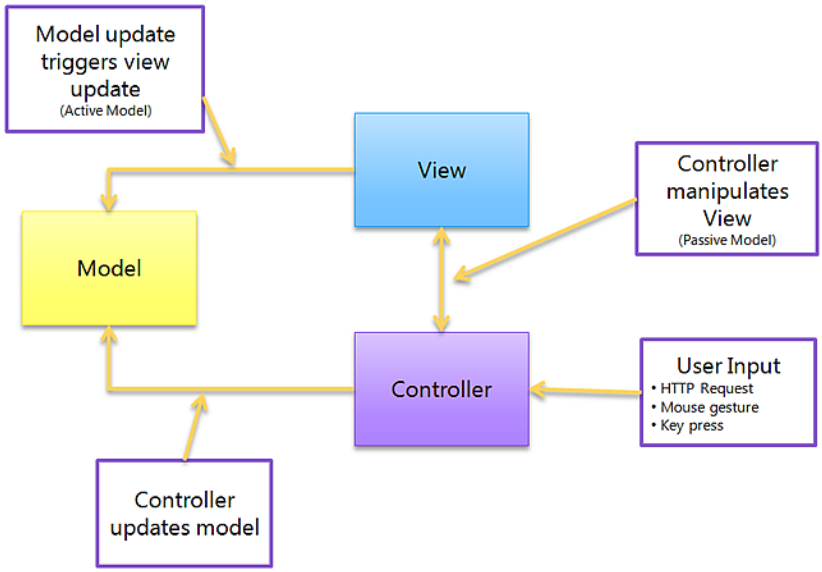
# ASP.NET

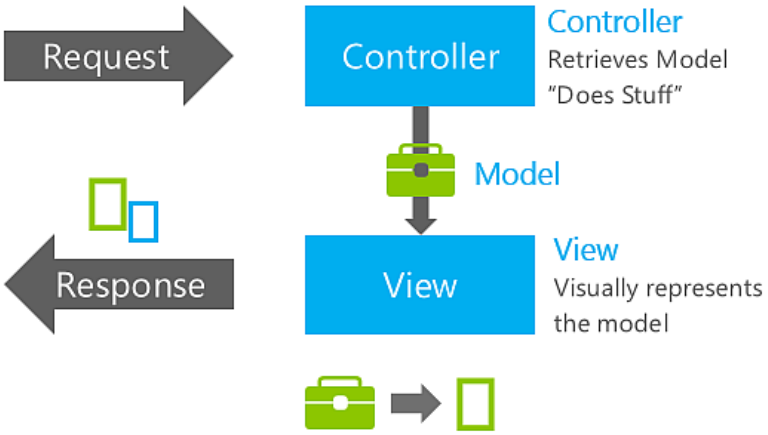
## Uvod

- ASP.NET MVC – okruženje za izradu web aplikacija primjenom generalnog Model-View-Controller predloška na ASP.NET platformu.
- Softversko okružje
    - univerzalna, ponovno uporabljiva okolina koja omogućava određene funkcionalnosti kao dio veće softverske platforme kako bi olakšala razvoj aplikacija. Može uključivati programe za podršku, kompajlere, biblioteke, API-je,...
    - platforma/osnova za razvoj aplikacija.
    - okružje služi kao temelj za razvoj aplikacije, a API omogućava pristup elementima okružja.
- ASP.NET **tipovi aplikacija**/načini razvoja:
    - **Web forms** – koristi web forme, gotove kontrole od kojih se slaže aplikacija. Temeljna kontrola je klasa Page. Kôd se odvija u *code-behind* datotekama.
    - **ASP.NET MVC** – temelji se na MVC predlošku. Koristi Razor sintaksu za kombiniranje HTML i C# kôda. Osnovni element prikaza je Page ugrađen u MVC predložak. Kôd se odvija u *code-behind* datotekama.
    - **Web API** – omogućava kreiranje RESTful servisa. Koristi se za izradu aplikacije namijenjene nekoj drugoj aplikaciji.
- Single Page App – s web servera se dohvaća samo jedna HTML stranica unutar koje se izvođenjem JavaScript kôda dohvaćaju informacije sa servera.
- Softverski **uzorak (predložak)** – općenito rješenje koje se koristi za slične probleme prilikom dizajniranja aplikacije. To je opis ili šablona po kojoj se rješava određeni dizajnerski problem u aplikaciji.
    - MVC – jedan od prvih predložaka za odvajanje prezentacije sadržaja aplikacije korisniku od ostatka aplikacije.




    - ASP.NET MVC predložak
        - HTTP zahtjevi se prosljeđuju kontrolerima koji ih obrađuju i izvode CRUD operacije nad podacima u modelu.

- Podaci se kombiniraju s pogledom i vraćaju u HTTP odgovoru.
- Model postoji samo na strani servera (kod novijih tehnologija dio se može prebaciti na klijenta)




- **Struktura direktorija** ASP.NET MVC aplikacije
  - Controllers – sadrži klase kontrolera
  - Models – sadrži podatkovne klase
  - Views – sadrži datoteke zadužene za prikaz (npr. HTML)
  - Scripts – sadrži JavaScript datoteke
  - Content – sadrži CSS i druge sadržaje osim skripta i slika
  - App_Data – sadrži podatkovne datoteke
  - App_Start – sadrži konfiguracijski kôd za Routing, Bundling, Web API, ...
- **Konvencija nad konfiguracijom** – paradigma kojom se smanjuje opseg odluka koje developeri trebaju donijeti tijekom razvoja aplikacije. Povećava se jednostavnost razvoja, bez smanjenja fleksibilnosti.
  - U ASP.NET MVC-u ovaj koncept je proveden kroz tri osnovna direktorija – Controllers, Models, Views.
  - Neka imena direktorija su podrazumijevana konvencijom pa ih nije potrebno definirati.
- Osnovne **ASP.NET MVC konvencije**
  - Svaka klasa kontrolera završava s riječi Controller.
  - Postoji samo jedan Views direktorij za sve prikaze aplikacije.
  - Prikazi koje kontroleri koriste smještaju se u poddirektorij Views direktorija i imenuju se prema imenu kontrolera ali bez sufiksa Controller.
  - Svi dijelovi sučelja koji su ponovno upotrebljivi smještaju se u poddirektorij Shared direktorija Viwes.

## Rutiranje

- Rutiranje – usmjeravanje klijentskih HTTP zahtjeva sa zahtjevnog URL-a na krajnju točku aplikacije.
- Klasični princip usmjeravanja – URL koji predstavlja globalni identifikator traženog resursa mapira se u lokalnu datoteku na disku.
  - Nedostaci:
    - Korisnik ne može razumjeti što se ustvari dohvaća.
    - Web tražilice ne mogu dobro povezati URI s pojmovima pretraživanja.

        - Nije estetski.
    - Smjernice dobro dizajniranih URL-ova:
        - Domena jednostavna za pamćenje
        - Kratki URL-ovi
        - Jednostavni za tipkanje
        - Odražavaju strukturu sitea
        - Mogu se jednostavno „hakirati“ promjenom kraja URL-a
        - Trajni su (PURL – persistent URL) – ne mijenjaju se unatoč promjeni Web infrastrukture
- **Vanity URL** – razumljiv korisnicima i tražilicama.
- Rutiranje klijentskih web zahtjeva omogućuje korištenje vanity URL-ova.
- SEO (Search Engine Optimization) – postupak optimizacije web sitea za tražilice kako bi se poboljšala vidljivost web stranice u tražilicama.
- ASP.NET podržava rutiranje kroz zasebni API.
    - Omogućava programeru da definira URL predloške koji se onda mapiraju u rukovatelje zahtjeva.
    - Kod Web Forms aplikacija rukovatelj je .aspx datoteka, a kod MVC-a klasa Controller i neka od njezinih Action metoda.
    - Sve konfigurirane rute aplikacije spremaju se u tablicu **RouteTable** koju mehanizam za rutiranje koristi kako bi za dolazni HTTP zahtjev na osnovu URL-a pronašao odgovarajući handler.
    - Dopušteni predlošci za URL rutiranje i URL-ovi koji su u skladu s predlošcima (primjer):

| Route definition | Example of matching URL |
|---|---|
| `{controller}/{action}/{id}` | `/Products/show/beverages` |
| `{table}/Details.aspx` | `/Products/Details.aspx` |
| `blog/{action}/{entry}` | `/blog/show/123` |
| `{reporttype}/{year}/{month}/{day}` | `/sales/2008/1/5` |
| `{locale}/{action}` | `/US/show` |
| `{language}-{country}/{action}` | `/en-US/show` |

    - Svaka ASP.NET MVC aplikacija mora imati barem jednu rutu kako bi definirala kako podržava dolazni zahtjev.
    - **Konfiguracija ruta** radi se metodom RegisterRoutes. Rute treba registrirati nakon pokretanja aplikacije pozivom RegisterRoutes metode u Global.asax.cs datoteci.
    - **Global.asax.cs** – ASP.NET aplikacijski file – opcionalna datoteka koja sadrži kôd koji odgovara na događaje na razini cijele aplikacije koje podiže ili ASP.NET ili HttpModul IIS servera.
        - Nalazi se u root direktoriju aplikacije.
- **Ruta** je kombinacija URL predloška i informacije o handleru.
    - **ASP.NET MVC rute** trebaju sadržavati **kontroler i akciju** koja će se izvršiti.

```
routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);
```

  - Rute nije obavezno imenovati, ali je poželjno ako ih je potrebno referencirati.
  - Parametar URL je regularni izraz kojim se pronalazi odgovarajući request handler na osnovu HTTP URL-a.
  - Prema ovoj ruti svi dolazni zahtjevi tipa http://server/BILOŠTO/bilokakvoime prevodit će se u pozive metode *bilokakvoime* u klasi koja se zove *BILOŠTOcontroller* (ili *BILOŠTOController*) i koja nasljeđuje klasu *Controller*.
  - Ukoliko je URL u HTTP zahtjevu tipa http://server/BILOŠTO/bilokakvoime/nešto prevodit će se u pozive metode *bilokakvoime* u klasi koja se zove *BILOŠTOcontroller* kojoj će se predati parametar *nešto*.
  - Treći parametar *MapRoute* metode definira defaultni objekt koji se poziva kao request handler ako nisu navedeni ni jedan konroler ni akcija.
- **Inicijalizatori objekata** – služe za inicijalizaciju svojstava klase.
  - Definira se samo jedan konstruktor (ili se koristi defaultni), a prilikom kreiranja objekta koristi se incijalizator ta postavljanje svojstava na određene vrijednosti.
  - Preimjer je treći parametar metode *MapRoute*.
- **Inicijalizatori kolekcija** – omogućavaju inicijalizaciju kolekcija koje implementiraju IEnumerable sučelje ili imaju Add metodu.
- **Višestruki parametri ruta**:

```
routes.MapRoute(
        name: "multiparameter",
        url: "{controller}/{action}/{id}/{key}",
        defaults: new { controller = "students", action = "multiparametar", id =
        UrlParameter.Optional, key = UrlParameter.Optional }
);
```

- **Parametri ruta** u akciji se moraju zvati isto kao i u ruti.
  - Parametar „id“ ne treba imenovati u zahtjevu i ne treba biti dio query stringa, već se može navesti i kao dio puta.
- Podsustav rutiranja zadužen je i za **dinamičko kreiranje** URL-ova koji trebaju biti u skladu s rutama podržanim u aplikaciji.
  - Sve metode koje kreiraju URL-ove pozivaju metodu GetVirtualPath koja ima dvije verzije:
    - Prva verzija kao parametre prima objekt koji sadrži informacije o trenutnom HTTP zahtjevu i tablicu definiranih ruta iz kojih odabere odgoarajuću.
    - Druga metoda prima ime rute kao treći parametar. Imena u kolekciji ruta su jedinstvena i ako imenovana ruta ne odgovara po parametrima, metoda vraća null.

# Model

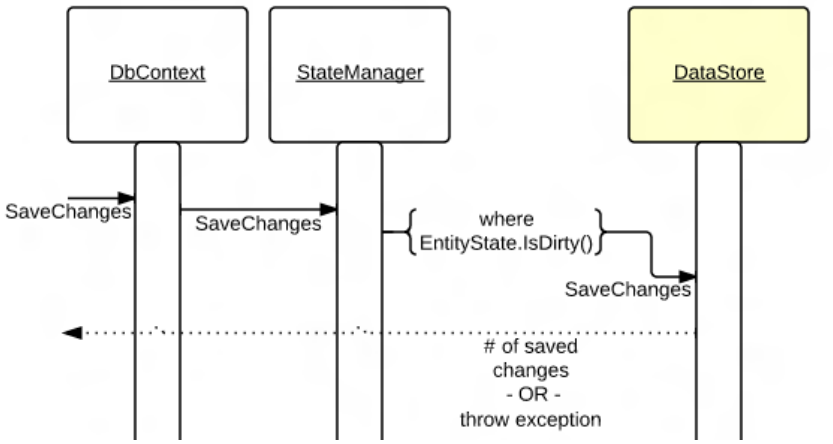
- MVC slijedi SoC princip.
    - Dobivaju se moduli maksimalne kohezije i minimalne sprege
    - Lakše održavanje, mijenjanje, testiranje, ponovna uporabljivost, ...
- **Object-relational mapping** (ORM) – postupak kojim se objekti aplikacije prevode u zapise u bazi podataka.
    - Model u MVC-u je klasa.
    - **Entity Framework** – Microsoftov ORM
        - Tri načina razvoja modela
            - Model First
            - Database First
            - Code First – model podataka se definira pisanjem klasa entiteta.
- EF podržava **Code First** stil razvoja aplikacije. Odmah se pišu klase neovisne o EF, a EF iz kôda zaključuje kako i gdje pohraniti instance.
    - Za zaključivanje o shemi baze na osnovu kôda, EF koristi niz konvencija.
        - Klasa objekata pohranjuje se u tablicu istog naziva kao klasa.
        - Svojstvo klase koje se zove ID ili imeklaseID mapira se u auto-inkrementirani primarni ključ.
        - Postoje i konvencije za strane ključeve, imena baze, ...
        - Konvencije se mogu i zaobići korištenjem **podatkovnih anotacija** – to su atributi kojima se definiraju svojstva entiteta podatkovnog modela.
            - Atributima se difiniraju deklarativna svojstva klase.
- Podatkovne anotacije koriste se za:
    - definiranje metapodataka koje EF koristi prilikom kreiranja baze,
    - validaciju na strani klijenta,
    - validaciju na strani servera – provjera podataka koje šalje klijent.
- Code First podržava **Fluent API** koji omogućava imperativno definiranje svojstava u kôdu umjesto deklarativnim anotacijama. Moguća je i kombinacija, pri čemu je Fluent API jači.
- Primjeri podatkovnih anotacija:
    - Required – definira da je određeno svojstvo obavezno. Služi za validaciju unosa tog svojstva na strani klijenta.
    - StringLength – definira veličinu string svojstva na strani servera i na strani klijenta.
    - Range – specificira minimalnu i maksimalnu granicu za numeričku vrijednost.
    - DataType – definira specifični tip podataka (specifičnije nego C#).
        - Npr. EmailAddress, MultilineText, ...
    - DisplayFormat – služi za formatiranje svojstava.
- Operator ?? omogućava definiranje null tipova vrijednosti. Vraća lijevi operand ako je različit od null, inače vraća desni operand.
- Tip varijable je **nullable** ako mu se može dodijeliti i null vrijednost. Podatkovni tip se može definirati kao nullable korištenjem oznake ? u deklaraciji (npr. int?).
- Atribut Column postavlja ime kolone u tablici koja odgovara klasi u kojoj se atribut koristi. Po defaultu je naziv kolone isti kao ime svojstva u klasi.
- Relacije među klasama izvode se iz konvencija ili se definiraju podatkovnim anotacijama ili Fluent API-jem.
- **Relacije** se definiraju kroz navigacijska svojstva klase – svojstva druge klase prema kojoj se definira relacija (navigacija). Code First na temelju navigacijskih svojstava zaključuje o relacijama među objektima.

- Svako svojstvo koje se zove jednako kao primarni ključ u drugom entitetu smatra se stranim ključem (nije case sensitive).
- Primjer za 1-N relaciju:

```
class Glazbenik {
        public int GlazbenikID { get; set; }
        [Required]
        Public string Ime { get; set; }
        Public virtual List<Album> Albumi { get; set; }
}
class Album {
        public int AlbumID { get; set; }
        [Display(Name = "Ime albuma")]
        public string Naziv { get; set; }
        [DisplayFormat(DataFormatString= "{0:dd/MM/yyyy}")]
        [Display(Name = "Datum izdavanja")]
        public DateTime? DatumIzdavanja { get; set; }
        public int GlazbenikID { get; set; }
        public virtual Glazbenik Glazbenik { get; set; }
}
```

  - Ukoliko u klasi postoji **svojstvo tipa druge klase** iz modela, EF zaključuje da postoji relacija između ta dva entiteta.
  - Kardinalnost se izvodi iz brojnosti svojstva druge klase.
  - Relacija treba biti definirana s obje strane.
  - Modalnost koja je defaultno null, postavlja se na not null tako da se iz klase doda obavezno svojstvo (npr. primarni ključ, u ovom slučaju GlazbenikID).
  - Default relacija je 1-N. Relacije 1-1 eksplicitno se definiraju u kôdu korištenjem ForeignKey atributa u klasi koja se tretira kao child u relaciji.
  - N-M relacije definiraju se dodavanjem svojstva tipa List na obje strane relacije.
- Code First zahtijeva klasu izvedenu iz klase DbContext kao sučelje prema bazi. Ona sadrži DbSet<T> svojstva gdje je T tip klase entiteta.
- **Lazy loading** – obrazac u programiranju prema kojem se inicijalizacija objekta odgađa sve do trenutka kada je taj objekt potreban, što pridonosi efikasnosti programa kada se pravilno koristi (suprotno od eager loading).
  - EF po defaultu podržava lazy loading
  - Kada se entitet pročita iz baze, podaci iz relacija nisu dohvaćeni. Oni se dohvaćaju automatski tek kada im se prvi put pristupa.
  - Kod eager loadinga podaci iz relacija dohvaćaju se zajedno s osnovnim entitetom.
  - Kod explicit loadinga relacijski podaci se dohvaćaju eksplicitno u kôdu.

    - Ako je baza spora ili ne treba dohvaćati baš sve podatke, bolje je koristiti lazy loading.
- DbContext koristi internu komponentu ObjectStateManager za upravljanje promjenama stanja entiteta.




- **Scaffolding** – mehanizam koji omogućava automatsko generiranje osnovnog kôda kontrolera i prikaza na temelju modela.
    - Imenuje kontrolere i prikaze, dodaje potrebni kôd u pojedine komponente, smješta generirane dijelove kôda u projekt, itd.
- Sinkronizacija modela i baze podataka obavlja se inicijalizatorima baze (samo prilikom razvoja jer kôd inicijalizatora ponovo kreira bazu pri svakom pokretanju)
- EF Power Tools – koristi se za vizualizaciju code first modela.

## Kontroler

- **Odgovara na zahtjeve** korisnika prema aplikaciji, tj. na HTTP zahtjeve i generira izlazne informacije prema korisniku.
- Obično koristi modele podataka.
- Svaki HTTP zahtjev mora ići preko kontrolera.
- Metode unutar kontrolera zovu se **akcije** (Actions).
    - Akcije moraju biti public i ne smiju biti static.
    - Akcija Index() je po defaultu u rutama definirana kao osnovna akcija koja će se pozvati ako ni jedna druga nije navedena u HTTP zahtjevu.
    - Obično se povezuju sa samo jednom aktivnošću korisnika (npr. klik na link).
        - Za funkcije koje se sastoje od dva koraka rade se dvije akcije.
    - MVC po defaultu sve metode kontrolera koje su public smatra akcijama. To se poništava NonAction atributom.
    - Kada se radi Create/Update/Delete, operacija se izvodi u dva koraka:
        - Prikaz forme preko HttpGet.
        - Prihvaćanje korisničkog ulaza preko HttpPost.
- Selektor **akcijskih glagola** (npr. HttpPost) je atribut koji se koristi za definiranje izbora akcija na osnovu metode u HTTP zahtjevu. Po defaultu akcija očekuje metodu GET.
- Obično je povratna vrijednost akcije objekt tipa klase izvedene iz apstraktne klase ActionResult.
    - Metoda View u klasi Controller kreira objekt tipa **ViewResult**.
        - View može imati ime prikaza kao parametar. Po defaultu je ime prikaza isto kao ime akcije.

- Prenošenje podataka **iz kontrolera u prikaz** moguće je na tri načina:
    - Strongly-typed prikaz – korištenje modela u prikazu koji se kreira pozivom metode View u kontroleru.
    - Korištenje ViewBag objekta – dinamički objekt koji je na raspolaganju samo promatranoj akciji. Zamjena je za ViewData objekt.
    - TempData – dinamički objekt sa string ključevima i objekt vrijednostima (kao i ViewData), ali je kod redirekcije na raspolaganju samo sljedećoj stranici.
- **Povezivanje modela** – mehanizam koji olakšava prosljeđivanje podataka s weba u kontrolere automatskim povezivanjem.
    - Model povezivanja po kojem radi defaultni povezivač (model binder) funkcionira na način da se imena ulaznih parametara iz HTTP zahtjeva mapiraju u parametre po imenu ako se radi o jednostavnim tipovima podataka, dok je za složene tipove potrebno navesti svojstvo s kojim se mapira.
    - Moguće je definirati i vlastite načine povezivanja.
- **Filteri** – modificiraju ponašanje akcije.
    - Mogu se postaviti na akciju ili na cijelu klasu kontrolera (tada se odnose na sve njegove akcije).
    - Tipovi filtera:
        - Autorizacijski filteri – izvršavaju se prije drugih filtera i prije kôda akcije.
        - Akcijski filteri – izvršavaju se prije i poslije akcije.
        - Filteri rezultata – izvršavaju se prije i nakon što se rezultat vratio iz akcije.
        - Filteri iznimki – izvršavaju se samo ako je akcija ili neki drugi filter generirao iznimku.
    - Npr. ValidateAntiForgeryToken – koristi se za zaštitu od CSRF napada.

## View

- Zadužen je za kreiranje sučelja MVC aplikacije prema korisniku.
- Prikazi se pohranjuju kao datoteke s ekstenzijom .shtml
- Moraju biti u poddirektoriju, koji je u Views direktoriju projekta i nazvan po kontroleru.
- Views/Shared direktorij sadrži zajedničke prikaze (npr. prikaz greške ili zajednički layout).
- Ime prikaza je obično isto kao i ime akcije u kontroleru koja poziva taj prikaz View metodom bez parametara.
- **Razor** – omogućuje kombiniranje HTML i C# kôda.
    - Znak @ označava početak C# kôda. Razor sam zaključuje gdje počinje HTML.
- Uobičajeni način povezivanja modela s prikazom je kreiranje prikaza pozivom metode **View** s objektima koji se prenose u prikaz kao argumentima – strongly-typed prikaz.
- **ViewData** – svojstvo u klasi Controller koje je instanca klase ViewDataDictionary koja služi postavljanju ili dohvaćanju rječnika za prikaz.
    - Ima svojstvo Model koje se postavlja na objekt koji predajemo metodi View prilikom kreiranja prikaza.
    - Klasa WebViewPage ima svojstvo Model koje poprima rječnik iz kontrolera.
    - @model deklaracija govori kompajleru kakav je model.
- **WebViewPage** – koristi se za dinamičko povezivanje. Ima Model svojstvo, ali podaci nisu čvrsto povezani.
    - Dinamičko povezivanje se koristi samo kad se prikaz ne povezuje uvijek s istim modelom.
- **Rasporedi prikaza** (layouts) pomažu održavati konzistentnost izgleda unutar više prikaza.

    - Raspored prikaza definira zajednički predložak za aplikaciju.
    - RenderBody – metoda koja u prikazu koji se koristi za raspored drugih prikaza iscrtava sve u prikazu koji je povezan s tim rasporedom.
- **Parcijalni prikazi** – koriste se za prikaz istog HTML sadržaja na više mjesta.
    - Imenuju se s _ na početku imena.
    - Mogu se prikazivati s podacima kao i obični prikazi.
    - Uključuju se s @Html.Partial helperom.
- HTML pomagači (helperi) su metode koje se pozivaju na Html svojstvu u prikazu.
    - Postoje i URL i Ajax pomagači.
    - Dijele se na:
        - Akcijske pomagače
        - Pomagače prikaza
        - Begin Form pomagače
        - Pomagače za editiranje
        - Validacijske pomagače
    - BeginForm – generira formu
    - Pomagači prikaza – npr. DisplayFor provjerava postoji li u anotacijama definirano određeno formatiranje i primjenjuje ga.
- Responzivni dizajn – podržan kroz Bootstrap.
    - Inteligentno prilagođavanje sadržaja različitim dimenzijama ekrana.
- Bundleing – povezuje različite JavaScript datoteke u jednu kako klijent ne bi morao slati puno HTTP zahtjeva prema serveru.

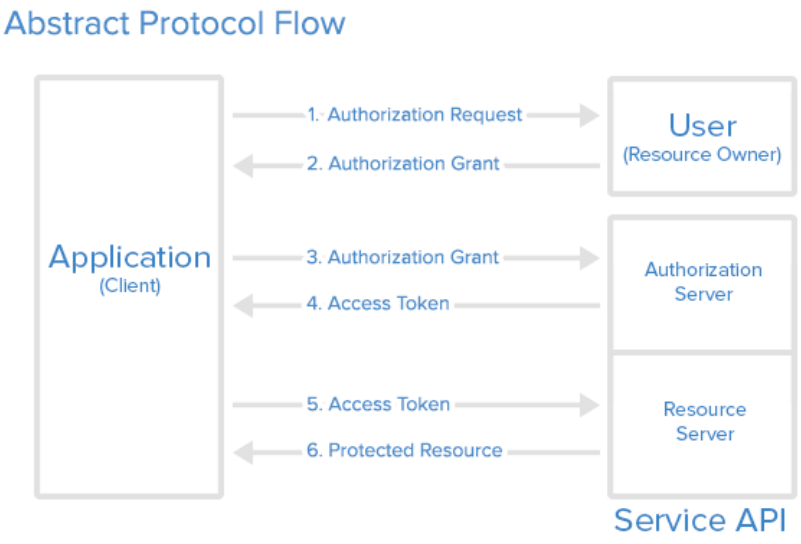
## Autentikacija

- Postupak identifikacije korisnika aplikacije na temelju korisničkog imena i lozinke koje potvrđuje ovlašteni autoritet.
- ASP.NET **implementira autentikaciju** preko autentikacijskih providera – gotovih modula kôda potrebnog za autentikaciju korisnika.
- ASP.NET **Membership** mehanizam – omogućava razvoj jednostavnog mehanizma autetikacije korisnika kroz web forme i MSSQL bazu podataka u koju se pohranjuju korisnički profili i autentikacijski podaci.
    - ASP.NET Simple Membership – isti mehanizam, ali prebačen s web formi na web pages.
    - ASP.NET Universal Providers – omogućavaju povezivanje s Azure SQL bazom i drugim bazama podržanim u EF-u.
- ASP.NET **Identity** – podržava autentikaciju i za web pages, web forms i MVC aplikacije.
    - Pruža podršku za više baza podataka, uključujući NoSQL.
    - Omogućava lako povezivanje aplikacije s drugim autentikacijskim aplikacijama poput Googlea ili Facebooka kroz OWIN middleware.
    - Podržava Account Lockout i Two-Factor Authentication.
- Two-factor authentication – pruža dodatnu razinu autentikacije uz korisničko ime i lozinku.
    - Korisnik mora imati dva od ova tri mehanizma kako bi se identificirao:
        - PIN, lozinku ili uzorak
        - ATM karticu ili uređaj (npr. mobitel)
        - Biometrijske informacije
- OWIN (Open web interfaces for .NET) – standard za definiranje sučelja između .NET aplikacije i web servera.

- Izbor autentikacije u projektu:

| Identity | Usage Scenarios |
|---|---|
| **Individual User Accounts (ASP.NET Identity with or w/o Social Identities)** | Internet apps, small and medium businesses, consumer apps |
| **Active Directory (AD)** | On-premises enterprise apps and users (LAN/VPN) |
| **Active Directory Federation Services (ADFS)** | On-premises enterprise apps with remote users |
| **Azure Active Directory (AAD)** | Cloud-based enterprise apps |

- Povezivanje s Google autentikacijom zahtjeva korištenje SSL-a (zbog HTTPS-a).
    - SSL se koristi i nakon logiranja radi zaštite login cookieja.
- **OAuth** – otvoreni standard za autorizaciju dizajniran za rad s HTTP protokolom.
    - Pruža klijentima pristup serverskim resursima korisnika koji se autenticira tako da definira proces preko kojeg se drugim aplikacijama dopušta korištenje resursa servera za autentikaciju, ali bez dijeljenja korisničkih podataka.
    - Autorizacijski server generira pristupne tokene za klijentsku aplikaciju, preko kojih se ona autenticira bez poznavanja korisničkih podataka.




    - Autentikacija korisnika se delegira servisu na kojem korisnik ima račun te se vanjska aplikacija autorizira za pristup računu.
    - OAuth 2 definira tok autorizacije za web, desktop i mobilne uređaje te samog vlasnika računa.

## Migracije, Ajax, SignalR

- ASP.NET MVC Code First **migracije** – služe za sinkronizaciju već deployane baze.
    - Omogućavanjem migracija stvara se direktorij Migrations u projektu.
    - Add-Migration – naredba koja provjerava postojeću bazu i kreira kôd koji ubaci promjene definirane u kôdu modela.
    - Update-Database – primjenjuje postojeće migracije na bazu.

- **Ajax** – omogućava ažuriranje dijelova prikazane stranice.
    - Integriran u ASP.NET MVC framework.
    - Uključen je u projekt preko jQuery biblioteka.
- JS kôd se piše u zasebnim datotekama u Scripts direktoriju.
- Uključivanje standardnih skripti radi se preko **bundleing** mehanizma – više datoteka se grupiraju u jednu koja se onda dohvaća samo jednim HTTP zahtjevom.
- Skripta koja se koristi u cijeloj aplikaciji uključuje se u _Layout.cshtml, a inače direktno u prikaz.
- **Parcijalno ažuriranje**
    - Npr. podatke u bazi osvježava neka druga aplikacija, a naša aplikacija treba korisniku periodički slati svježe podatke.
        - JS kôd periodički generira Ajax podatke za dohvat svježih podataka.
        - Kôd na strani servera dohvaća podatke iz baze na Ajax zahtjev.
        - U prikazu se poziva JS kôd.
- **Postupno unaprjeđenje** – strategija dizajna web aplikacija koja osigurava pristupačnost aplikacijama na način da se korisnicima neovisno o preglednicima pruži barem osnovni sadržaj stranice i osnovna funkcionalnost.
    - Nenametljivi JS jedan je od principa koji to omogućuju.
- **Bootstrap** – web frontend framework napisan u CSS-u i JS-u.
    - Automatski uključen u ASP.NET MVC projekt.
    - Responzivni dizajn, podrška za teme i gotove komponente.
- **SignalR** – biblioteka koja olakšava razvoj web aplikacija koje imaju funkcionalnosti koje se izvršavaju u stvarnom vremenu.
    - Web funkcionalnost koja se izvršava u stvarnom vremenu je mogućnost servera da klijentu šalje sadržaj u stvarnom vremenu, čim isti postane dostupan, bez da klijent zatraži taj sadržaj.
    - SignalR omogućava stvaranje dvosmjenih trajnih konekcija između klijenta i servera preko HTTP protokola.
    - Pruža API za kreiranje RPC poziva od servera prema JS funkcijama klijenta iz serverskog .NET kôda.
    - Uključuje upravljanje konekcijama.
    - Koristi WebSocket ukoliko ga i klijent i server podržavaju, inače se prebacuje na drugu raspoloživu tehnologiju.
    - Komunikacijski mehanizmi:
        - **Hub** – omogućava jednostavno pozivanje metoda između klijenta i servera.
            - Na strani servera izvodi se iz klase Hub iz SignalR paketa.
            - Na strani klijenta koristi se Hub proxy definiran u dinamički kreiranoj skripti /signalr/hubs.
        - **Persistent Connection** – spušta se na nižu razinu upravljanja i složeniji je.
    - Omogućava opsluživanja do 100 klijenata asinkrono ih obrađujući.
    - Svaki spojeni klijent ima svoju instancu huba.