

Vežbe 06

MVC (eng. Model View Controller)

MVC je jedan od dizajn paterna po kojem razvijamo veb aplikacije. MVC je skraćenica od glavnim komponenti koje ulaze u strukturu veb aplikacije.

- Model
 - opisuje biznis logiku
 - podaci su nezavisni od prezentacionog sloja
- View
 - predstavlja prezentacioni sloj
 - .cshtml (od MVC 5) i html stranice
- Controller
 - veza između korisnika i podatka
 - upravlja tokom izvršenja aplikacije

Struktura projekta


- App_Data - direktorijum koji sadrži podatke koje koristi veb aplikacija
- App_Start - sadrži konfiguracione klase, definiše ponašanje aplikacije na globalnom nivou
 - RouteController - definisano rutiranje za našu aplikaciju. (definisan šablon kako da pozivamo akcije unutar kontrolera tj. način na koji klijent šalje zahteve kontroleru)
- Content - sadrži datoteke kao što su slike, stilovi (.css)
- Controllers - direktorijum koji sadrži kontrolne aplikacije
- fonts - sadrži font datoteke
- Models - direktorijum u kojem se nalaze POCO klase sa kojim predstavljamo entitete aplikacije.
- Scripts - direktorijum u kojem se nalaze JavaScript datoteke
- Views - direktorijum koji sadrži .cshtml stranice i predstavlja prezentacioni sloj aplikacije.
- Global.asax - opcionalna datoteka, služi za inicijalizaciju aplikacije, i možemo da definišemo događaje na aplikacionom nivou i na nivou sesije
- packages.config - sadrži konfiguracije instaliranih paketa u aplikaciji
- Web.config - datoteka koja sadrži konfiguracije za podešavanje web projekta

Kreiranje MVC projekta


Postupak kreiranje ASP.NET veb aplikacije koja podržava MVC je sličan kao što smo kreirali prazan projekat na prethodnim vežbama. Jedina razlika je u poslednjem prozoru na slici 1.

Create a new ASP.NET Web Application


1.


Empty


An empty project template for creating ASP.NET applications. This template does not have any content in it.


Web Forms


A project template for creating ASP.NET Web Forms applications. ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.


MVC

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.


Web API

A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.


Single Page Application

A project template for creating rich client side JavaScript driven HTML5 applications using ASP.NET Web API. Single Page Applications provide a rich user experience which includes client-side interactions using HTML5, CSS3, and JavaScript.

Authentication
No Authentication
[Change](#)

Add folders & core references
☐ Web Forms
☒ MVC 2.
☐ Web API

Advanced
☐ Configure for HTTPS 3.
☐ Docker support
(Requires [Docker Desktop](#))
☐ Also create a project for unit tests

WebApplication1.Tests

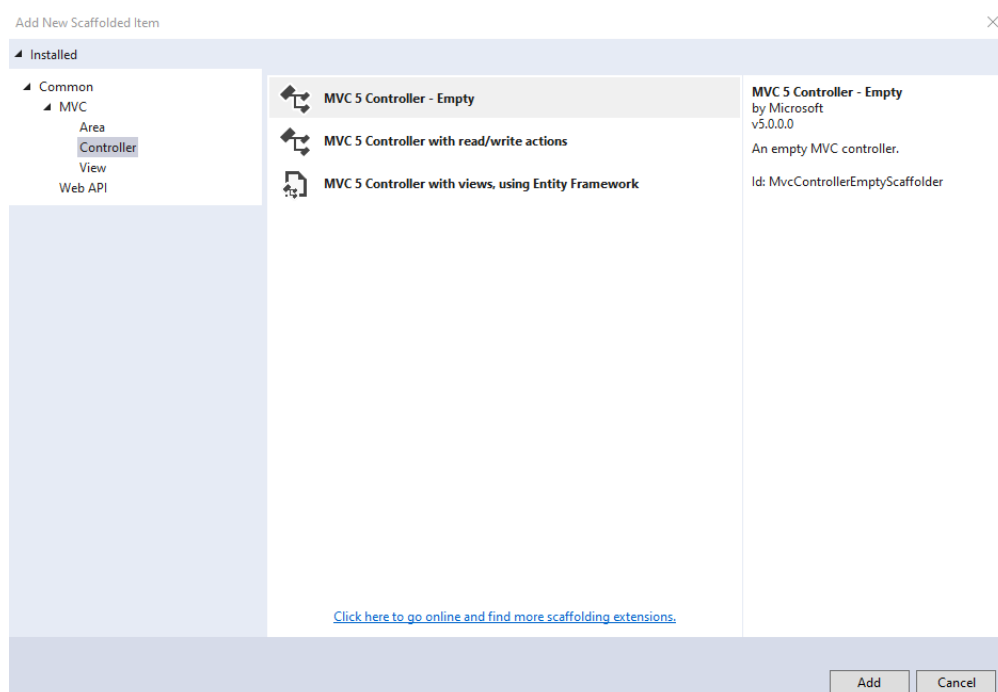
4.

Back

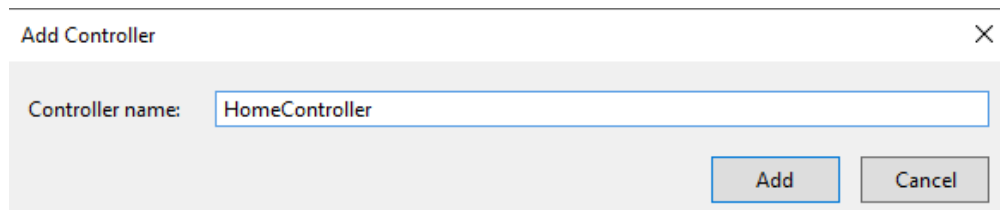
Create

Slika 1. Kreiranje MVC projekta

Kreiranje kontrolera



Slika 1. Kreiranje kontrolera

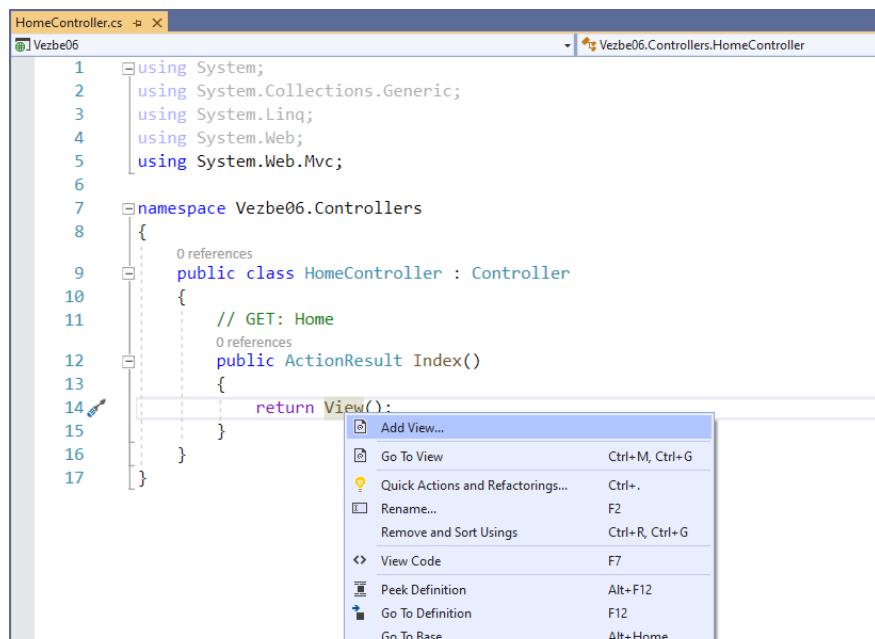


Slika 2. Imenovanje kontrolera

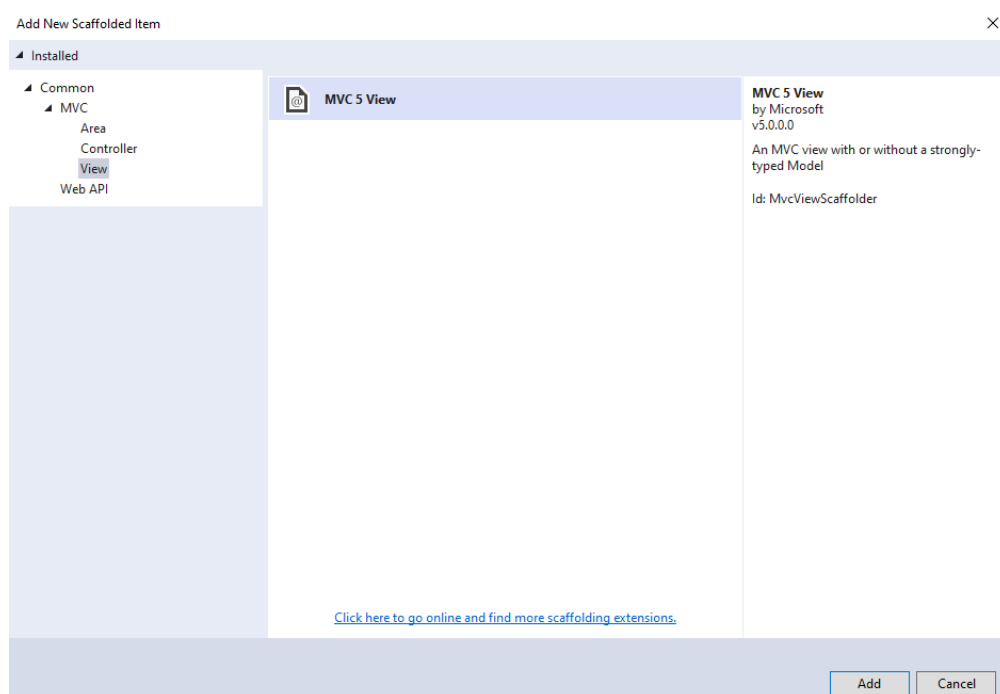
Kreiranje pogleda

Kontroleri se sastoje od akcija i metode koje nisu akcije (Slika 3). Akcije su metode kojim se putem zahteva pristupa po nazivu. Prilikom kreiranja praznog kontrolera inicijalno se kreira akcija (metoda) Index. Ovoj metodi klijent pristupa putem zahteva localhost:port/Home/Index ili localhost:port/Home ili localhost:port. Razlog je prethodno objašnjen u RoutConfig.cs klasi, gde je definisano mapiranje putanje na osnovu koje se zadaju zahtevi.

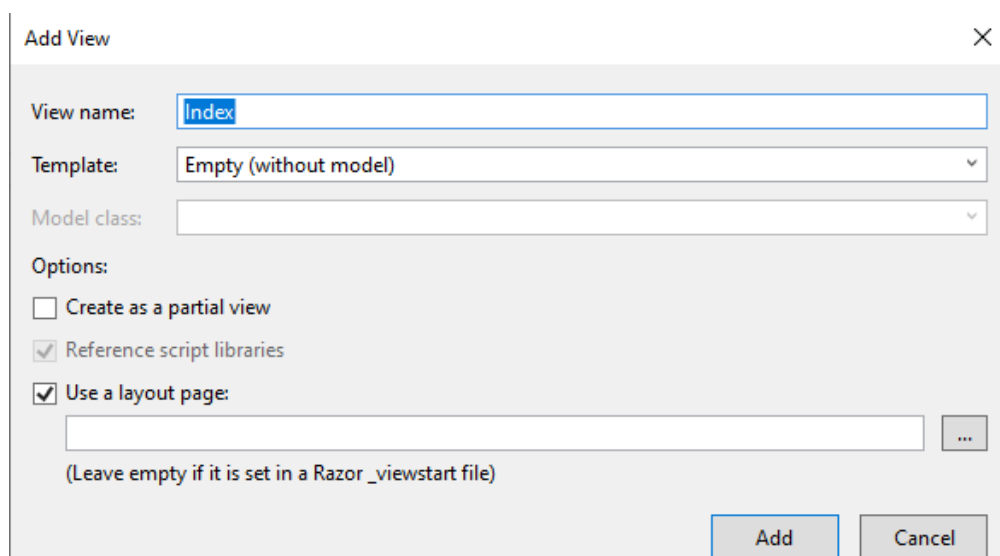
Akcija u ovom primeru vraća View() tj. pogled koji se isto zove kao i Index metoda. (Ako se ništa ne prosledi u View() onda je podrazumevan naziv pogleda isti kao i naziv metode u kojoj se return nalazi). Potrebno je kreirati pogled na sledeći način (Slika 3.)



Slika 3. Kreiranje prikaza (View-a)



Slika 4. Prozor za odabir MVC 5 View



Slika 5. Imenovanje prikaza

Pozivom localhost:port ili localhost:port/Home ili localhost:port/Home/Index vratiće se Index.cshtml pogled (stranica).

Napomena: Stranicama više ne pristupa po nazivu kao što na uvodnom času ASP.NET. Neispravan zahtev localhost:port/Views/Home/Index.cshtml!

Prenos podataka iz kontrolera u pogled

Ukoliko želimo u sklopu pogleda da vratimo još neke podatke (npr. listu elemenata, objekat, .itd) to možemo da postignemo preko sledećih ugrađenih objekata:

- ViewData
 - pristup preko ključa
 - zahteva castovanje za kompleksne tipove
- ViewBag
 - pristup kao kod property (svojstva)
 - nema castovanje za kompleksne tipove

```
In [6]: using System.Web.Mvc;

namespace Vezbe06.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            ViewBag.Message = "Hello, World!";
            return View();
        }
    }
}
```

```
In [7]: @{
    Layout = null;
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<p>@ViewBag.Message</p>
```

- TempData
 - pristup preko ključa
- Model
 - ViewModel - ako želimo da prenosimo više složenih tipova

Dopuniti Index.cshtml (u direktorijumu Views/Home/)

```
In [ ]: @{
    Layout = null;
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<p>@ViewBag.Message</p>

<a href="/~/Users/">Show all users!</a>
```

Dodajte novi Model, klasu User.cs u direktorijum Models

```
In [ ]: namespace Vezbe06.Models
{
    public class User
    {
        public string Username { get; set; }
        public string Password { get; set; }
    }
}
```

Dodati novi UsersController.cs u direktorijum Controllers i prekopirati sledeći kod

```
In [ ]: using System.Collections.Generic;
using System.Web.Mvc;
using Vezbe06.Models;

namespace Vezbe06.Controllers
{
    public class UsersController : Controller
    {
        // GET: Users
        public ActionResult Index()
        {
            List<User> users = new List<User>();
            users.Add(new User
            {
                Username = "pera",
                Password = "pera"
            });
            return View(users);
        }
    }
}
```

Kreirajte pogleda koji prikazuje listu korisnika (složeni tip podatka, klasu User) (za sada je i dalje Empty template prilikom kreiranja View, sledeće vežbe radićemo sa HTML helper-ima)

```
In [ ]: @using Vezbe06.Models
        @model IEnumerable<User>

        @{
            Layout = null;
        }

        <!DOCTYPE html>

        <html>
        <head>
            <meta name="viewport" content="width=device-width" />
            <title>Index</title>
        </head>
        <body>
            <div>
                @foreach(var user in Model)
                {
                    <p>@user.Username</p>
                }
            </div>
        </body>
        </html>
```

Prenos podataka iz pogleda u kontroler

Ukoliko želimo da prosledimo podatke iz pogleda u kontroler to možemo da postignemo uz pomoć:

Prvi način je preko Request objekta

- ugrađen objekat
- vezan je za zahtev
- čita input parametre forme po nazivu Request["nazivInputPolja"]

Dopuniti Index.cshtml (View/Users)

```
In [9]: @using Vezbe06.Models
        @model IEnumerable<User>

        @{
            Layout = null;
        }
        <div>
            @foreach(var user in Model)
            {
                <p>@user.Username</p>
            }
        </div>

        <form method="post" action="~/Users/Add">
            <input name="username" />
            <br />
            <input name="password" />
            <br />
            <input type="submit" />
        </form>
```

Prepraviti UsersController.cs


```

In [ ]: using System.Collections.Generic;
        using System.Web.Mvc;
        using Vezbe06.Models;

        namespace Vezbe06.Controllers
        {
            public class UsersController : Controller
            {
                // GET: Users
                public ActionResult Index()
                {
                    List<User> users = (List<User>)HttpContext.Application["users"];
                    if (users == null)
                    {
                        users = new List<User>
                        {
                            new User
                            {
                                Username = "pera",
                                Password = "pera"
                            }
                        };
                    }

                    HttpContext.Application["users"] = users;
                    return View(users);
                }

                [HttpPost]
                public ActionResult Add()
                {
                    List<User> users = (List<User>)HttpContext.Application["users"];

                    var username = Request["username"] != null ? Request["username"] :
string.Empty;
                    var password = Request["password"] != null ? Request["password"] :
string.Empty;

                    users.Add(new User
                    {
                        Username = username,
                        Password = password
                    });

                    return RedirectToAction("Index");
                }
            }
        }

```

Metoda RedirectToAction redirectuje na akciju. Na osnovu prosleđenog parametra pozvaće se odgovarajuća akcija.

1. Ukoliko želimo preusmerimo na akciju koja je u istom kontroleru onda zadajemo samo naziv akcije
 - `return RedirectToAction("Index")` redirectujemo na akciju Index koja se nalazi u UsersController-u
2. Ukoliko želimo da preusmerimo na akciju koja se nalazi u drugom kontroleru onda zadajemo nazivkontrolera/akcija
 - `return RedirectToAction("Home/Index")` redirectujemo na akciju Index koja se nalazi u HomeController-u

Kao rezultat akcije možemo da vratimo i pogled:

1. Ukoliko želimo da rezultat akcije bude View koji se zove kao ta akcija
 - unutar UsersController.cs, akcija Add() sa samo `return new View()`, vraća Add.cshtml
2. Ukoliko ne želimo da rezultat akcije bude View koji se zove kao ta akcija, kao parametar navodimo naziv pogleda gde želimo da preusmerimo klijenta
 - unutar UsersController.cs, akcija Add() sa samo `return new View("Index")`, vraća Index.cshtml koji se nalazi u Views/Users/ direktorijumu
 - Napomena: pogled koji je zadat samo po nazivu mora da bude unutar direktorijuma koji je povezan sa datim kontrolerom tj. (Views/Users/).
3. Ako želimo da vratimo pogled koji je u sklopu drugog kontrolera (npr. Index.cshtml koji se nalazi u Views/Home/), onda moramo da zadamo prvo naziv kontrolera pa tek onda naziv pogleda
 - unutar UsersController.cs, akcija Add() sa samo `return new View("Home/Index")`, vraća Index.cshtml koji se nalazi u Views/Home/ direktorijumu

Napomena: Da bi metoda bila akcija mora da ima public modifikator pristupa! Ako akcija prihvata post zahteve mora da ima anotaciju iznad naziva [HttpPost], a ne navedete anotaciju podrazumevana vrednost je [HttpGet].

Drugi način je Model binding:

- manuelno povezivanje
 - FormCollection
- automatsko povezivanje
 - Preko POCO klase

Dopunite Index.cshtml (Views/Users)

```

In [ ]: @using Vezbe06.Models
        @model IEnumerable<User>

        @{
            Layout = null;
        }
        <div>
            @foreach (var user in Model)
            {
                <p>@user.Username</p>
            }
        </div>

        <p>Add by Request</p>
        <form method="post" action="~/Users/Add">
            <input name="username" />
            <br />
            <input name="password" />
            <br />
            <input type="submit" />
        </form>

        <p>Add by Model</p>
        <form method="post" action="~/Users/AddModel">
            <input name="username" />
            <br />
            <input name="password" />
            <br />
            <input type="submit" />
        </form>

```

Dopunite UsersController.cs

```

In [ ]: using System.Collections.Generic;
        using System.Web.Mvc;
        using Vezbe06.Models;

        namespace Vezbe06.Controllers
        {
            public class UsersController : Controller
            {
                // GET: Users
                public ActionResult Index()
                {
                    List<User> users = (List<User>)HttpContext.Application["users"];
                    if (users == null)
                    {
                        users = new List<User>
                        {
                            new User
                            {
                                Username = "pera",
                                Password = "pera"
                            }
                        };
                    }

                    HttpContext.Application["users"] = users;
                    return View(users);
                }

                [HttpPost]
                public ActionResult Add()
                {
                    List<User> users = (List<User>)HttpContext.Application["users"];

                    var username = Request["username"] != null ? Request["username"] :
string.Empty;
                    var password = Request["password"] != null ? Request["password"] :
string.Empty;

                    users.Add(new User
                    {
                        Username = username,
                        Password = password
                    });

                    return RedirectToAction("Index");
                }

                [HttpPost]
                public ActionResult AddModel(User user)
                {
                    List<User> users = (List<User>)HttpContext.Application["users"];
                    users.Add(user);
                    return RedirectToAction("Index");
                }
            }
        }

```

Napomena: Prilikom prosleđivanja podataka sa pogleda na kontroler, naziv input polja moraju da se poklapaju sa nazivim properties (svojstvima) klase Users.

Praćenje sesije

HTTP protokol je stateless protokol. Server ne čuva status klijenta ili korisničke sesije, zahtevi koji se šalju od strane klijenta (klijenata) su nezavisni.

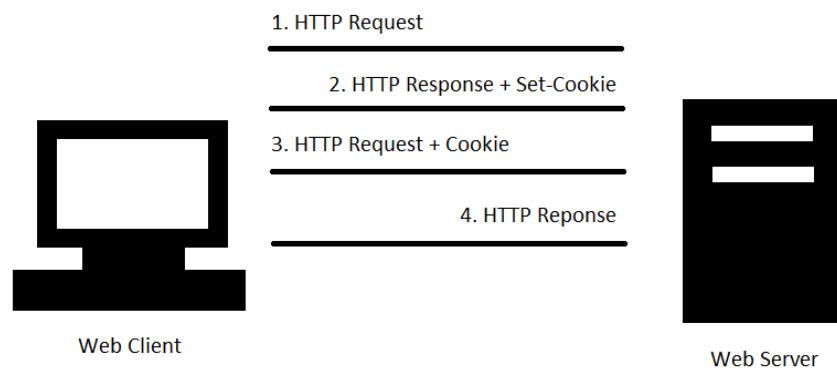
Kako bismo omogućili čuvanje podataka o ulogovanom korisniku postoje metode za održavanje i upravljanje sesijom.

1. Praćenje sesije preko Session objekta

- neophodno je kreirati POJO klasu koja će nam predstavljati korisnika u našem sistemu
- prilikom uspešne prijave (login) na sistem u Session dodajemo korisnika npr. `Session["LOGINUSER"] = prijavljenKorisnik`

2. Praćenje sesije preko trajnog Cookie mehanizma (Slika 7.)

- kako se ne bismo prijavljivali na sajt svaki put kada uđemo ponovo na stranicu, postoje trajni cookie
 - prilikom prve prijave dodaje se trajni cookie
 - prilikom provere da li je korisnik prijavljen, proverava se da li ima trajni cookie
- ako veb čitač ne prihvata cookie mehanizam, onda se koristi URL Rewriting mehanizam



Slika 7. Prijava na sistem, cookie mehanizam

File Upload

Prilikom implementacije file upload neophodno je proslediti post zahtev sa enctype multipart/form-data format.

Forma treba da ima attribute `method="post"` i `enctype="multipart/form-data"`. Unutar forme treba da imate input sa atribut `type="file"` i input `type="submit"`. Tag `input type="file"` bitan je i atribut `name` npr. `name="file"`.

Na serverskoj strani, u kontroleru, možete upload da realizujete na nekoliko načina.

Jedno moguće rešenje je sledeće:

U kontroleru koji je zadužen za zahteve sa .cshtml stranice gde smo implementirali file upload prikaz, treba da imamo akciju anotiranu sa `[HttpPost]` koja kao parametar ima tip klase `HttpPostedFileBase`. Naziv parametra mora da se poklapa sa nazivom `input type="file"` taga. U našem primeru naziv parametra je `file`, poklapa se sa nazivom `input type="file" name="file"` npr. akcija `UploadFile(HttpPostedFileBase file)`. Unutar `UploadFile` akcije potrebno je definisati naziv fajla i putanju do direktorijuma (koji se nalazi na serveru, tj. putanja do direktorijuma na vašem računaru) gde će se fajl sačuvati. Dodati sve fajlove (po nazivu i putanji) unutar `Application` objekta, kako bi se svi fajlovi koji su uspešno uploadovani ispisali krajnjem korisniku na prikazu u tabeli.

Zadaci

Zadaci za ovu nedelju su prva tri zadatka koji se nalaze sekciji Zadaci sa vežbi>Web programiranje>ASP.NET MVC zadaci.

Četvrti zadatak se radi sledeće nedelje uz pomoć HTML helper-a.

Za prvi zadatak potrebno je razviti MVC prazan projekat i kreirati registrovanje, i ispis svih registrovanih korisnika, pretragu i brisanje korisnika. Ovaj zadatak je bez logovanja.

Drugi zadatak treba da se implementira Web Shop aplikacije. Kao referentni projekat koristiti WebShopMVC primer sa predavanja (ili dopunite WebShopMVC zadatak). Akcenat je na autentifikaciji korisnika. Treba da se omogući registrovanje i prijava korisnika. Potrebno je implementirati novi kontroler za autentifikaciju. Prilikom prijave korisnik se upisuje u sesiju i kreira mu se korpa. Korisnik (Kupca) može da dodaje proizvode u korpu, na kraju ima opciju kupi (buy). Nije neophodno kreirati CRUD funkcionalnosti za proizvode. Ne zaboravite da kreirate i opciju odjavu sa sistema. Prilikom odjave briše se korisnik i njegova korpa iz sesije.

Treći zadatak je implementacija file upload-a. Sličan primer je rađen na predavanju.

Napomena: Dodatno, pokušajte da implementirate autorizaciju tj. prava pristupa stranicama u zavisnosti od ulogovanog (uloge) korisnika. (Drugi zadatak).

Koja je razlika između autorizacije i autentifikacije?