήστης πατάει το πλήκτρο Πίσω και επιστρέφει στην κεντρική σελίδα, όπου εμφανίζονται όλες οι κρατήσεις και φυσικά η νέα κράτηση του χρήστη.

Όλες οι Κρατήσεις

Προσθήκη Κράτησης | Προσθήκη Κράτησης με Xml | Προβολή Κράτησης | Προσθήκη Αρχείου

ID	Όνομα	Αρχική Θέση	Τελική Θέση	Ενημέρωση	Ενημέρωση με Patch	Διαγραφή
1	John	Dublin	Singapore			
2	George	Brussels	Miami			
3	Robert	Manchester	Rome			
4	James	Quebec	Tokyo			
5	Nick	Athens	Moscow			
6	Peter	Lisbon	Beograd			

Εικόνα 85 – Νέος Πίνακας Κρατήσεων με 6 εγγραφές

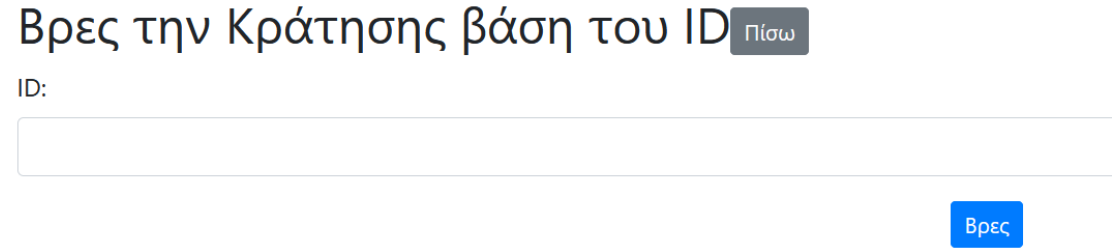
Εμφανίζεται ξανά η γνωστή πλατφόρμα της κράτησης. Ο χρήστης μπορεί να κάνει μια νέα κράτηση και να κινηθεί όπως πριν.

Μπορεί ο χρήστης να κάνει προβολή κράτησης πατώντας το κουμπί Προβολή Κράτησης.

Προβολή κράτησης: `GetReservation.cshtml`

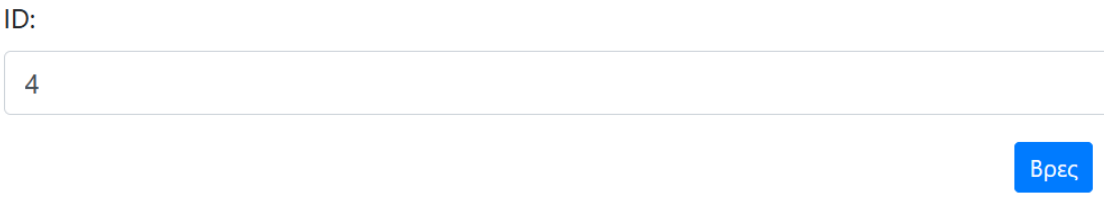
```
@model Reservation
@{ Layout = "_Layout"; ViewBag.Title = "Βρες Κράτηση μέσω Id";}
<h2>Βρες την Κράτησης βάση του ID<a asp-action="Index" class="btn btn-sm btn-
secondary">Πίσω</a></h2>
<h3>@ViewBag.StatusCode</h3>
<form method="post">
    <div class="form-group">
        <label for="id">ID:</label>
        <input class="form-control" name="id" />
    </div>
    <div class="text-center panel-body">
        <button type="submit" class="btn btn-sm btn-primary">Βρες</button>
    </div>
</form>
@if (Model != null)
{
    <h2>Κράτηση</h2>
    <table class="table table-sm table-striped table-bordered m-2">
        <thead>
            <tr>
                <th>ID</th>
                <th>Όνομα</th>
                <th>Αρχική Θέση</th>
                <th>Τελική Θέση</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>@Model.Id</td>
                <td>@Model.Onoma</td>
                <td>@Model.Arxikithesi</td>
                <td>@Model.Telikithesi</td>
```

```
            </tr>
        </tbody>
    </table>
}
```



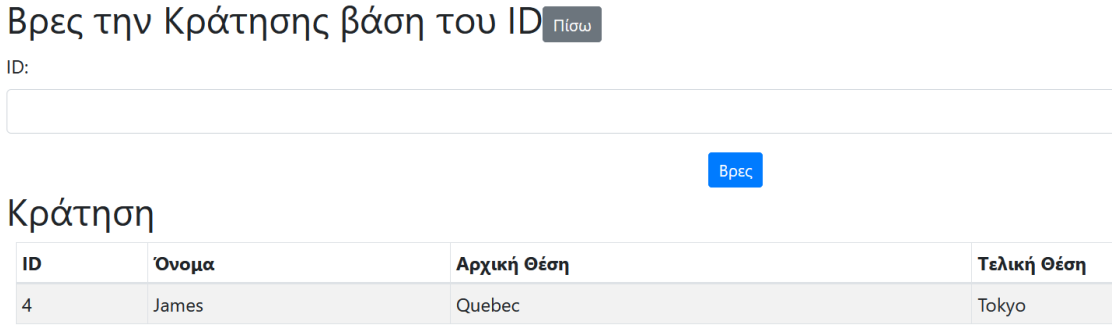
Εικόνα 86 – Εύρεση Κράτησης

Ο χρήστης εισάγει την τιμή ID 4 και πατάει το πλήκτρο Βρες.



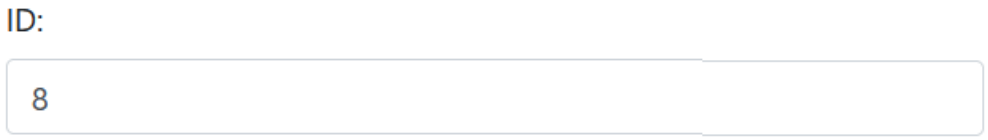
Εικόνα 87 – Εύρεση Κράτησης με ID=4

Το σύστημα εμφανίζει την κράτηση με ID 4.

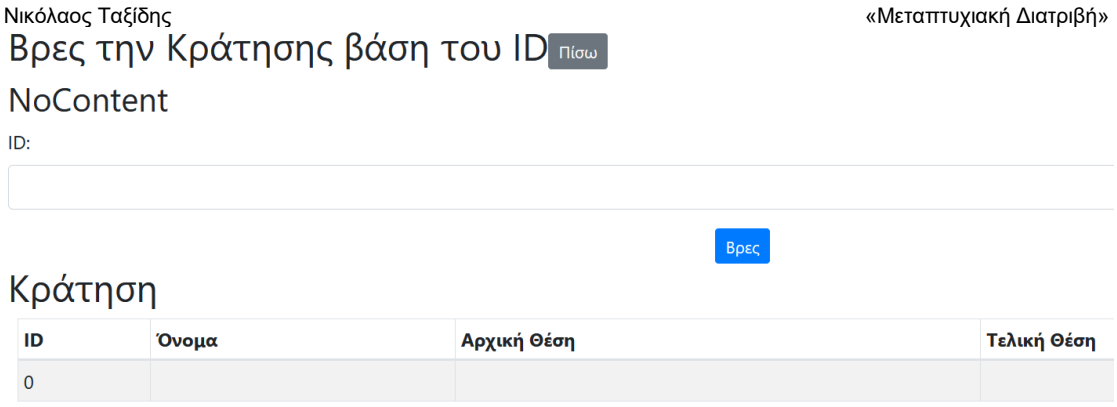


Εικόνα 88 – Εμφάνιση Κράτησης με ID=4

Αν ο χρήστης βάλει ένα ID που δεν υπάρχει στη λίστα με τις κρατήσεις, όπως το ID = 8 , που δεν υπάρχει τότε θα εμφανίσει μήνυμα λάθους NoContent και μια κενή εγγραφή.



Εικόνα 89 - Εύρεση Κράτησης με ID=8



Εικόνα 90 – Εμφάνιση Λάθους

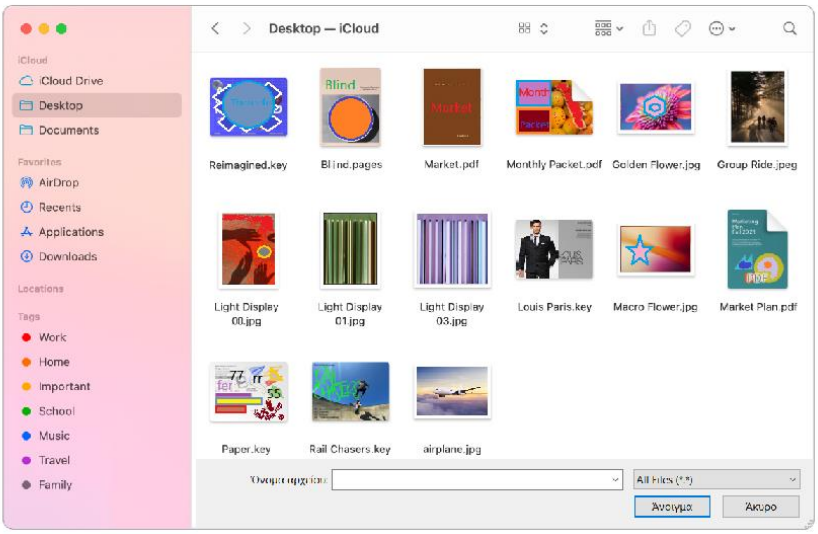
Ο χρήστης πατάει το πλήκτρο Πίσω και επιστρέφει στην κεντρική σελίδα, όπου πατάει το πλήκτρο Προσθήκη Αρχείου.

AddFile.cshtml

```
@model string
@{ Layout = "_Layout"; ViewBag.Title = "Προσθήκη Αρχείου";}

<h2>Πρόσθεσε Αρχείο</h2>
<a asp-action="Index" class="btn btn-sm btn-secondary">Πίσω</a>
<form method="post" enctype="multipart/form-data">
    <input type="file" name="file" />
    <div class="text-center panel-body">
        <button type="submit" class="btn btn-sm btn-primary">Πρόσθεσε</button>
    </div>
</form>

@if (Model != null)
{
    <h2>Ανεβασμένο Αρχείο</h2>
    <img src="@Model" />
}
```

Ο χρήστης πατάει το πλήκτρο Browse… και μετά θα επιλέξει πιο αρχείο θα προσθέσει στην εφαρμογή.
Ο χρήστης θα επιλέξει το αρχείο airplane.jgp και θα πατήσει το πλήκτρο Άνοιγμα.

Εικόνα 91 – Προσθήκη Αρχείου

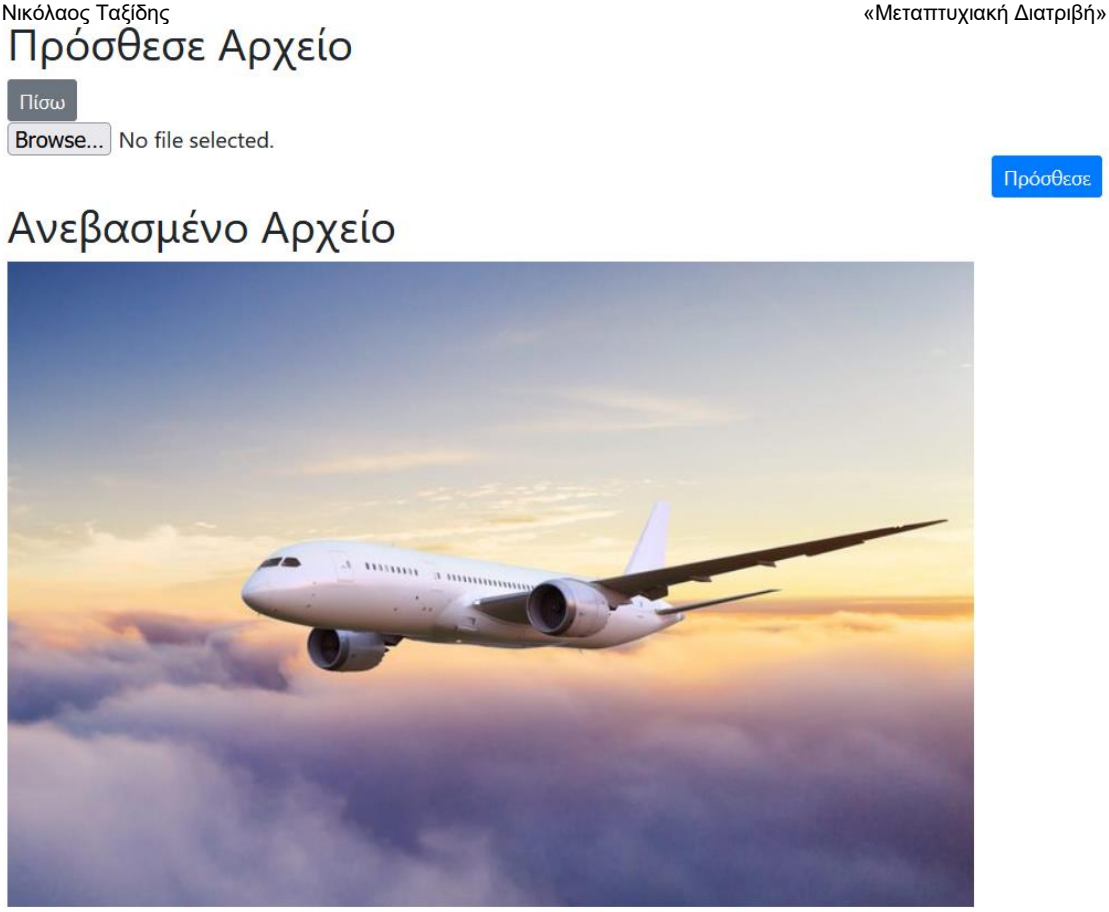


Εικόνα 92 – Εύρεση Αρχείου

Εδώ έχει επιλεγεί η εικόνα airplane.jpg και ο χρήστης πατάει Πρόσθεσε.

Εικόνα 93 – Προσθήκη Αρχείου airplane.jpg

Μετά θα εμφανιστεί η εικόνα στον χρήστη.



Εικόνα 94 – Εμφάνιση Ανεβασμένου Αρχείου

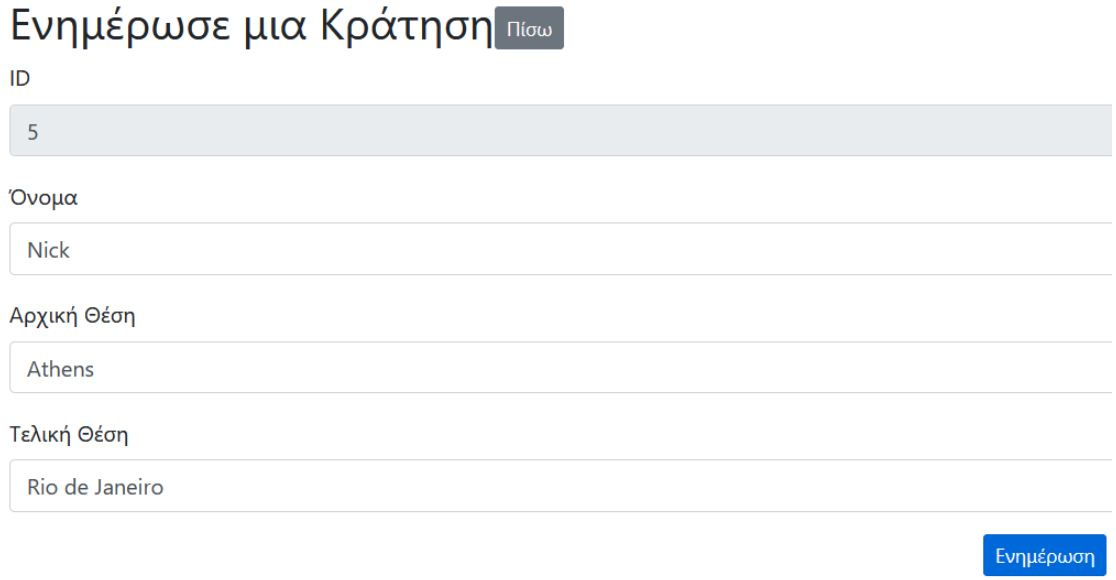
Ο χρήστης πατάει το πλήκτρο Πίσω και επιστρέφει στην κεντρική σελίδα, όπου θα επιλέξει το κουμπί της ενημέρωσης(σκούρο μολύβι) όποιας κράτησης επιθυμεί.

Ενημέρωση κράτησης: UpdateReservation. cshtml

```
@model Reservation
@{ Layout = "_Layout"; ViewBag.Title = "Ανανέωσε μια Κράτηση";}

<h2>Ενημέρωσε μια Κράτηση<a asp-action="Index" class="btn btn-sm btn-
secondary">Πίσω</a></h2>
<form method="post">
    <div class="form-group">
        <label asp-for="Id">ID</label>
        <input class="form-control" asp-for="Id" readonly />
    </div>
    <div class="form-group">
        <label asp-for="Onoma">Όνομα</label>
        <input class="form-control" asp-for="Onoma" />
    </div>
    <div class="form-group">
        <label asp-for="Arxikithesi">Αρχική Θέση</label>
        <input class="form-control" asp-for="Arxikithesi" />
    </div>
    <div class="form-group">
```

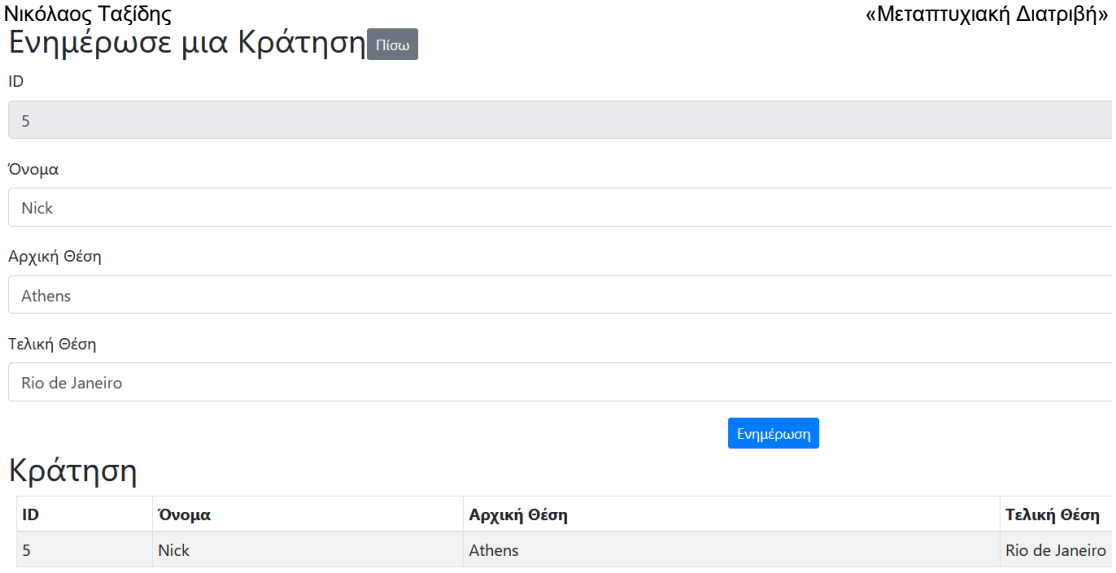
```
        <label asp-for="Telikithesi">Τελική Θέση</label>
        <input class="form-control" asp-for="Telikithesi" />
    </div>
    <div class="text-center panel-body">
        <button type="submit" class="btn btn-sm btn-primary">Ενημέρωση</button>
    </div>
</form>

@if (ViewBag.Result == "Success")
{
    <h2>Κράτηση</h2>
    <table class="table table-sm table-striped table-bordered m-2">
        <thead>
            <tr>
                <th>ID</th>
                <th>Όνομα</th>
                <th>Αρχική Θέση</th>
                <th>Τελική Θέση</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>@Model.Id</td>
                <td>@Model.Onoma</td>
                <td>@Model.Arxikithesi</td>
                <td>@Model.Telikithesi</td>
            </tr>
        </tbody>
    </table>
}
```

Ο χρήστης αλλάζει την κράτηση με ID 5 και θα βάλει ένα νέο αεροδρόμιο ως προορισμό και πατάει το πλήκτρο Ενημέρωση.



Εικόνα 95 – Ενημέρωσε μια Κράτηση

Εμφανίζεται στον χρήστη η κράτηση προσαρμοσμένη έτσι όπως τη ρύθμισε ο χρήστης.



Εικόνα 96 – Ενημέρωση Κράτησης με ID=5

Ο χρήστης πατάει το πλήκτρο Πίσω και επιστρέφει στην κεντρική σελίδα, όπου η κράτηση με ID=5 έχει αλλάξει πλέον.

Εικόνα 97 – Πίνακας Κρατήσεων με ενημερωμένη την Κράτηση με ID=5

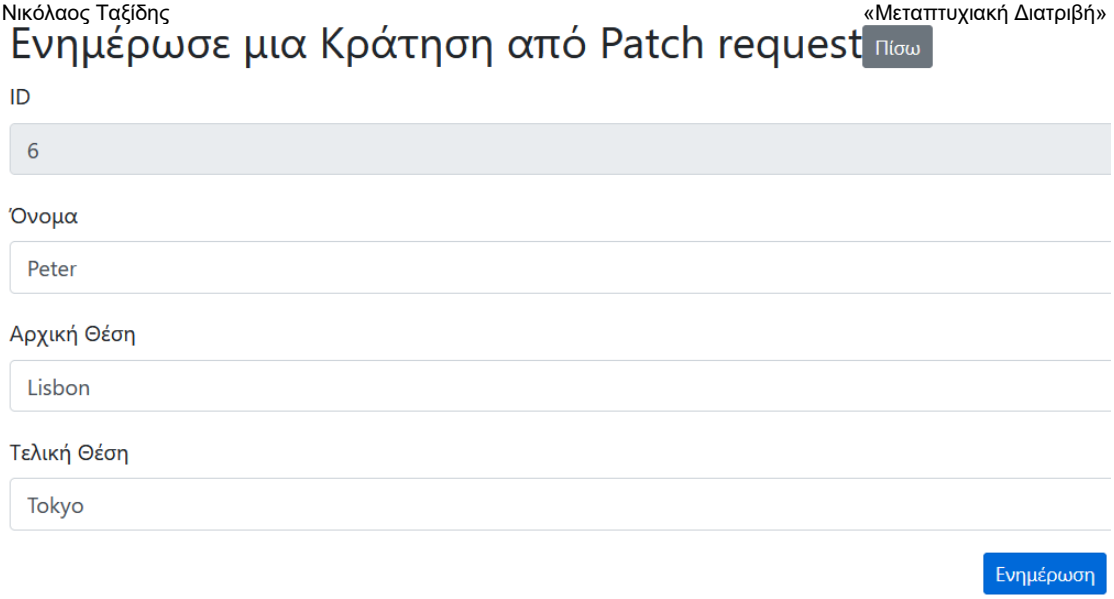
Ο χρήστης πατάει το κουμπί της ενημέρωσης με Patch(χρωματιστό μολύβι) όποιας κράτησης επιθυμεί.

Ενημέρωση κράτησης με Patch: UpdateReservationPatch. cshtml

```
@model Reservation
@{ Layout = "_Layout"; ViewBag.Title = "Ανανέωσε μια Κράτηση με PATCH";}

<h2>Ενημέρωσε μια Κράτηση από Patch request<a asp-action="Index" class="btn
btn-sm btn-secondary">Πίσω</a></h2>
<form method="post">
    <div class="form-group">
        <label asp-for="Id">ID</label>
        <input class="form-control" asp-for="Id" readonly />
    </div>
    <div class="form-group">
```

```
        <label asp-for="Onoma">Όνομα</label>
        <input class="form-control" asp-for="Onoma" />
    </div>
    <div class="form-group">
        <label asp-for="Arxikithesi">Αρχική Θέση</label>
        <input class="form-control" asp-for="Arxikithesi" />
    </div>
    <div class="form-group">
        <label asp-for="Telikithesi">Τελική Θέση</label>
        <input class="form-control" asp-for="Telikithesi" />
    </div>
    <div class="text-center panel-body">
        <button type="submit" class="btn btn-sm btn-primary">Ενημέρωση</button>
    </div>
</form>

@if (ViewBag.Result == "Success")
{
    <h2>Κράτηση</h2>
    <table class="table table-sm table-striped table-bordered m-2">
        <thead>
            <tr>
                <th>ID</th>
                <th>Όνομα</th>
                <th>Αρχική Θέση</th>
                <th>Τελική Θέση</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>@Model.Id</td>
                <td>@Model.Onoma</td>
                <td>@Model.Arxikithesi</td>
                <td>@Model.Telikithesi</td>
            </tr>
        </tbody>
    </table>
}
```

Ο χρήστης αλλάζει την κράτηση με ID 6 και θα βάλει ένα νέο αεροδρόμιο ως αρχικό και πατάει το πλήκτρο Ενημέρωση.



Εικόνα 98 – Ενημέρωσε μια Κράτηση από Patch request

Αυτή τη φορά επιστρέφει κατευθείαν στην κεντρική σελίδα, όπου η κράτηση με ID=6 έχει αλλάξει πλέον.

Εμφανίζεται η κράτηση προσαρμοσμένη έτσι όπως τη ρύθμισε ο χρήστης.

ID	Όνομα	Αρχική Θέση	Τελική Θέση	Ενημέρωση	Ενημέρωση με Patch	Διαγραφή
1	John	Dublin	Singapore			
2	George	Brussels	Miami			
3	Robert	Manchester	Rome			
4	James	Quebec	Tokyo			
5	Nick	Athens	Rio de Janeiro			
6	Peter	Lisbon	Tokyo			

Εικόνα 99 - Πίνακας Κρατήσεων με ενημερωμένη την Κράτηση με ID=6

Ο χρήστης πατάει το κουμπί της διαγραφής(κάδος) όποιας κράτησης επιθυμεί.

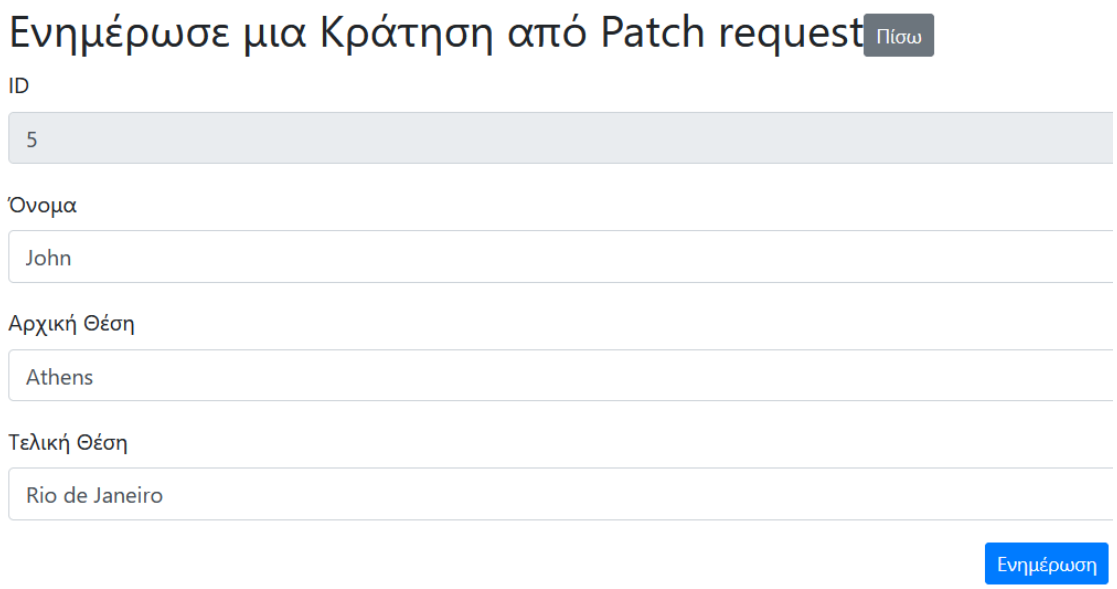
Ο χρήστης επιλέγει την κράτηση με ID=6 και έτσι αυτή η κράτηση εξαφανίζεται.

ID	Όνομα	Αρχική Θέση	Τελική Θέση	Ενημέρωση	Ενημέρωση με Patch	Διαγραφή
1	John	Dublin	Singapore			
2	George	Brussels	Miami			
3	Robert	Manchester	Rome			
4	James	Quebec	Tokyo			
5	Nick	Athens	Rio de Janeiro			

Εικόνα 100 - Πίνακας Κρατήσεων με διαγραμμένη την Κράτηση με ID=6

Έτσι λοιπόν εμφανίζονται οι 5 κρατήσεις πλέον εκτός από την κράτηση με ID=6.

Ο χρήστης πατάει το κουμπί της ενημέρωσης με Patch(χρωματιστό μολύβι) της κράτησης με ID ίσο με 4 και αλλάζει το όνομα της κράτησης.



Εικόνα 101 – Αλλαγή Ονόματος της Κράτησης με ID=5

Ο χρήστης αλλάζει την κράτηση με ID 5 και θα βάλει ένα νέο όνομα και πατάει το πλήκτρο Ενημέρωση.

ID	Όνομα	Αρχική Θέση	Τελική Θέση	Ενημέρωση	Ενημέρωση με Patch	Διαγραφή
1	John	Dublin	Singapore			
2	George	Brussels	Miami			
3	Robert	Manchester	Rome			
4	James	Quebec	Tokyo			
5	John	Athens	Rio de Janeiro			

Εικόνα 102 - Εμαφάνιση νέου Πίνακα Κρατήσεων

Έτσι λοιπόν εμφανίζονται οι 5 κρατήσεις προσαρμοσμένες έτσι όπως ήθελε ο χρήστης.

Κάποιες από τις λειτουργίες της εφαρμογής μπορεί να πραγματωθούν με 2 τρόπους για τη διευκόλυνση του χρήστη.

Βιβλιογραφία

Programming ASP.NET Core, First Edition by Dino Esposito του Dino Esposito

ASP.NET Core in Action του Andrew Lock

https://www.yogihosting.com/category/aspnet-core/

ΣΥΜΠΕΡΑΣΜΑΤΑ

Στην σημερινή εποχή που η ανάπτυξη του Προγραμματισμού είναι ραγδαία, η ανάγκη για συνεχή ενημέρωση, γνώση και κατάρτιση αυξάνεται εκθετικά.

Η νέα έκδοση της ASP.NET, η ASP.NET Core είναι ένα πολύ δημοφιλές πλαίσιο ανάπτυξης ιστοσελίδων για τη δημιουργία εφαρμογών Ιστού στην πλατφόρμα .NET.

Από το 2016 έχει μονοπωλήσει την ανάπτυξη εφαρμογών στα Windows, καθώς παρέχει ένα πλήρες σύστημα εργαλείων όπου ο χρήστης μπορεί να δημιουργήσει την web εφαρμογή που έχει ονειρευτεί.

Επιπλέον, το περιβάλλον της πλατφόρμας ASP.NET Core είναι μοντέρνο και υποστηρίζεται από όλα τα ηλεκτρονικά μέσα χωρίς προβλήματα και η πλοήγηση σε αυτό είναι εύκολη και ευχάριστη.

Έχει παρατηρηθεί ότι η ASP.NET Core παρέχει την καλύτερη λειτουργία της C# και ότι στο μέλλον μπορεί να μονοπωλήσει την πρώτη θέση στα προγράμματα που χρησιμοποιούν την C#.