.NET Okruženje

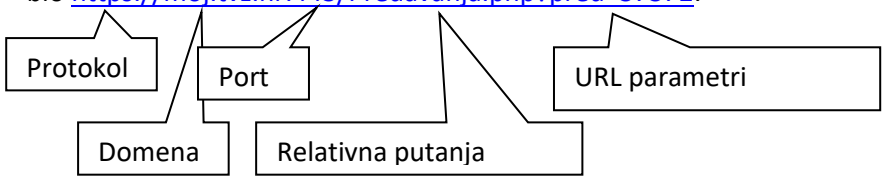
Vježba 4: HTML, Bootstrap, Basic Routing

Sadržaj

Klijent-server komunikacija

Komunikacija između klijenta (primjerice, web browsera) i servera se odvija putem zahtjeva i odgovora – klijent šalje zahtjev (request), a server zaprima zahtjev i šalje odgovor (response). Zahtjev je definiran s nekoliko važnijih dijelova:

- **URL** – svaki zahtjev je naslovljen na neku URL adresu, koja se sastoji od protokola, domene, (porta), relativne putanje (relative path), URL parametara. Primjer jednog takvog zahtjeva bi bio https://moj.tvz.hr:443/Predavanja.php?pred=37372.

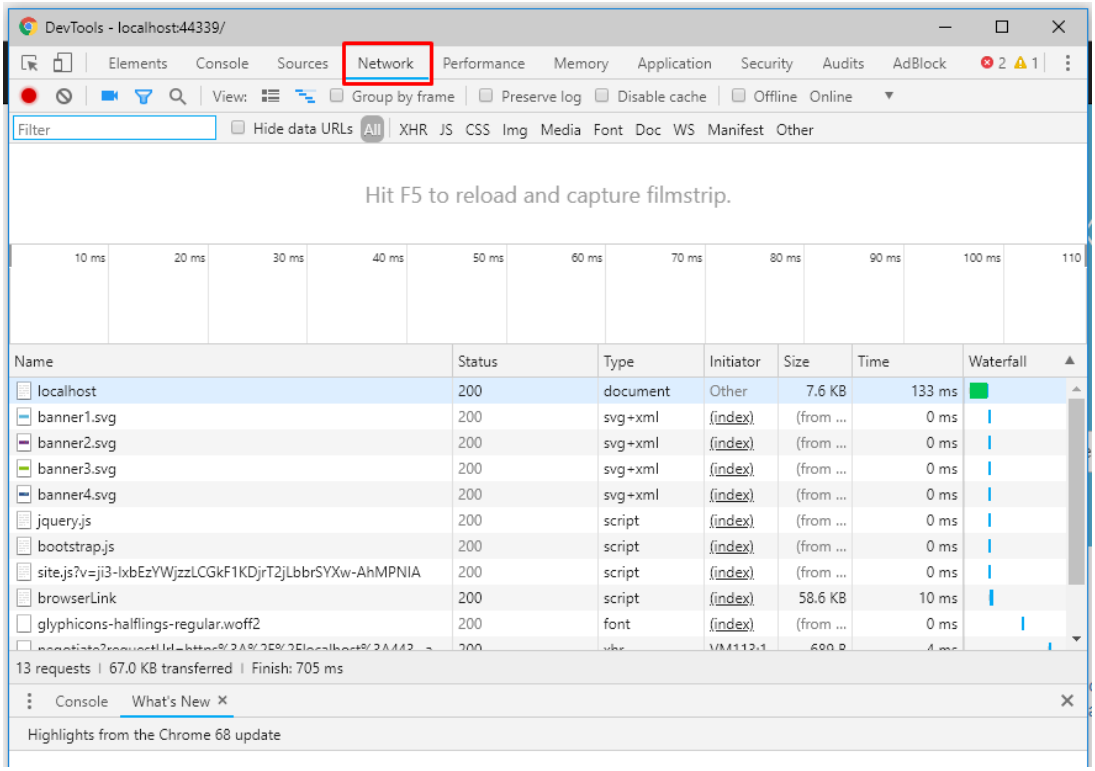


- **Zaglavlje** – dodatni parametri koji specificiraju primjerice jezik, format očekivanog sadržaja, autentikacijske podatke i sl.
- **POST vrijednosti** – zahtjev može sadržavati i dodatne parametre i vrijednosti
- **Tip zahtjeva** – GET, POST, PUT, DELETE

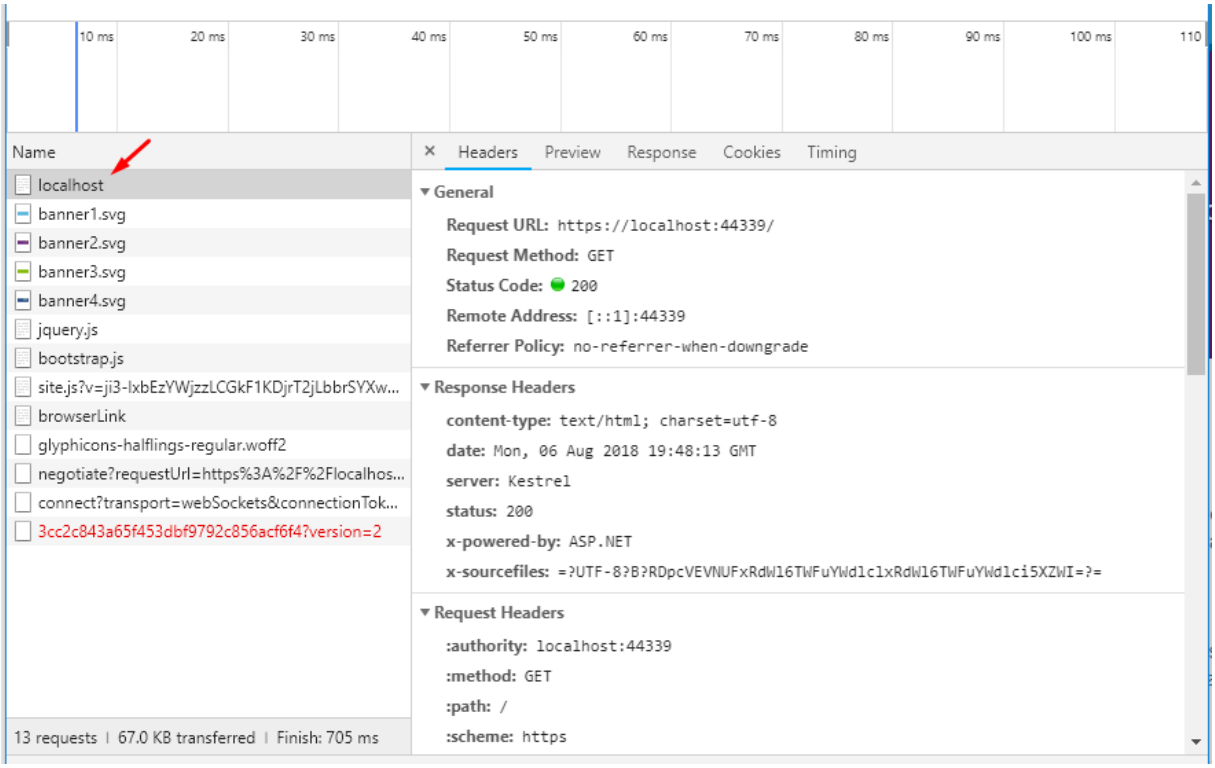
Postoji nekoliko tipova zahtjeva, mi ćemo istaknuti sljedeća dva:

- **GET** – zahtjev koji šalje parametre preko URL-a, i kao povratnu vrijednost dobiva (najčešće) HTML koji se prikazuje korisniku u web pregledniku
- **POST** – isto kao GET, no šalje dodatne parametre POST metodom koji mogu biti obrađeni na serveru. Najčešće se na ovaj način šalju podaci koji se popunjavaju u formi na web stranici.

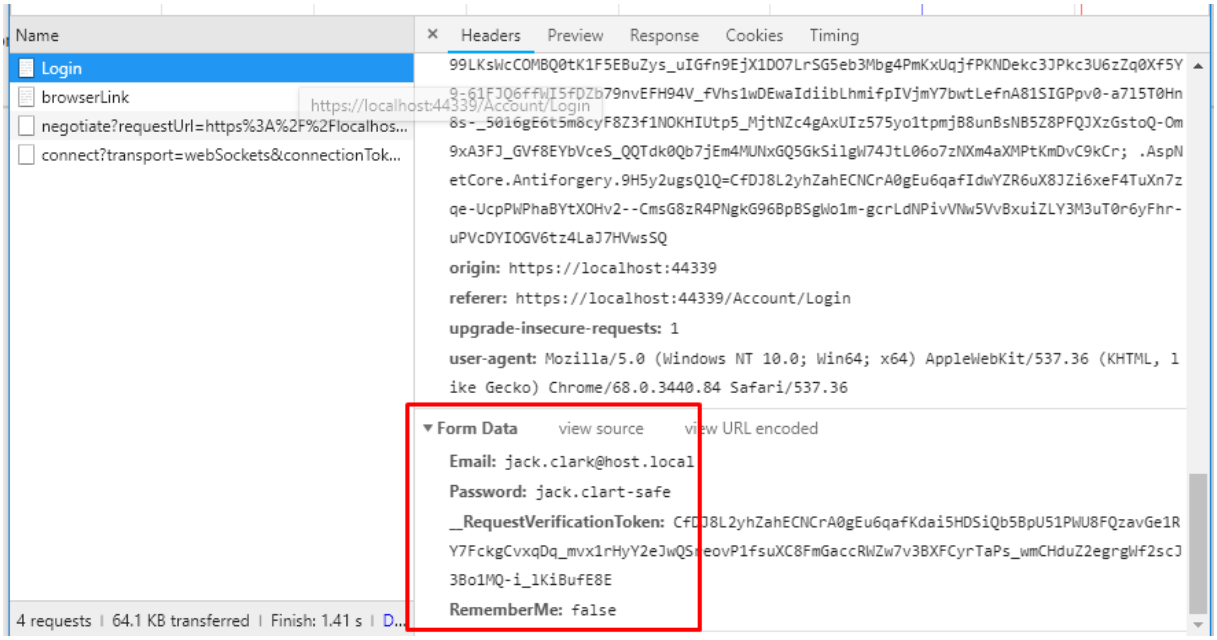
Komunikaciju servera i klijenta možemo pregledavati u posebnom dijelu developer tools alata:



Ukoliko napravimo reload stranice na kojoj smo trenutno, možemo pregledati točno kako izgleda zahtjev i kako izgleda odgovor:



Ukoliko otvorimo stranicu koja na sebi ima formu za unos podataka, možemo vidjeti i naše podatke koji se šalju POST metodom na server (pomoću istog alata):



Kako bi mogli ostvariti komunikaciju servera i klijenta, potrebno je poznavati (barem) HTML jezik.

Upoznavanje s MVC paradigmom

MVC je skraćenica sa Model View Controller. Samim time lako je izolirati 3 bitna pojma koji se koriste u MVC paradigmi:

- **View** – kombinacija HTML koda i server side naredbi. U konačnici, view se pretvara isključivo u HTML kod koji se može prikazati u internet preglednicima. Pri kreiranju tog HTML koda se koriste razne server-side naredbe koje generiraju odgovarajući HTML.
- **Model** – C# klasa koja definira podatke koji se prikazuju u odgovarajućem view-u. Ta ista klasa se u jednostavnijim projektima koristi i u raznim ORM alatima za spremanje u bazu podataka. U kompleksnijim projektima, to su posebne klase koje sadrže prilagođene podatke. Mi ćemo se u ovom kolegiju pri govoru o modelima u pravilu referirati na njihovu upotrebu u jednostavnijim projektima.
- **Controller** – C# klasa koja se sastoji od niza akcija koje se povezuju s određenim URL adresama

Dodatno, bitni su sljedeći pojmovi:

- **Akcija controllera** – jedan controller ima jednu ili više akcija koje kao rezultat vraćaju generirani HTML kod. Akcija controllera sadrži logiku pomoću koje kreira odgovarajuću **instancu model klase**, te model klasu prosljeđuje **odgovarajućem view-u**, koji serverskim naredbama izvlači podatke iz model klase i na temelju njih generira odgovarajući HTML kod koji se vraća kao rezultat akcije.[1]
- **ViewModel** – dodatni pojam koji označava pomoćnu klasu koja sadrži sve prilagođene podatke koje koristi view da bi ispravno iscrtao odgovarajući HTML.

Razlika između *Model* klase i *ViewModel* klase je u tome što *Model* klasa sadrži samo podatke koji se spremaju u bazu, dok *ViewModel* može sadržavati dodatne redundantne podatke ukoliko je to potrebno. U velikoj većini osnovnih slučajeva, *Model* klasa je dovoljna; no u kompliciranijim scenarijima potrebno je uvesti dodatnu, *ViewModel* klasu.

Samim time, možemo reći da akcija controllera obrađuje parametre koje je dobila, izvršava potrebnu poslovnu logiku kojom nastaje odgovarajuća instanca model klase, a kao rezultat se iz view datoteke (cshtml) generira odgovarajući kod. Općenito, praksa je da se u view datoteke ne stavlja nikakva logika osim:

- **Jednostavnog grananja** – if(Model.SomeProperty == true) { ... } else { .... }
- **Petlji** – foreach(var item in Model.Items) { .... }
- **TagHelper elemenata** – Novitet u .NET Core radnom okviru

Ukoliko se kompleksnija logika od gore navedene nalazi u view-u, vjerojatno je posrijedi konceptualna pogreška.

Česta je praksa da za jedan semantički pojam (primjerice, Quiz) imamo jednu Controller klasu (**QuizController.cs**), nekoliko potrebnih view model klasa (**Quiz.Models.cs** datoteka – više klasa u istoj datoteci), te nekoliko view datoteka (najčešće **Index.cshtml, Create.cshtml, Edit.cshtml,**

[1] Rezultat ne mora biti nužno HTML kod, no zasad ćemo na taj način definirati rezultat akcija controllera.

Details.cshtml, po potrebi i druge). Dodatno, vrlo vjerojatno postoji i model klasa **Quiz** koja se koristi za spremanje u bazu podataka.

Controller

Pri obradi zahtjeva se stvara instanca odgovarajuće Controller klase. Svaka controller klasa treba naslijediti iz Controller klase, čime nasljeđuje i postojeće metode koje će detaljnije biti objašnjene kasnije. Unutar controller-a nalaze se **akcije** (primjer jedne akcije LoginController-a je u nastavku), a svaka akcija se sastoji od sljedećih bitnih elemenata:

- **Naziv akcije** – zapravo je naziv metode. U donjem primjeru to je „Login“
- **Async/sync** – akcija može biti označena s async, međutim na samo izvođenje nema utjecaja u okruženju s malo korisnika
- **Anotacija** – anotacije pobliže opisuju ponašanje akcije, i nalaze se u **[**uglatim zagradama**]** iznad naziva metode.
- **Povratni tip** – Najčešće to je IActionResult objekt (kao u donjem primjeru), koji je povratni tip koji obuhvaća sve standardne povratne vrijednosti neke akcije, u pravilu objekt koji nastaje kao posljedica renderiranja view predloška.
- **Parametri** – parametri koji se prenose preko URL-a ili preko forme. U donjem slučaju imamo jedan parametar tipa **string**, koji se zove **returnUrl**. Primjerice, kod poziva ovog Controller-a preko URL-a taj parametar bi napunili ovako: „/Account/Login?returnUrl=…“. S obzirom na tip parametra, vrijednost se automatski konvertira (primjerice, moguće je staviti odmah int, bool ili neki drugi tip)
- **Vraćanje rezultata** – Nakon obrade podataka, potrebno je vratiti rezultat. Najčešće će to biti pozivom View() metode, koja prima dva parametra:
    - **Naziv predloška** – naziv cshtml datoteke prema kojoj se generira HTML. Ukoliko ovaj parametar nije proslijeđen, naziv view-a se dodijeljuje automatski prema nazivu akcije controller-a[2].
    - **Model** – objekt koji sadrži podatke koje prenosimo u view. Ukoliko se ne proslijedi, prilikom generiranja HTML-a ne možemo koristiti @Model objekt.

```
[HttpGet]
[AllowAnonymous]
public async Task<IActionResult> Login(string returnUrl = null)
{
    // Clear the existing external cookie to ensure a clean login process
    await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);

    ViewData["ReturnUrl"] = returnUrl;
    return View();
}
```

View

Do sada smo jedino pomoću ViewData objekta ispisivali podatke u .cshtml datoteci, međutim to nije najoptimalniji način ispisa iz razloga što **ViewData objekt nije strongly typed**, tj., mapa (dictionary).

[2] Postoji još pravila vezano za pronalazak odgovarajućeg predloška, no o tome kasnije.

Puno je bolje i sigurnije koristiti tipizirani model ili view model uz odgovarajući view. Objekt koji prenosimo kao model je najčešće jednostavna C# klasa koju smo sami kreirali, ili kolekcija klasa koje smo sami kreirali. Kao što je već spomenuto, objekt koji predstavlja model se puni u akciji controllera i proslijeđuje u view preko direktive @model na vrhu cshtml datoteke, kao što se može vidjeti na donjem primjeru:

```
@model LoginViewModel

@{
    ViewData["Title"] = "Log in";
}

<h2>@ViewData["Title"]</h2>
```

Ukoliko tip modela nije prepoznat automatski, potrebno je napraviti jedno od sljedećih:

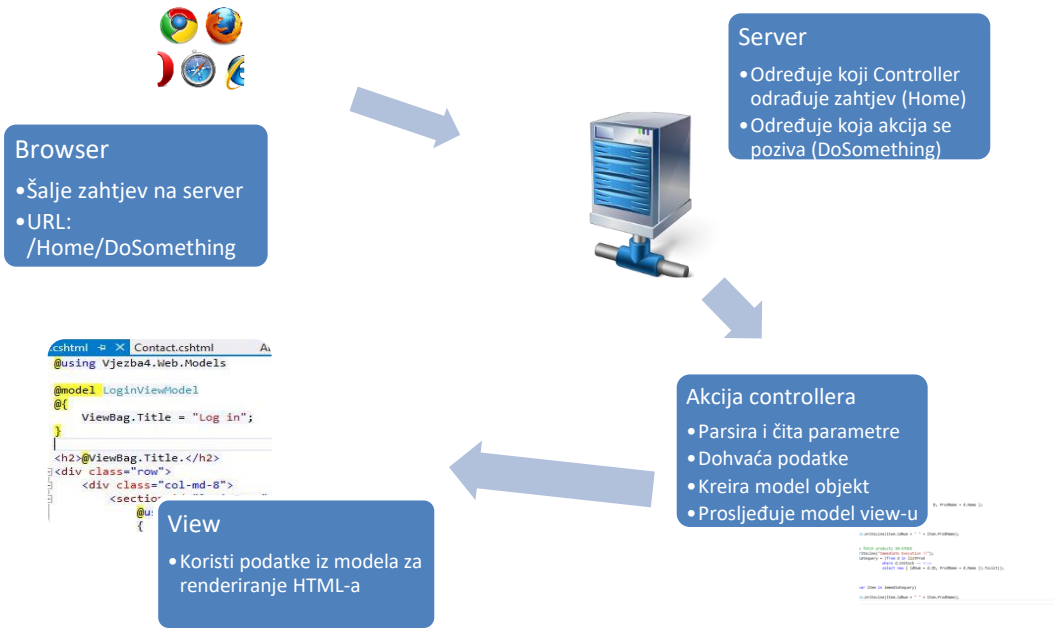
- u _ViewImports datoteci dodati odgovarajući namespace pomoću @using, ili
- koristiti @using direktno u view-u:
  @using QuizManager.Web.Models.AccountViewModels.LoginViewModel
- koristiti puno ime klase - QuizManager.Web.Models.AccountViewModels.LoginViewModel

Željeni podaci se ispisuju unutar HTML-a naredbom **`@Model.Svojstvo`**, kao što je vidljivo u donjem primjeru:

```
<div>
    IP: @Model.IPAddress
</div>
```

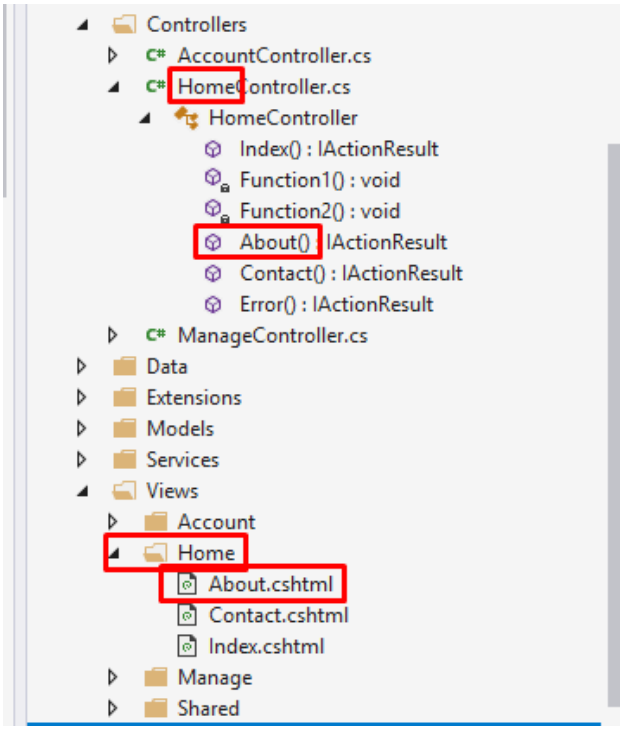
Na ovaj način, pri generiranju HTML-a će se **@Model.IPAddress** zamijeniti sa vrijednosti koja je zapisana u svojstvu IPAddress objekta kojieg smo proslijedili kao model.

Općenito, procesiranje jednog zahtjeva u ASP.NET MVC tehnologiji izgleda kao što je demonstrirano na sljedećem dijagramu (od poziva URL metode do vraćanja HTML stranice):



Nomenklatura i konvencije

U screenshotu s lijeve strane, uočiti nazive datoteka, klasa i metoda. Na primjeru HomeController controllera i About metode, pogledajmo kako izgleda struktura:



- U folderu Controllers, nalazi se HomeController. Svaki kontroler završava sufixom „Controller“.
- HomeController sadrži nekoliko akcija – promotrimo akciju About.
- Akcija About vraća „return View()“. O kojem točno view-u se radi, povezuje se preko naziva akcije. Konkretno u ovom slučaju, iz foldera Views/Home se pokušava locirati .cshtml datoteka istog imena kao i akcija – u ovom slučaju About.cshtml.
- Struktura Views odgovara strukturi kontrolera – uočimo foldere Home, Manage, Account – to su upravo nazivi kontrolera iz mape Controllers.

Sve od gore navedenog je moguće prilagoditi, ali za većinu situacija upravo ovakva osnovna postavka je dovoljna.

URL parametri akcije

Ranije je spomenuto kako je u akciju controller-a moguće proslijediti i parametre preko URL query stringa. Primjerice, sljedeći URL se veže na akciju controllera kako je skicirano na donjoj shemi:

http://localhost/Home/About

Controller: HomeController
Action: About

```
public class HomeController : Controller
{
    public IActionResult About()
    {
        //Obrada
        //...

        return View();
    }
}
```

Međutim, ukoliko URL promijenimo na način da proslijedimo parametar koji želimo pročitati pri obradi, URL i kod mogu izgledati ovako:

http://localhost/Home/About?**lang**=hr

Controller: HomeController
Action: About
Parametar: lang, vrijednost parametra = hr

```
public class HomeController : Controller
{
    public IActionResult About(string lang)
    {
        //Obrada
        //...

        return View();
    }
}
```

Vrijednost varijable **lang** u ovoj akciji za taj specifični URL će biti string "hr".

URL usmjeravanja (routing)

Već je ranije spomenuto da server, pri obradi zahtjeva, usmjerava zahtjev na odgovarajući controller i odgovarajuću akciju, i prenosi eventualno potrebne parametre. Kako bi server znao, za neki URL, koju akciju kojeg controllera da aktivira, potrebno je specificirati pravila usmjeravanja (routes).

Pravila usmjeravanja su definirana u datoteci **Startup.cs**, u korijenskoj datoteci Web projekta. Da bi uspješno definirali usmjeravanja u aplikaciji, bitno je znati sljedeće:

- Jedna definicija rute se može primjeniti na više controller-a i akcija – naime, definicija rute podržava parametrizaciju naziva controllera i naziva akcije
- Rute su procesirane jedna za drugom, poštivajući redoslijed, te je aktivirana prva ruta koja zadovoljava parametrima zahtjeva

Routes – jednostavni primjer

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    ...

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

Pogledajmo gornju jedinu definiranu rutu – osnovnu rutu (otisnuto masnim slovima). U ovoj fazi, najbitniji parametar nam je „template“, i to je url shema preko koje se aktivira ruta. Primjerice, gornja ruta definira da se URL shema može sastojati od najviše 3 dijela (recimo HOST/Xxx/Yyy/123), gdje se iz prvog parametra (Xxx) iščitava naziv odgovarajućeg Controller-a – u ovom slučaju to bi bila klasa XxxController; iz drugog parametra se iščitava naziv odgovarajuće akcije u Controlleru – to bi bila akcija (ActionResult Yyy(...) { }); a iz trećeg dijela se izvlači parametar koji se prosljeđuje akciji (ActionResult Yyy(int id), gdje parametar „id“ poprima vrijednost 123). Id nije nužno cijeli broj – može biti i string, iako u većini aplikacija se iza imena id očekuje cijeli broj.

Važna napomena: nomenklatura u ASP.NET MVC je veoma bitna, i treba poštivati određena pravila. Ukoliko je potrebno, moguće je ta pravila zaobići i prilagoditi potrebama, no to je preporučljivo samo u iznimnim situacijama. Evo nekoliko bitnijih pravila:

- Controller klasa za pojam „Xyz“ će se zvati XyzController.
- Controller klasu je obvezno staviti na istu razinu s ostalim controllerima
- View datoteke vezane uz taj controller, moraju biti u mapi Views/Xyz/...
- Pri definiranju rute, ne koristi se naziv XyzController, nego samo Xyz

U ovoj vježbi koristit će se isključivo jednostavni routing na način da je prva komponenta URL-a naziv Controller-a a druga komponenta naziv odgovarajuće akcije. Primjerice, poziv **Home/Contact** će izvršiti **HomeController**, akcija **Contact**.

URL rute – naredbe ActionLink i Url

Česti je slučaj da je potrebno napraviti preusmjeravanje na željenu stranicu unutar ASP.NET MVC aplikacije iz samog HTML-a ili iz JavaScript funkcije – primjerice kod izrade izbornika. Zbog načina kako su rute napravljene i održavanja istih, bilo bi izuzetno nezgodno da se takvo održavanje radi ručnim definiranjem linkova (<a href="..."), već imamo na izbor koristiti nekoliko mogućnosti kojima se kao parametri mogu proslijediti naziv željenog controllera i naziv željene akcije, te kao rezultat dobijemo URL rutu ili cjelokupni link koji se u konačnici u obliku HTML-a prenosi na preglednik klijenta:

- **@Html.ActionLink()** – kao rezultat vraća <a> element koji služi za redirekciju. Podržava konfiguraciju proizvoljnih html parametara, kao što su dodatni atributi i/ili css klasa, itd. Ukoliko je potrebno prenijeti parametar (primjerice, id parametar u neku željenu akciju), potrebno je predati novi (anonimni) objekt kao parametar naziva „routeValues"
  - `@Html.ActionLink("Home", "Index", "Home", routeValues: new { lang = "en" })`
- **@Url.Action(...)** – kao rezultat vraća string koji predstavlja željenu rutu, generiranu na temelju controllera i akcije. Taj string se može iskoristiti unutar <a href="..."> elementa ili unutar javascript funkcije
  - `<a href="@Url.Action("About", "Home", values: new { lang = "en" })">About</a>`
- **<a asp-*> Tag Helper** – .NET Core uvodi mogućnost definiranja vlastitih HTML elemenata, te donosi niz već ugrađenih korisnih elemenata. Jedan od njih je element za link (<a>)
  - `<a asp-controller="Home" asp-action="Contact" asp-route-lang="en">Contact</a>`

Svaka od gornjih naredbi generira URL koji odgovara onima definiranima u tablici usmjerenja (*Route Table*).

_Layout.cshtml

```
<ul class="nav navbar-nav">
    <li>
        @Html.ActionLink("Home", "Index", "Home", routeValues: new { lang = "en" })
    </li>
    <li>
        <a href="@Url.Action("About", "Home", values: new { lang = "en" })">About</a>
    </li>
    <li>
      <a asp-controller="Home" asp-action="Contact" asp-route-lang="en">
        Contact
      </a>
    </li>
```

Vraća kompletni <a> tag.
Parametri:

1. Tekst koji se prikazuje korisniku
2. Naziv akcije
3. Naziv controllera

Generira link koristeći TagHelper:

1. Naziv controller-a
2. Naziv akcije
3. Parmetri

Vraća samo URL kao string. Parametri:

1. Naziv akcije
2. Naziv controllera

```
▼<ul class="nav navbar-nav">
    ::before
  ▼<li>
      <a href="/">Home</a>
    </li>
  ▼<li>
      <a href="/Home/About">About</a>
    </li>
  ▼<li>
      <a href="/Home/Contact">Contact</a>
    </li>
    ::after
  </ul>
```

Zadatak 4.1

Kreirati novu stranicu Home/FAQ, koja ima 5 pitanja i odgovora (biti kreativni, ali ne previše). Na stranicu *Privacy*, dodati linkove na:

- FAQ stranicu, da se pritom selektira pitanje broj 4. Poslati preko URL-a parametar „selected=4“
- FAQ stranicu bez da se selektira pitanje (preko URL-a se ne šalje parametar)

Hint: Za oba linka treba biti identična akcija controller-a koja opcionalno prima parametar **int? selected = null**. Selektirano pitanje postaviti da bude „bold“.

Osnove HTML jezika

S obzirom da je znanje HTML-a usvojeno na obveznom kolegiju u prvom semestru, i kasnije primjenjeno, u ovom poglavlju dat će se samo osnovni pregled bitnijih HTML elemenata i principa.

Container elementi

- **<html>** – korijenski element stranice. Unutar njega su još <head> i <body>
- **<head>** - sadrži direktive kao što su meta podaci o stranici, naslov, reference na klijentske skripte ili stilove
- **<body>** - element koji sadrži ostale elemente za prikaz sadržaja
- ** ** – označava najčešće korišteni container element koji zauzima cijelu liniju u HTML strukturi (osim ako mu je postavljeno plutanje – float). Unutar njega elementi se slažu ovisno o svojim postavkama
- **** - element koji neće zauzeti cijelu liniju, već samo onu širinu koja mu je potrebna. U njega se dalje mogu slobodno umetati ostali elementi
- **<table>** - služi za tablični raspored elemenata. Koristi dodatno elemente **<th>**,**<tr>**, **<td>**

Elementi za rukovanje unešenim vrijednostima

- **<form>** - element unutar kojeg se nalaze **<input>** elementi čije vrijednosti se šalju POST metodom na server
- **<input>** - postoji nekoliko različitih tipova <input> elemenata, gdje svaki služi za razni standardni tip podataka koji se treba poslati na server
  - **<input type="text" … />** - jednostavni element za unos teksta
  - **<input type="submit">** - iscrtava se u obliku gumba, na čiji pritisak se cijela forma i njene vrijednosti šalju na server[3]
- **<select>** - vrlo sličan input elementu, no sadrži predefiniran skup vrijednosti koje se mogu odabrati
- **<textarea></textarea>** - kontrola za unos teksta koja podržava više linija unešenog teksta

Zadatak 4.2: Modificirat ćemo *Contact.cshtml* datoteku na način da dodamo **form** element i nekoliko input elemenata i pošaljemo podatke koje će korisnik unijeti na server te ih nakon toga ispišemo korisniku na drugoj stranici.

1. Proučiti **HomeController** klasu, posebno akcije **Contact**() i **SubmitQuery**(), te pročitati komentare u kodu
2. U **Contact.cshtml** kreirati formu (HTML form element) koja šalje podatke na URL /Home/SubmitQuery.
3. Dodati kontrole za unos sljedećih podataka:
   a. Ime i prezime
   b. Email
   c. Poruku (multiline)
   d. Tip poruke (predefinirana lista vrijednosti: pohvala, upit, poslovni prijedlog, žalba)

[3] Više o drugim tipovima kontrola za unos može se naći ovdje:
http://www.w3schools.com/html/html_form_input_types.asp

   e. Checkbox kontrolu za označavanje da želimo primati newsletter
   f. Submit gumb
4. Testirati slanje podataka i promotriti zahtjev i odgovor servera
5. U akciji **SubmitQuery** pročitati vrijednosti poslane s forme (preko parametra akcije koji je klase **IFormCollection**), te sastaviti opisnu rečenicu koju je potrebno proslijediti kao odgovor nakon slanja podataka. Rečenica treba biti oblika: „Poštovani IME PREZIME (EMAIL) zaprimili smo vašu poruku te će vam se netko ubrzo javiti. Sadržaj vaše poruke je: [TIP PORUKE] SADRZAJ PORUKE. Također, (obavijestit ćemo vas/nećemo vas obavijestiti) o daljnjim promjenama preko newslettera.“

Twitter Bootstrap

Ili popularnije, samo **bootstrap**, je skup javascript i CSS biblioteka koje omogućavaju lakši razvoj web aplikacija, pri tome inicijalno postavljajući dobar dizajn i pružajući niz funkcionalnosti koje su potrebne u većini slučajeva prilikom izrade web stranica i web aplikacija.

S obzirom na veliku količinu postojeće dokumentacije[4], ovdje će biti samo istaknuto nekoliko glavnih principa izrade aplikacija temeljenih na bootstrap bibliotekama.

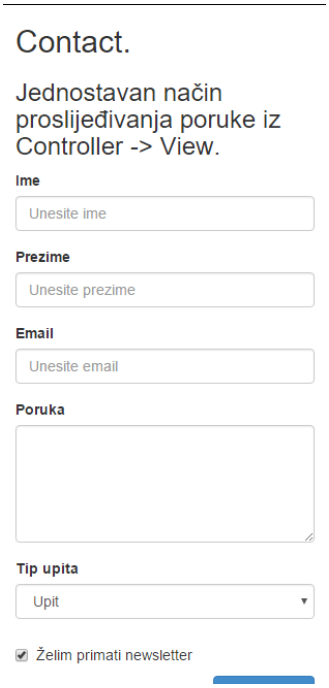
Grid system[5]

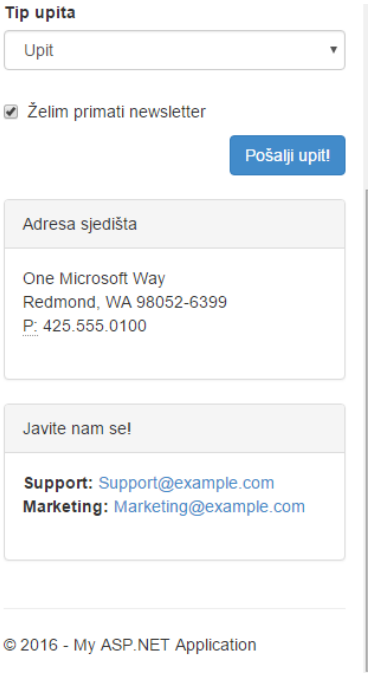
Ideja je da se cijelo korisničko sučelje podijeli u mrežu manjih dijelova, koji se dalje dijele opet u mrežu, itd. Bitno svojstvo ovakve mreže je da je (manje-više) automatski prilagodljivo veličini ekrana (mobiteli, tableti, široki ekrani) te na taj način drastično poboljšava iskustvo korisnika na stranici.

Zadatak 4.3

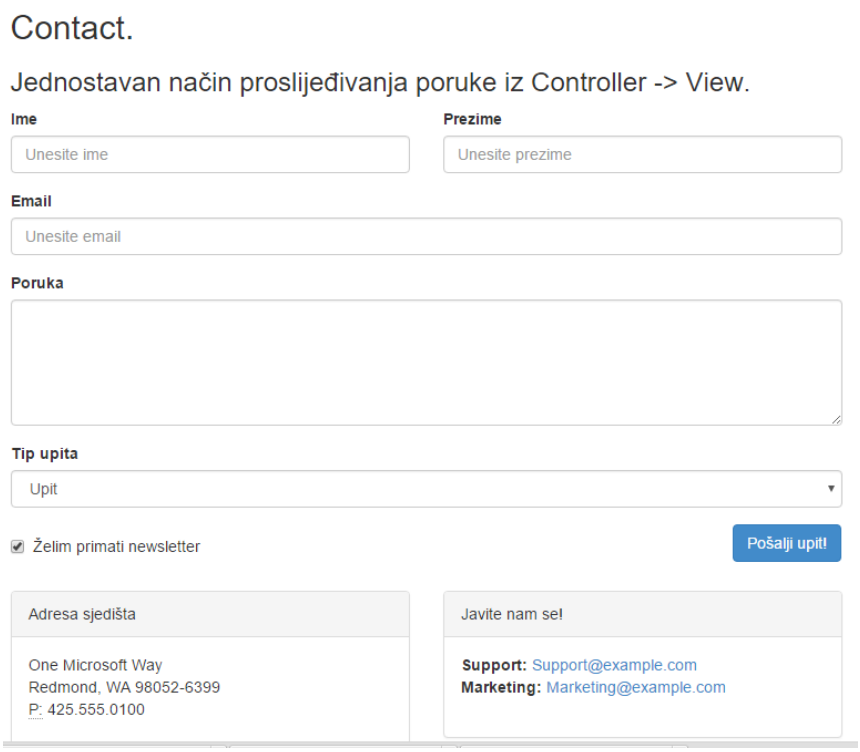
Implementirati kontakt formu koristeći grid system.

- Isprobati i uočiti razlike između –**xs**, -**sm**, -**md**, -**lg** sufixa na primjeru smanjenja browser prozora (skaliranja na manje rezolucije)
- Proučiti bootstrap forms i implementirati ispravni HTML i CSS kako bi forma izgledala kao na slici na **idućoj stranici,** te kada se rezolucija smanji (suziti browser), onda treba izgledati ovako (lijeva slika je scroll na vrhu, desna scroll na dnu):





[4] Kompletna dokumentacija s nizom primjera može se pronaći ovdje: http://getbootstrap.com/
[5] Detalji o grid system konceptu: https://getbootstrap.com/docs/4.1/layout/grid/

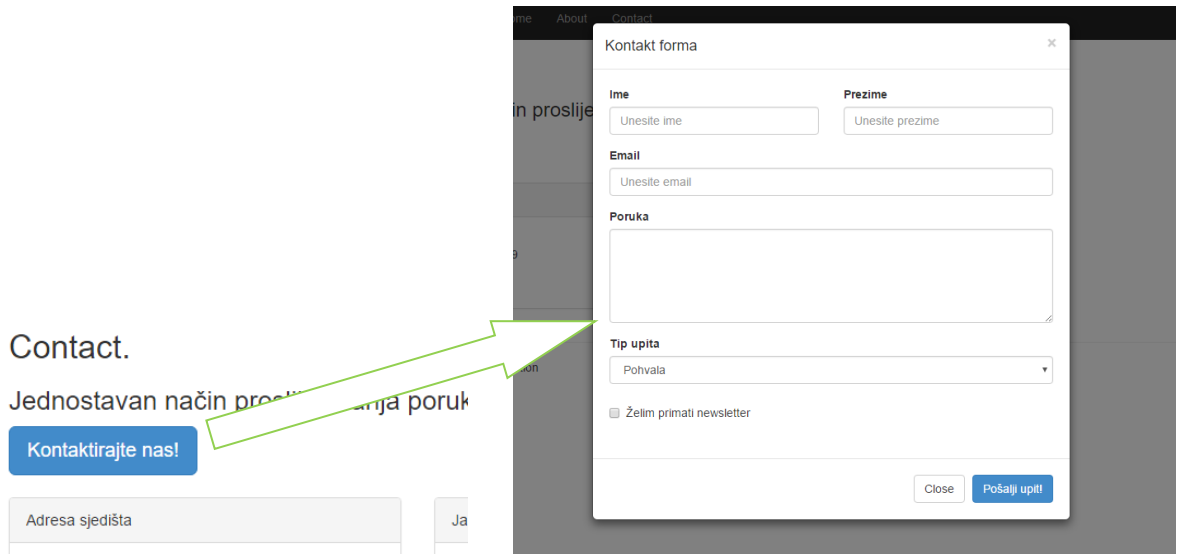


Modal[6]

Vrlo često korištena komponenta za prikaz informacija u popup prozoru. Uz mogućnost otvaranja popup prozora samo korištenjem HTML-a, dodatno omogućava i proširenja korištenjem javascript funkcija.

Zadatak 4.4

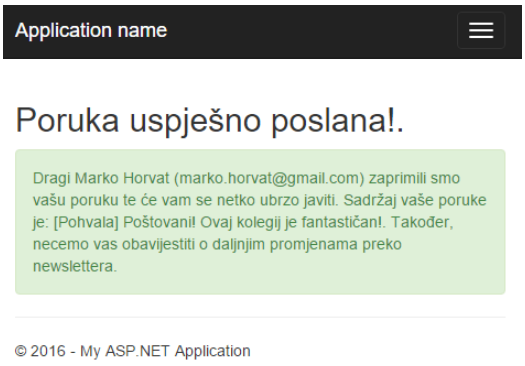
Modificirati Contact.cshtml stranicu na način da se kontakt forma prikazuje u bootstrap popup prozoru. Gumb za otvaranje modalnog prozora mora imati id = „modal-toggle-button“



[6] Modal: https://getbootstrap.com/docs/4.1/components/modal/

Zadatak 4.5

Iskoristiti alert prozor za prikaz poruke na zadnjem ekranu:



Zadatak 4.6

Iskoristiti „breadcrumbs“ bootstrap komponentu kako bi prikazali korisniku gdje se trenutno nalazi: