.NET Okruženje

Vježba 1: Razvojno okruženje

Sadržaj

Razvojno okruženje

S obzirom da ASP.NET Core nije vezan za operativni sustav, moguće je koristiti i Linux za razvojno okruženje, ali u tom slučaju nije dostupan Visual Studio alat, te se ipak preporuča Win10 kao razvojno okruženje.

Instalacija potrebne programske podrške

Navedeni software je najoptimalnije instalirati navedenim redoslijedom.

MSSQLExpress with tools 2014+

MSSQL je Microsoft baza podataka koju je najbolje koristiti u .NET razvojnom okruženju - pruža najveću razinu kompatibilnosti. Uz MSSQL bazu dolazi i MSSQL management studio express kojim se može mijenjati baza podataka (dodavanje tablica, izvršavanje upita, itd..). Zajednička instalacija za verziju 2017 se nalazi na ovom linku:

https://www.microsoft.com/en-us/sql-server/sql-server-downloads

Međutim, ova predavanja će također biti moguće pratiti i koristeći tzv. local db bazu podataka ukoliko iz nekog razloga nije moguće instalirati SQL express.

Važna napomena: potrebno je odabrati instalaciju "with tools", kako bi se automatski instalirao i MSSQL management studio.

Za one upoznate s Docker virtualizacijom, moguće je koristiti i MSSQL Image: https://hub.docker.com/_/microsoft-mssql-server ili bilo koju drugu bazu podataka koju podržava EF Core (PgSQL, MySQL, itd..)

Visual studio 2019

Za uspješno završavanje ovog kolegija, kao i za potencijalni daljnji razvoj web aplikacija u ASP.NET MVC tehnologiji dovoljno je instalirati community edition verziju VS2019 alata. S obzirom da TVZ ima ugovor s MSDNAA organizacijom, moguće je besplatno instalirati i Professional verziju Visual Studio alata. Na ovom linku je moguće skinuti besplatnu community verziju koja sadrži sve potrebne funkcionalnosti VS alata:

https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx

Što se tiče community verzije u odnosu na professional verziju – nema puno razlika. Glavne razlike su u licenci za korištenje, koja je vrlo fleksibilna:

- Ograničenje na veličinu firme (do 5 zaposlenika)
- Ograničenje na godišnji prihod firme (do 1,000,000$)

Najbolje je instalirati alat s osnovnim (default) postavkama, no može se izostaviti instalacija primjerice C++ biblioteka ukoliko je potrebno, a može se i dodati (po želji) dodatak za razvoj mobilnih aplikacija Xamarin.

Također, potrebno je instalirati i paziti da se koristi **ASP.NET Core 5.0** radni okvir.

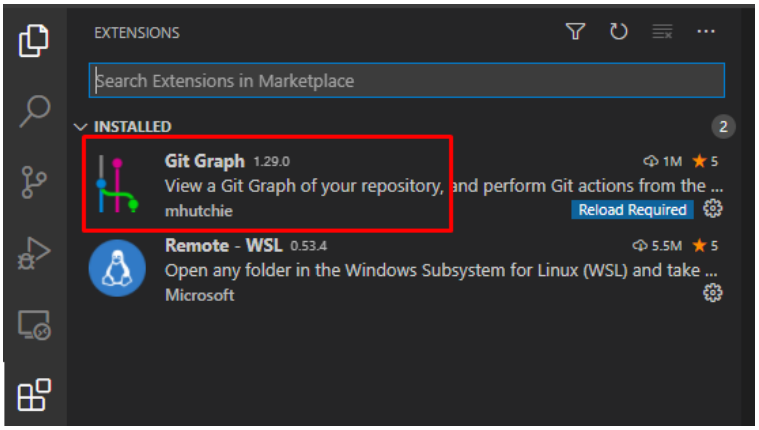
Visual Studio Code / git

Git je moderni source control alat, koji omogućava kolaboraciju i sudjelovanje na istom projektu više članova nekog tima. Visual Studio Code je alat koji nudi niz razvojnih i kolaboracijskih opcija. Mi ćemo ga na ovom kolegiju koristiti za vizualni pregled git repozitorija:

https://code.visualstudio.com/download

Sami Visual Studio ima dodatak za korištenje git alata za verzioniranje koda, ali ćemo preferirati VS Code zbog bolje vizualizacije.

Dodatno, potrebno je instalirati ekstenziju „**Git Graph**“:



Potrebno je i instalirati git konzolnu podršku ukoliko već nije instalirana. Provjeriti se može na način da se pokuša izvesti „git“ naredba u Powershell ili cmd sučelju.

Google chrome / Mozilla firefox

Pošto je ASP.NET MVC tehnologija uvelike zasnovana na HTML-u i javascript-u, potrebno je na odgovarajući način omogućiti debugiranje javascript koda, te modifikaciju HTML/CSS-a. Za to je najpogodnije koristiti jedan od dva alata:

- Google chrome developer tools (na bilo kojoj HTML stranici moguće je na željeni HTML element desnim klikom otvoriti developer alat – desni klik + inspect element ili naredbom F12). U svim opisima/screenshotovima će se koristiti ovaj alat.
- Mozilla firefox također ima developer konzolu, koju je moguće aktivirati na sličan način kao i u google chrome pregledniku

Upoznavanje radne okoline

VS Code i source control

Cijela ideja source control alata je bazirana na sljedećem: kod se drži na serveru, a pri razvoju se lokalno na računalima pojedinih članova tima razvija i mijenja kod, koji se u željenim trenucima sinkronizira sa serverom. Primitivan oblik source control alata bi mogao biti dropbox, no on je pogodan samo za razvoj jednog člana tima na više računala. Glavni nedostatak je mogućnost kontrole sinkronizacije i spajanja dokumenata, što „pravi“ source control alat rješava. Pri osnovnom radu sa git source control alatom, razlikujemo sljedeće pojmove:

- **Clone** – inicijalno dohvaćanje koda sa servera na vlastito računalo. Potrebno je posjetiti stranicu projekta (https://github.com/icesar-tvz/net-ok-20-21) i napraviti „git clone git@github.com:icesar-tvz/net-ok-20-21.git“. S obzirom da se radi o javnom repozitoriju, nije potrebno dodavati SSH ključeve te u sklopu ovog kolegija nećemo ulaziti u detaljnjiji opis tog procesa. U navedeni repozitorij će biti dodavan inicijalni source kod koji će se modificirati i koristiti na vježbama.
- **Commit** – naredba kojom se napravljene izmjene spremaju na lokalni git server, no nisu poslane na „pravi“ server i ostali članovi tima još ne vide taj kod. Prednost je da se izmjene koje su samo commitane mogu opovrgnuti prije nego se pošalju na online server
- **Fetch** – promjene koje su online na sreveru se dohvaćaju u lokalni git repository, ali još nisu aplicirane u lokalnu kopiju
- **Merge** – spaja promjene s jedne grane (eng. branch) na drugu ili novonastale promjene s online repozitorija na lokalnu radnu kopiju. Prilikom spajanja postoji mogućnost konflikta koji se moraju razriješiti u tom slučaju.
- **Pull** – naredba kojom se dohvaćaju zadnje izmjene sa servera i kod na lokalnom računalu se sinkronizira sa stanjem na serveru, s tim da se spajanje (merge) radi automatski (naravno, konflikti se naknadno rješavaju)

Sinkronizacija koda za vježbu

Prije početka svake vježbe je potrebno na računalu dohvatiti pripremljen kod za vježbu sa servera. Taj kod se (najčešće, pogotovo od vježbe 4) nastavlja na prethodnu vježbu i moguće će ga biti skinuti i nešto prije samog labosa (min 1-2 dana). Sam proces izgleda ovako:

1. Napraviti clone repozitorija
2. Prebaciti se u ispravnu granu (branch) koja odgovara traženoj vježbi
3. Raditi izmjene na toj grani.
    a. U iznimnim situacijama može biti zatraženo da se na vježbi napravi posebna grana (svaki student zasebno) i pošalju podaci na tu granu, no to će biti detaljno objašnjeno na samoj vježbi u tom slučaju
    b. Po želji, moguće je lokalno spremati promjene (sjetimo se, git uvijek ima lokalni repozitorij)

Mjesto na koje se uploada source kod nazivamo repozitorij. Git alat funkcionira na način da kod promjene aktivne grane mijenja lokalnu radnu kopiju, po čemu se razlikuje od primjerice SVN alata koji svaku granu drži u odvojenoj mapi. Što se tiče terminologije, bitno je istaknuti i sljedeće pojmove:

- **Remote / Origin** – u git svijetu se pojam „remote“ koristi kada se referiramo na online server gdje je repozitorij (github, primjerice). Origin je konkretna instanca tog servera.
- **Branch** - koncept koji omogućava paralelni razvoj aplikacije u nekoliko odvojenih smjerova koji se kasnije mogu spojiti po potrebi. Kod koji se razvija na nekom od branch-eva se kasnije može pripojiti glavnoj grani (master branch, ili trunk). Kad govorimo o granama (branch) razlikujemo sljedećih nekoliko scenarija[1][2].

[1] Dobro objašnjenje grananja uz vizualni prikaz može se naći primjerice ovdje: http://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

  - **Master/main branch** – u jednom trenutku u vremenu određeni source kod se postavlja u produkcijsko okruženje, dok se razvoj nastavlja dalje. Problem nastaje ako se pojavi bug u produkcijskom okruženju kojeg je hitno ispraviti, a razvoj se nastavio i postoje dijelovi koda koji nisu završeni/testirani. Kako bi se izbjegao taj problem, sav razvoj se radi na ostalim granama, dok je master branch rezerviran za „netaknuti" kod koji je u svakom trenutku spreman za postavljanje u produkcijsko okruženje ili ispravljanje hitnih bugova (hotfix).
  - **Develop branch** – redovni (normalni) razvoj se u pravilu događa na develop grani, te se u trenutku puštanja u produkciju, develop grana spoji s master granom i radi se deploy s master grane
  - **Feature branch** – ukoliko dio tima razvija feature koji uzima mnogo vremena, otvara se posebni feature branch na kojem će se kod nesmetano razvijati dok god je potrebno, ne blokirajući pri tome razvoj na develop grani
  - **Hotfix branch** – ukoliko je s master grane napravljen publish na produkcijsko okruženje, i tijekom daljnjeg razvoja se dogodi nepredviđena situacija i potrebno je brzo popraviti neki hitan bug, tada se za popravak hitnog buga kreira tzv. hotfix branch, koji se kasnije spaja na master po završetku ispravka pogreške, i master opet ide u „deploy"
- **Tag** – svojevrstan snapshot koda u željenom trenutku – primjerice, ukoliko razvijamo aplikaciju koja se postavlja na nekoliko mjesta, možda neće svi klijenti ili sva mjesta imati istu verziju aplikacije, te je tada u nekim situacijama dobro kreirati „tag" koji primjerice označava stabilnu release verziju aplikacije (recimo v1.0, v2.0, …)

Kao što je ranije spomenuto, git se razlikuje od, primjerice, SVN alata utoliko što je projekt na jedinstvenom mjestu u željenoj mapi (ne postoje podmape za grane) – te se promjenom aktivne grane git pobrine da u željenoj mapi bude točno onakvo stanje kakvo smo zatekli u tom trenutku.

Visual studio alat

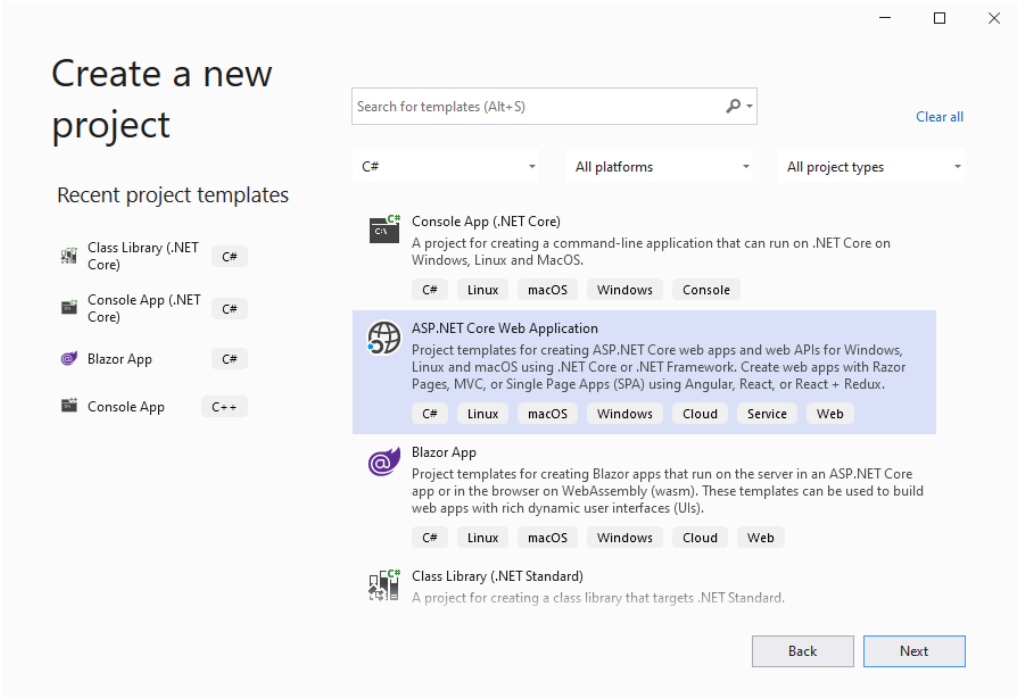
U ovom dijelu će biti objašnjeno i kako kreirati ASP.NET Core Web aplikaciju koju će se koristiti za razvoj završnog projekta, te će biti objašnjeni i prvi koraci po pitanju sinkronizacije sa surce control serverom.

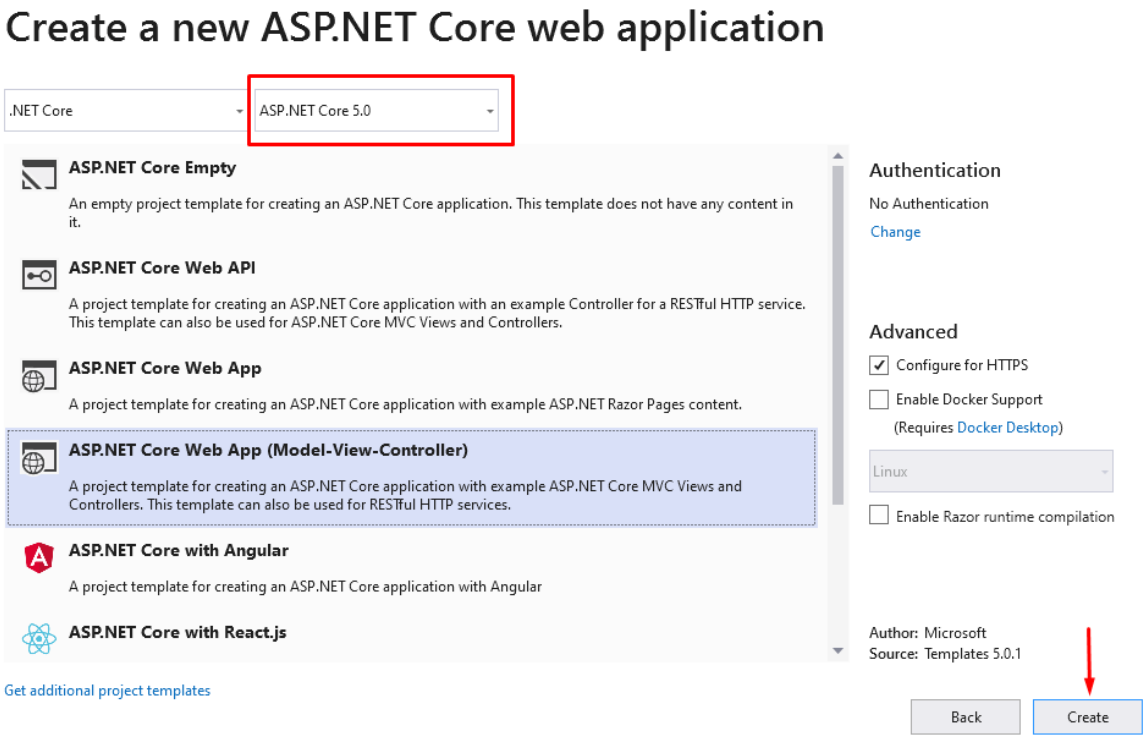
Kreiranje novog projekta

Visual studio je alat za .NET programske jezike (C#, VB.NET), koji omogućava razvoj raznih tipova projekata. Za kreiranje novog projekta (seminara) potrebno je na početnom ekranu odabrati opciju new project. Na idućem ekranu potrebno je odabrati projekt tipa ASP.NET Core Web application, odabrati Web Application (Model-View-Controller) aplikaciju. Lokaciju projekta potrebno je postaviti u željenu mapu na računalu. Kao naziv solutiona potrebno je postaviti ime koje ste odabrali kao temu vašeg projekta: primjerice, QuizManager[3]. Naziv projekta potrebno je postaviti kao QuizManager.Web. Naime, u jednom solutionu može biti više raznih projekata, što će biti obrađeno u kasnijim predavanjima.

[2] Jedan primjer strategije kreiranja grana se može pogledati u ovom video uratku: https://www.youtube.com/watch?v=to6tIdy5rNc

[3] U vašem konkretnom slučaju potrebno je nazvati solution ispravnim imenom. Primjerice, ako se izrađuje aplikacija za Parking, tada ju nazvati ParkingManager ili slično.





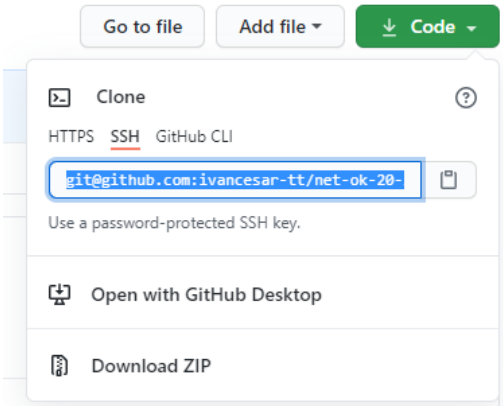
Otvaranje postojećeg projekta

Na vježbama će se koristiti već kreirani projekt s već napisanim source kodom prilagođenim zadacima za vježbu. Iz tog razloga, potrebno je na početku svake vježbe obaviti sinkronizaciju sa serverom i dohvatiti potrebni projekt te ga otvoriti u Visual Studio alatu. Samim time dolazimo i do prvog zadatka.

Zadatak 1.1

Spojiti se na repozitorij https://github.com/icesar-tvz/net-ok-20-21, te dohvatiti sadržaj repozitorija te otvoriti projekt u Visual Studio alatu.

1. U konzoli pokrenuti git clone i c/p putanju:



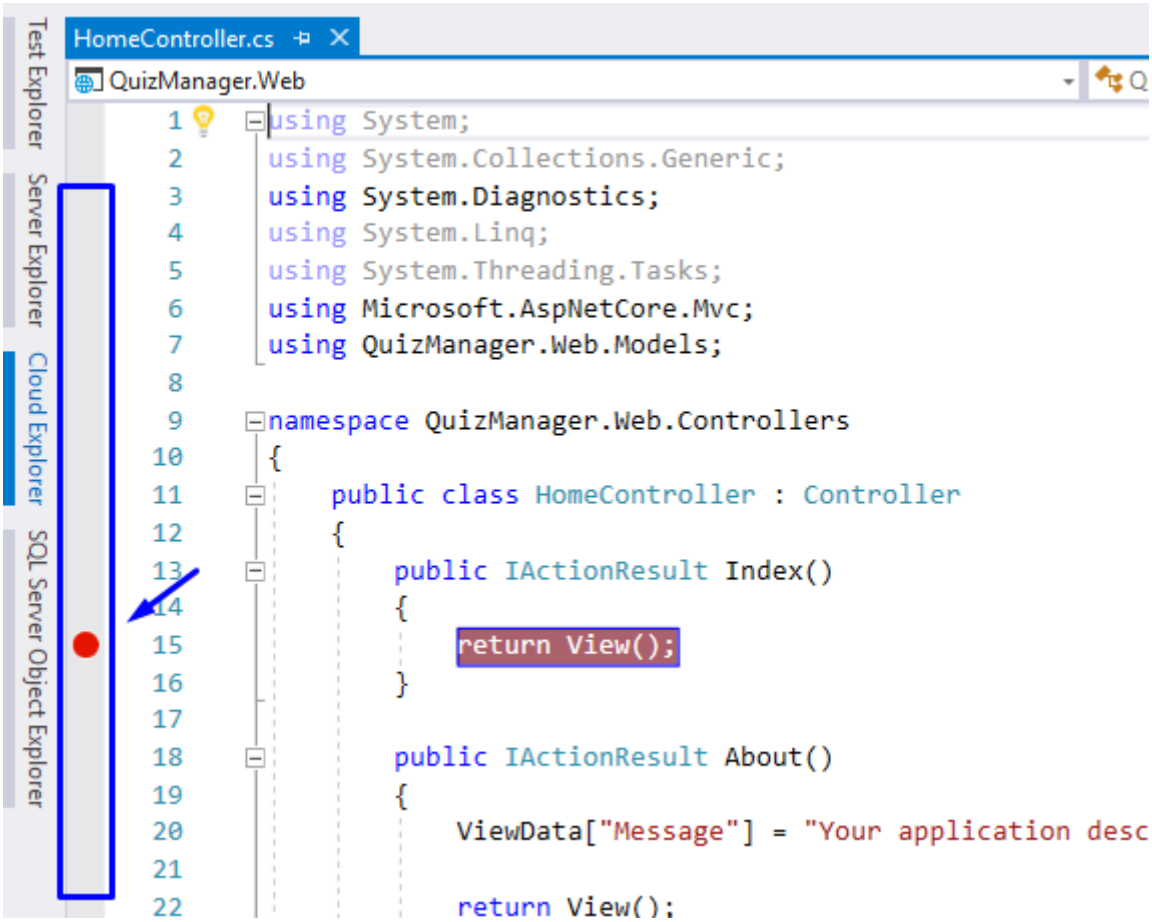
2. Po uspješnom dohvatu, potrebno je odabrati aktivnu granu „vjezba-01/setup“. Aktivaciju grane je moguće napraviti u VS Code alatu ili konzoli naredbom „git checkout vjezba-01/setup“.
3. Pronaći traženu mapu na disku, te otvoriti datoteku sa ekstenzijom „.sln“. Visual Studio bi automatski trebao prepoznati i otvoriti projekt

Prevođenje i pokretanje projekta

Kako bi provjerili prevodi li se projekt uspješno, potrebno je odabrati Build -> Build solution. Ukoliko postoje greške prilikom prevođenja, potrebno ih je razriješiti prije pokretanja projekta. Nakon kreiranja projekta, projekt se sigurno uspješno prevodi. Aplikacija se može pokrenuti tako da odaberemo Debug -> Start debugging ili kraće, F5.

Nakon pokretanja aplikacije, VS diže razvojni web server koji imitira pravi IIS server i omogućava debugiranje i razvoj koda. Prilikom prevođenja aplikacije, nastaju .dll datoteke koje sadržavaju MSIL prijevod izvornog koda. Te .dll datoteke uzima i IIS server i razvojni web server, te pomoću njih kreira privremenu web adresu na kojoj možemo pristupiti našoj aplikaciji. Našoj aplikaciji se dodjeljuje privremeni port (primjerice, 53336 ili 3242 i sl.). Iz tog razloga, promjene koje se naprave u C# kodu se ne mogu vidjeti prije ponovnog pokretanja aplikacije (potrebno je izaći iz debug mode-a odabirom Debug -> Stop debugging). Također, možemo uočiti u donjem desnom dijelu ekrana balončić koji sadrži obavijest na kojem portu je pokrenuta aplikacija, te ukoliko je potrebno može se i tamo nasilno prekinuti (ukoliko se dogodi veća pogreška i sl.)

Debugiranje je proces u kojem možemo dobiti detaljni uvid u stanje varijabli tokom izvođenja koda. Izvršavanje koda se zaustavlja u točkama prijeloma (breakpoint). Kako bi na odgovarajućem mjestu u kodu zaustavili izvršavanje i proučili vrijednosti varijabli i stanje memorije, potrebno je staviti točku prijeloma na odgovarajuće mjesto – primjerice na neka od mjesta kao na ovoj slici:



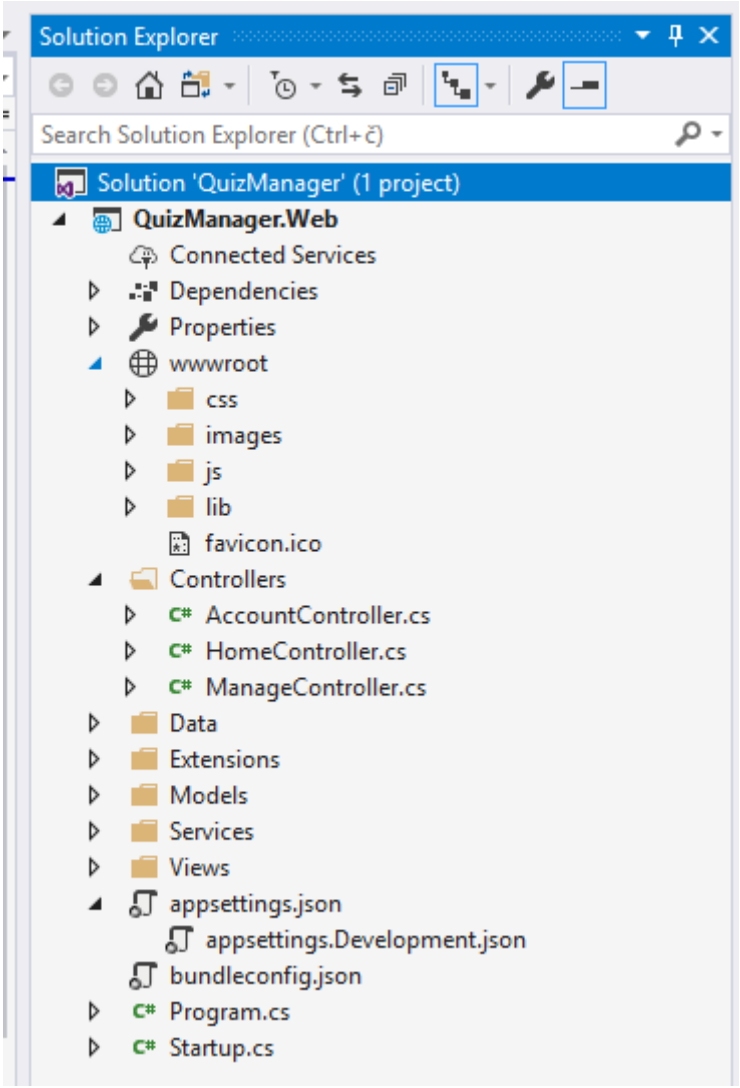
Solution explorer

S desne strane ekrana u razvojnom okruženju možemo uočiti solution explorer, koji služi za navigaciju po datotekama i projektima u našem riješenju. Postoji nekoliko tipova datoteka koje se pojavljuju u ASP.NET MVC projektima:

- Ekstenzija **.cs** – označava datoteku s C# kodom (klasa ili više klasa)
- Ekstenzija **.cshtml** – označava datoteku u kojoj se nalazi HTML kod koji modificiramo za prikaz. To su tzv. view datoteke.
- Ekstenzija **.js** – označava datoteke s javascript kodom
- Datoteka **appsettings.json** – konfiguracijska datoteka
- Properties/launchsettings.json – konfiguracija pokretanja aplikacije u VS alatu
- Ostale ekstenzije – moguće je dodati i razne slike (niz ekstenzija), dokumente po volji, itd...

Datoteka wwwroot

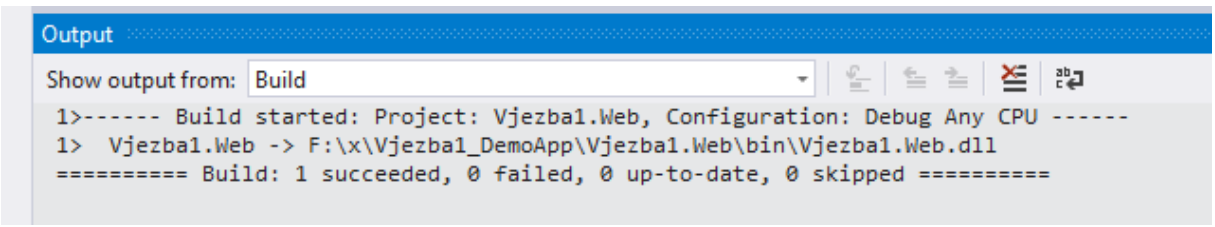
Možemo uočiti jednu specifičnu datoteku – wwwroot. Unutar te datoteke se nalaze u pravilu statički elementi – oni koji se ne mijenjaju u ovisnosti u podacima u aplikaciji. To su najčešće datoteke za stilove, skripte (JS), slike i ikone te po potrebi datoteke koje ćemo dopustiti korisniku da skine (recimp upute za korištenje aplikacije).



Raspored prozora se može promijeniti ovisno nalazimo li se u fazi razvoja i pisanja koda, ili u fazi pronalaska pogrešaka (debugiranja). U nastavku su objašnjeni dijelovi korisničkog sučelja Visual Studio 2019 alata u ovisnosti o trenutnom modu u kojem alat radi.

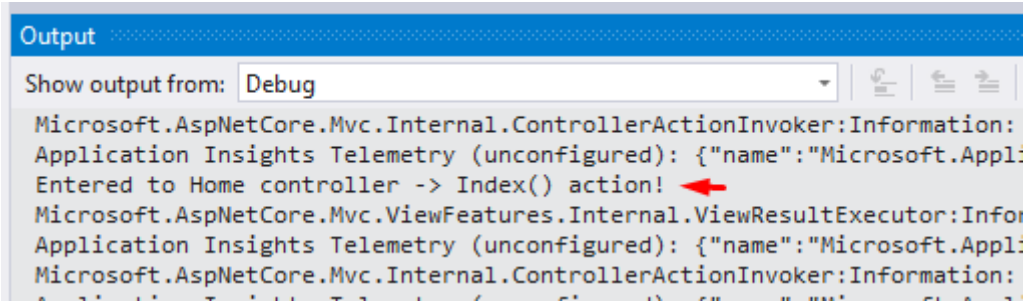
Ostali prozori u VS2019 alatu – dev mode

U donjem dijelu alata automatski su vidljiva dva prozora: output i error list. **Output** sadrži ispise prilikom prevođenja (1. slika) ili dijagnostičke ispise prilikom izvođenja aplikacije (2. slika).

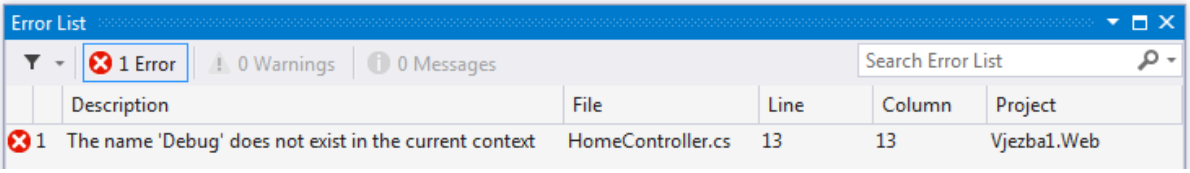


HomeController.cs

```
namespace QuizManager.Web.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            System.Diagnostics.Debug
                    .WriteLine("Entered to Home controller -> Index() action!");
            return View();
        }
...
```

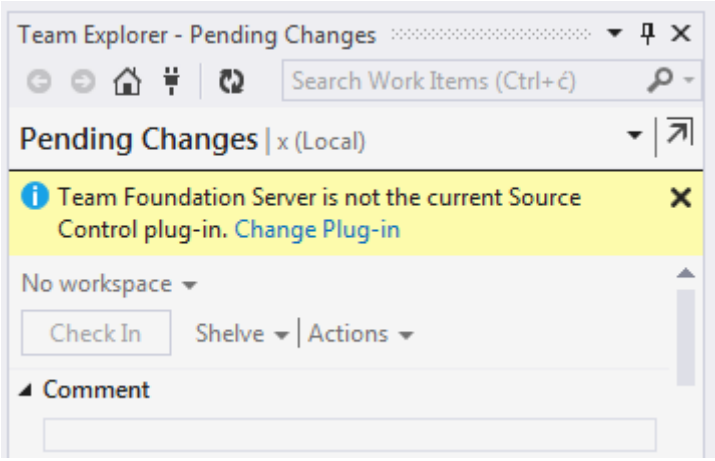


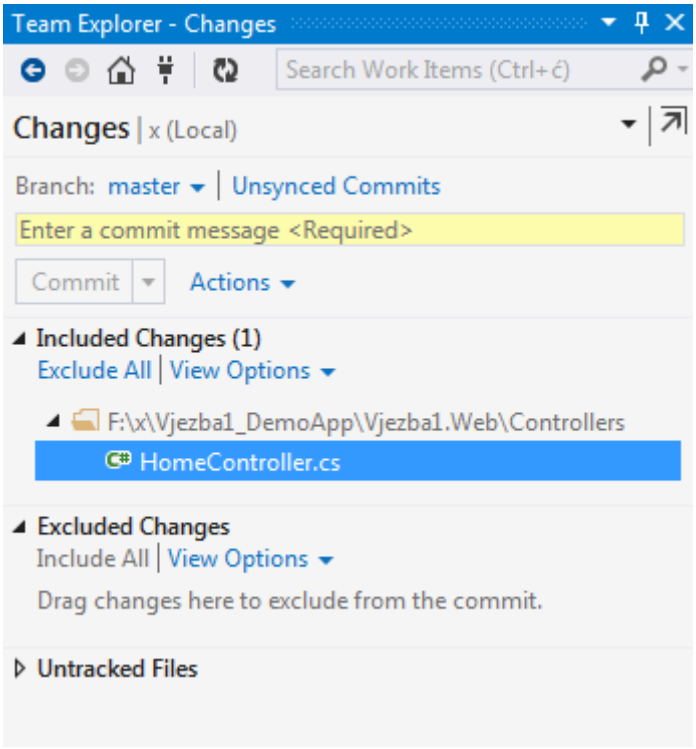
Drugi bitan ekran prilikom razvoja je ekran za pregled pogrešaka (**error list**). Nakon što pokušamo prevesti kod (Build), moguće je da postoje neke pogreške:



Pogreške prilikom prevođenja je lakše ispraviti, no za neiskusne programere i to može biti prilično zahtjevno dok se kroz iskustvo ne stekne dojam o najčešćim pogreškama.

S desne strane se nalazi još jedan koristan prozor – **team explorer**. S obzirom da koristimo git, VS2019 ima automatsku podršku za git source control sustav, te je moguće koristiti prilagođeni pregled trenutnih promjena u kodu:

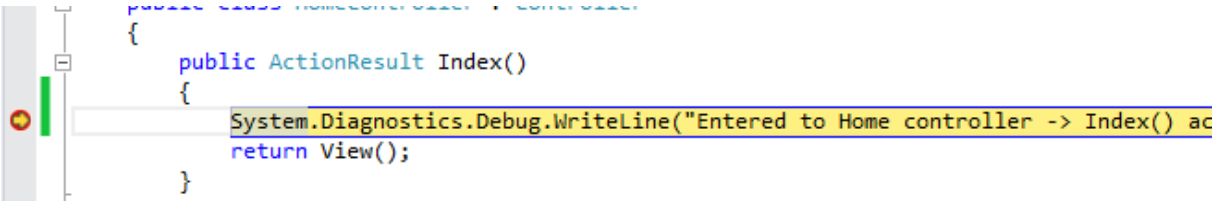




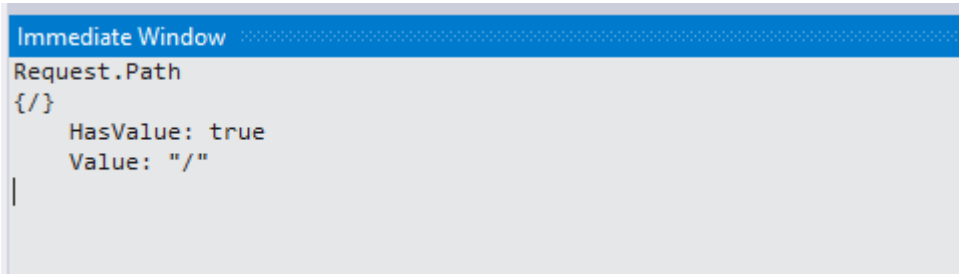
Međutim, preporuča se korištenje SourceTree alata jer omogućava više opcija i nudi adekvatniji prikaz promjena i stanja u repozitoriju.

Ostali prozori u VS2019 alatu – debug mode

Kao i prilikom razvoja, **output** prozor je prisutan i u njega se ispisuju eventualne dijagnostičke poruke. Uz njega, postoji **immediate window** koji nam omogućuje da kada stavimo točku prijeloma izvršimo kod po želji.



U nastavku je prikazan immediate window nakon što kao naredbu upišemo „Request.Path“ unutar controllera. To samo po sebi nije naredba, ali Request.Path je objekt, i kada predamo objekt kao parametar u ovom prozoru, ispisuju se sva svojstva tog objekta:



Možemo, naravno, izvoditi i konkretne funkcije poput ove (koja se primjerice ne izvršava dobro, te možemo vidjeti detalje iznimke):

```
HttpContext.Session.Clear()
'HttpContext.Session.Clear()' threw an exception of type 'System.InvalidOperationException'
    Data: {System.Collections.ListDictionaryInternal}
    HResult: -2146233079
    HelpLink: null
    InnerException: null
    Message: "Session has not been configured for this application or request."
    Source: "Microsoft.AspNetCore.Http"
    StackTrace: "   at Microsoft.AspNetCore.Http.DefaultHttpContext.get_Session()"
    TargetSite: {Microsoft.AspNetCore.Http.ISession get_Session()}
```

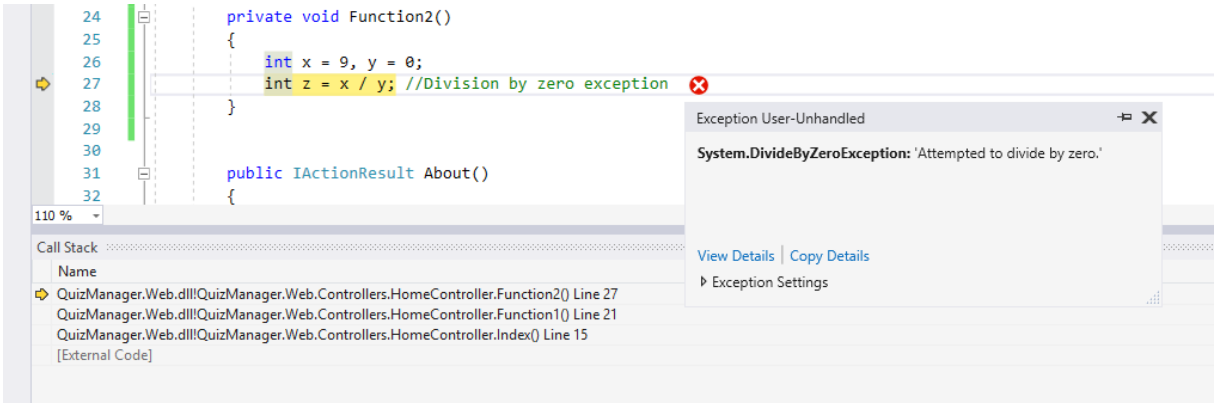
Još jedan bitan prozor je **call stack** prozor. On je posebno važan kada se dogodi iznimka, kako bi mogli jednostavno pratiti u kojoj točno funkciji se dogodila iznimka – naime, čak i u jednostavnim aplikacijama, postoji nekoliko razina poziva funkcija (jedna funkcija zove drugu, druga treću, treća četvrtu, …) i kada se dogodi iznimka, može se dogoditi na bilo kojoj od razina. U tim trenucima korisno je znati točno koja je funkcija zvala koju, te imati pristup varijablama u pojedinoj funkciji:

HomeController.cs

```
namespace Vjezba1.Web.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            Function1();
            return View();
        }

        private void Function1()
        {
            Function2();
        }

        private void Function2()
        {
            int x = 9, y = 0;
            int z = x / y; //Division by zero exception
        }
```



Duplim klikom na pojedinu liniju u Call stack prozoru, možemo doći upravo na tu liniju.

Ostali prozori

Do ostalih prozora moguće je doći preko izbornika **view** ili **view -> other windows**.