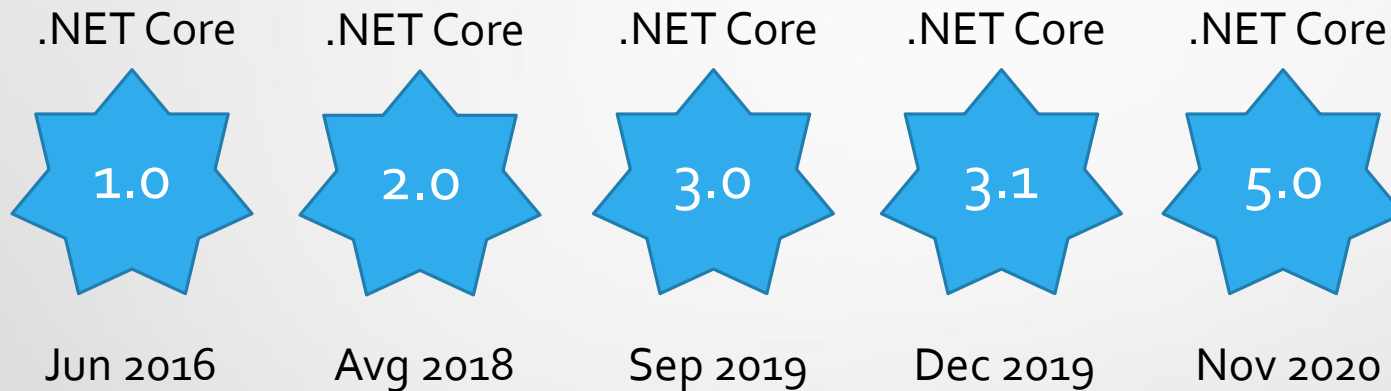




ASP.NET Core

SERVISNO-ORIJENTISANE ARHITEKTURE 2021

Istorija



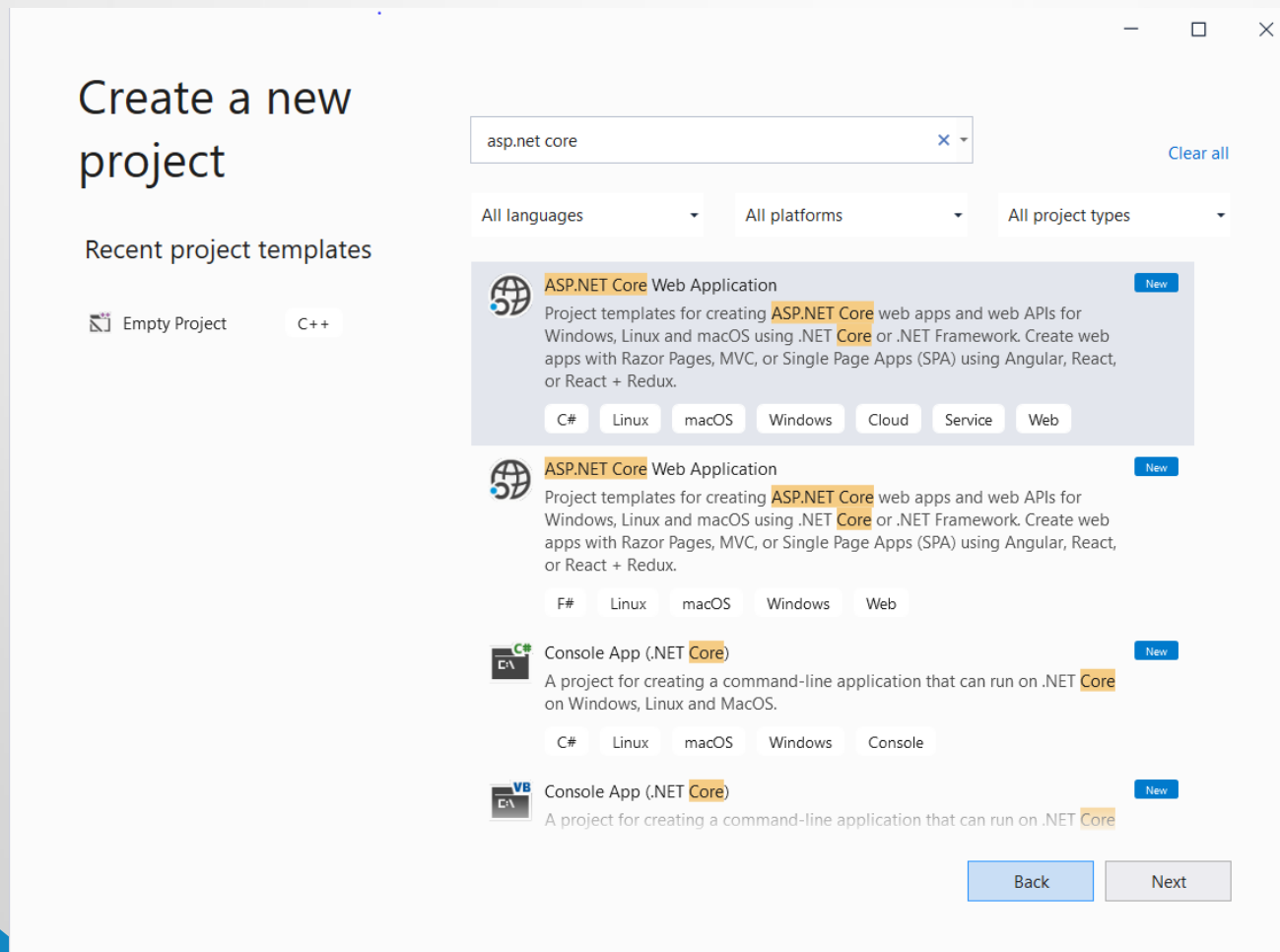
.NET CORE

- .NET Core je verovatno najveća izmena koju je .NET imao od svog početka
- .NET Core je besplatna, open-source, cross-platform verzija .NET Framework-a.
- .NET Core Framework se može koristiti za razvoj različitih tipova aplikacija (mobilne, desktop, web, cloud, IoT, ML, mikroservisi, igrice, ...)
- .NET Core uključuje core funkcionalnosti koje su neophodne za pokretanje .NET Core aplikacija. Ostale funkcionalnosti su dostupne kao NuGet paketi, koji se mogu po potrebi dodavati aplikaciji
 - Benefiti: poboljšane performanse, manji memorijski footprint, lako održavanje

ASP.NET Core

- ASP.NET Core je nova verzija Microsoft-ovog ASP.NET-a. Radi se o brzom, open-source web okviru koji podržava Windows, Mac i Linux.
- ASP.NET Core ima built-in podršku za dependency injection
- Update friendly
- Cloud friendly
- Odlične performanse

Kreiranje ASP.NET Core aplikacije - 1



Kreiranje ASP.NET Core aplikacije - 2

— □ ×

Configure your new project

ASP.NET Core Web Application C# Linux macOS Windows Cloud Service Web

Project name

Location

 ...

Solution name ⓘ

☐ Place solution and project in the same directory

Back Create


Kreiranje ASP.NET Core aplikacije - 3


×


Create a new ASP.NET Core web application


.NET Core


ASP.NET Core 5.0


**ASP.NET Core Empty**
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

**ASP.NET Core Web API**
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

**ASP.NET Core Web App**
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.

**ASP.NET Core Web App (Model-View-Controller)**
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

**ASP.NET Core with Angular**
A project template for creating an ASP.NET Core application with Angular

**ASP.NET Core with React.js**

Authentication

No Authentication
[Change](#)

Advanced

☒ Configure for HTTPS
☐ Enable Docker Support
(Requires [Docker Desktop](#))

Linux

☒ Enable Razor runtime compilation

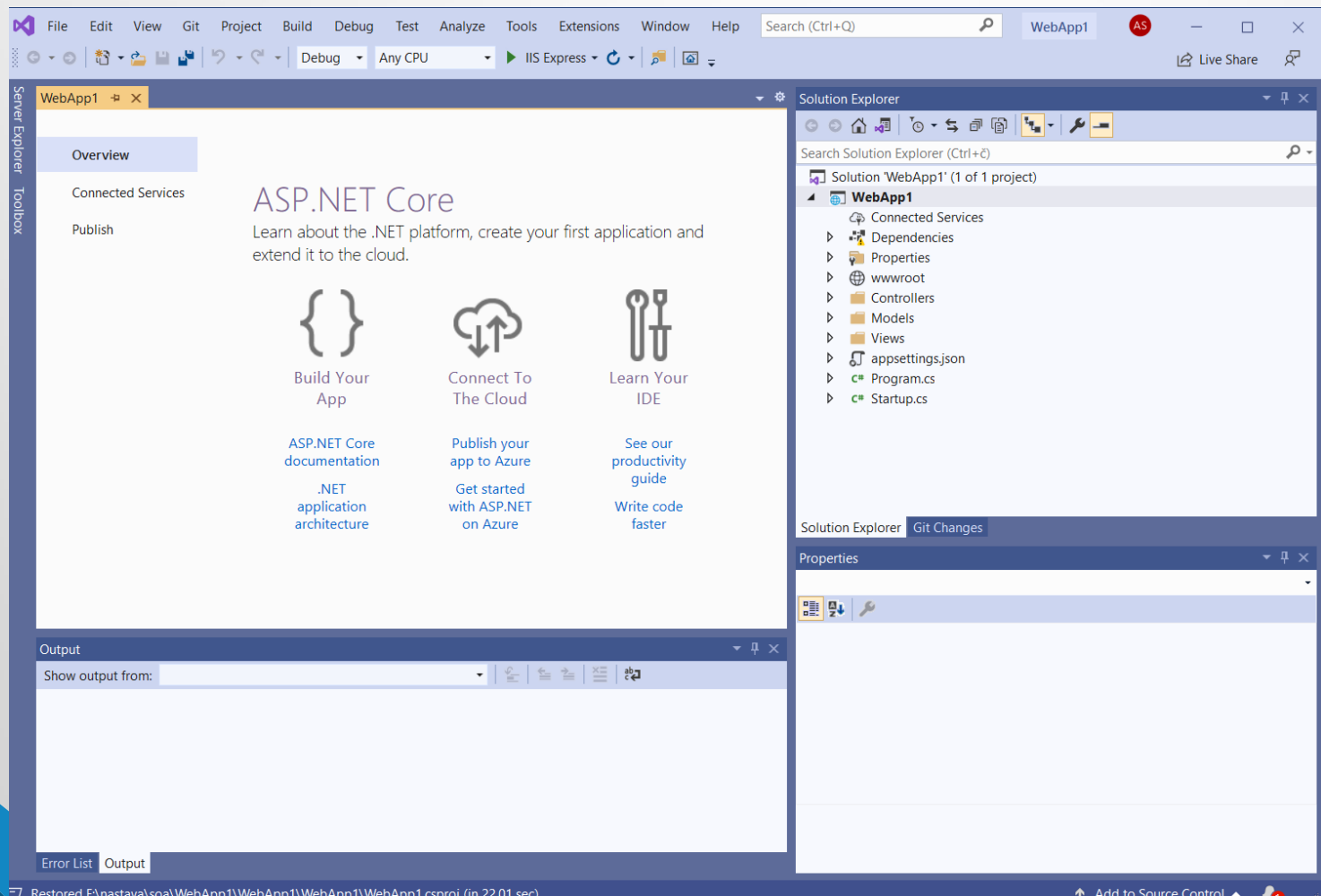
Author: Microsoft
Source: Templates 5.0.0

[Get additional project templates](#)

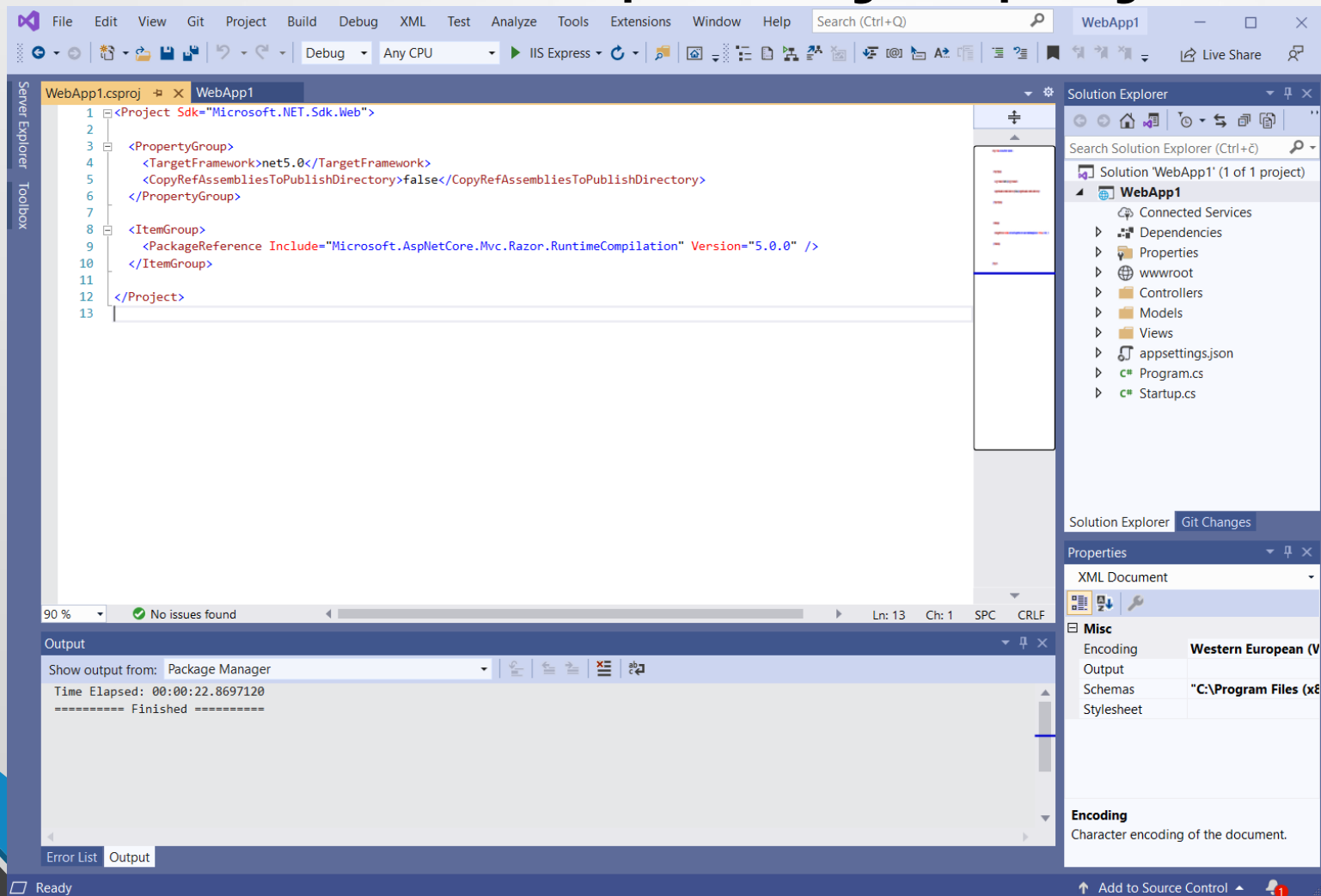
Back

Create

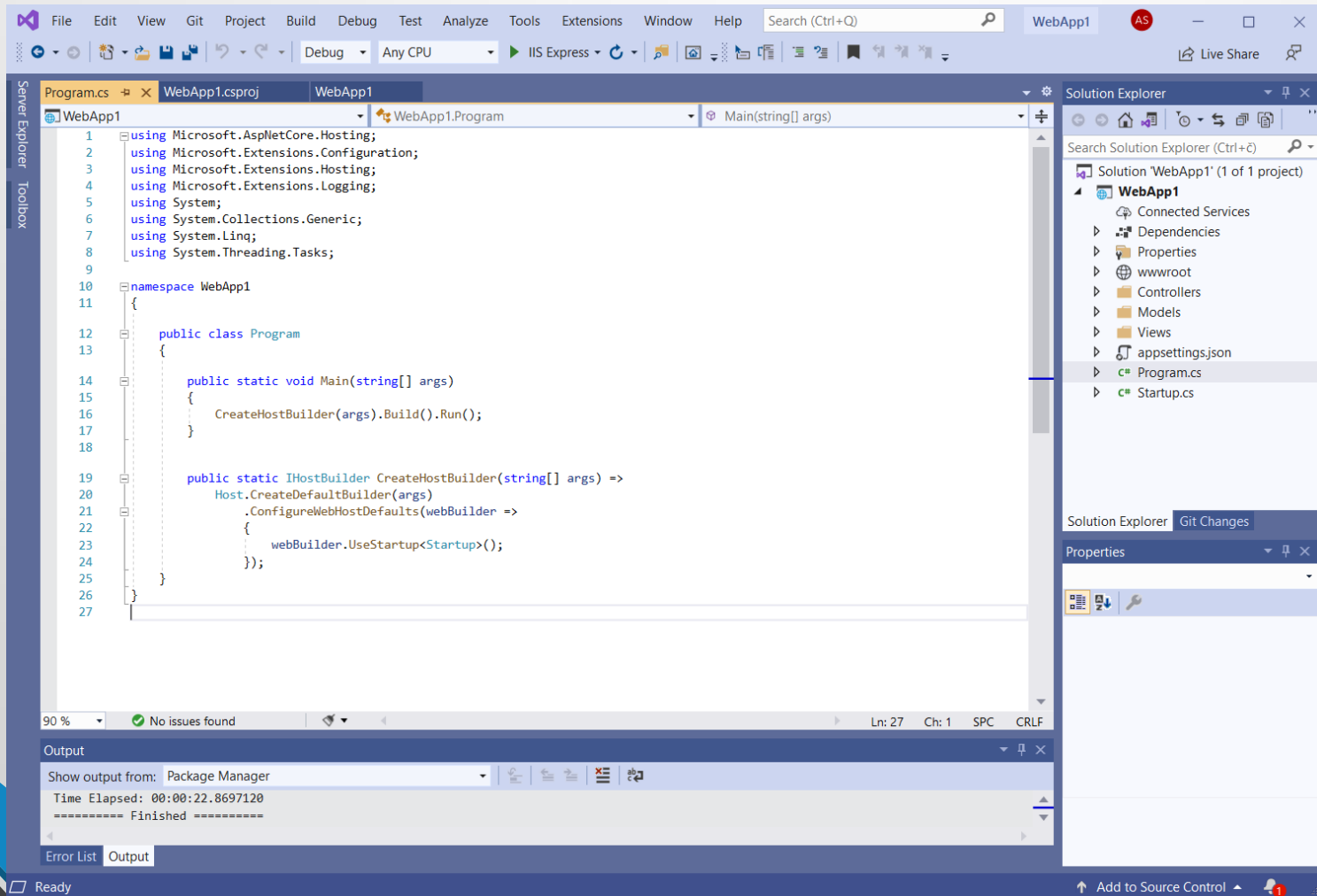
ASP.NET Core aplikacija - templatejt



ASP.NET Core aplikacija - projekat

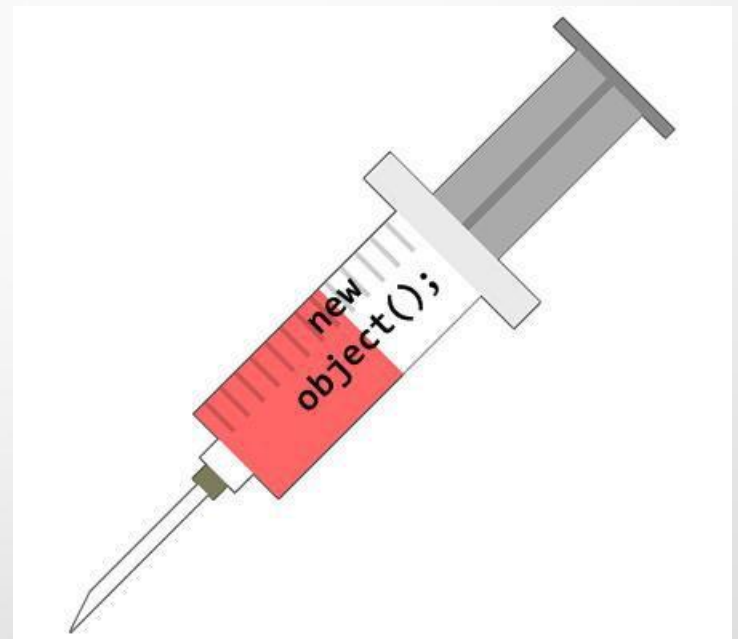


ASP.NET Core aplikacija – Program.cs



Dependency Injection

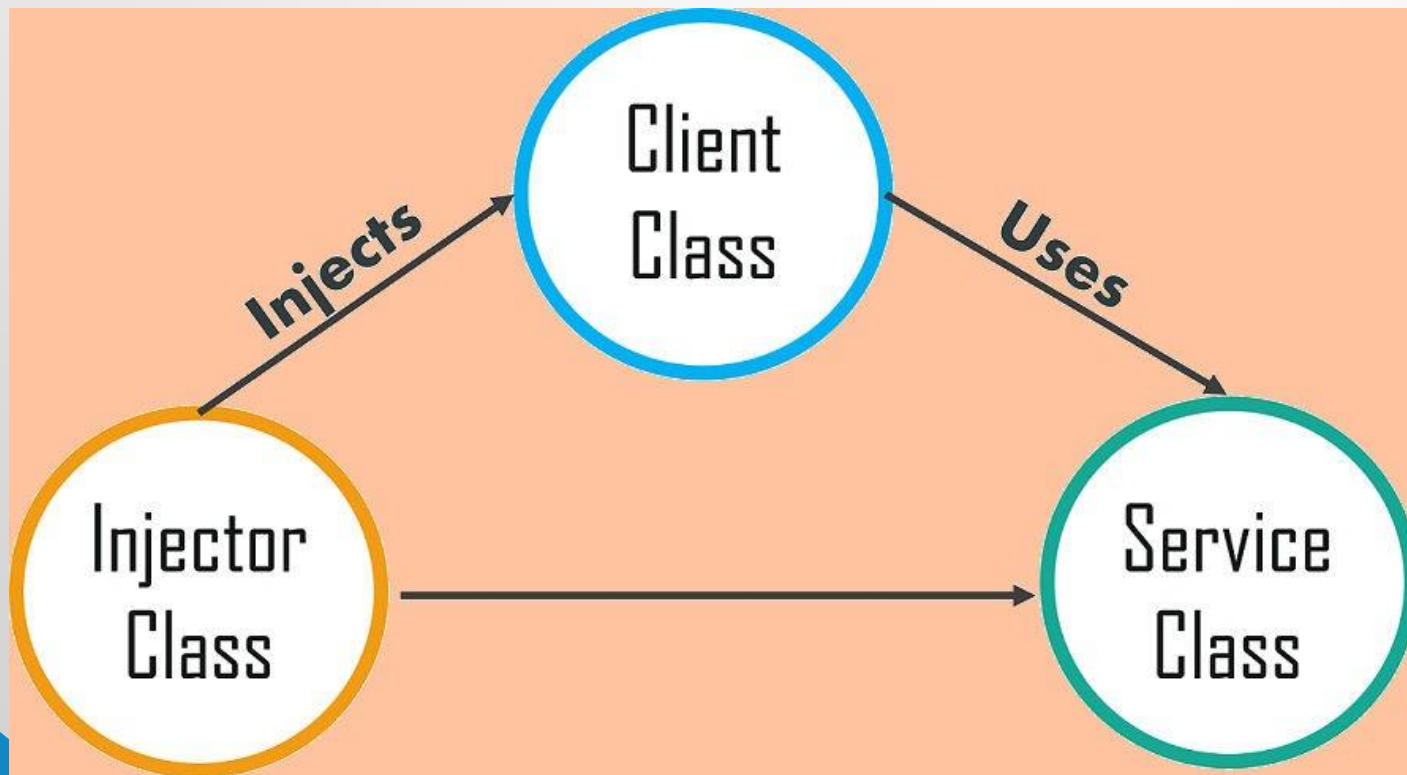
- Dependency Injection - projektni obrazac koji se koristi za pisanje koda u skladu sa SOLID principima razvoja softvera
- Ovo se odnosi na **D** deo skraćenice – Dependency Inversion – sa namerom da se omogući prilagođavanje klase koja pruža uslugu (servis) klasi koja uslugu konzumira



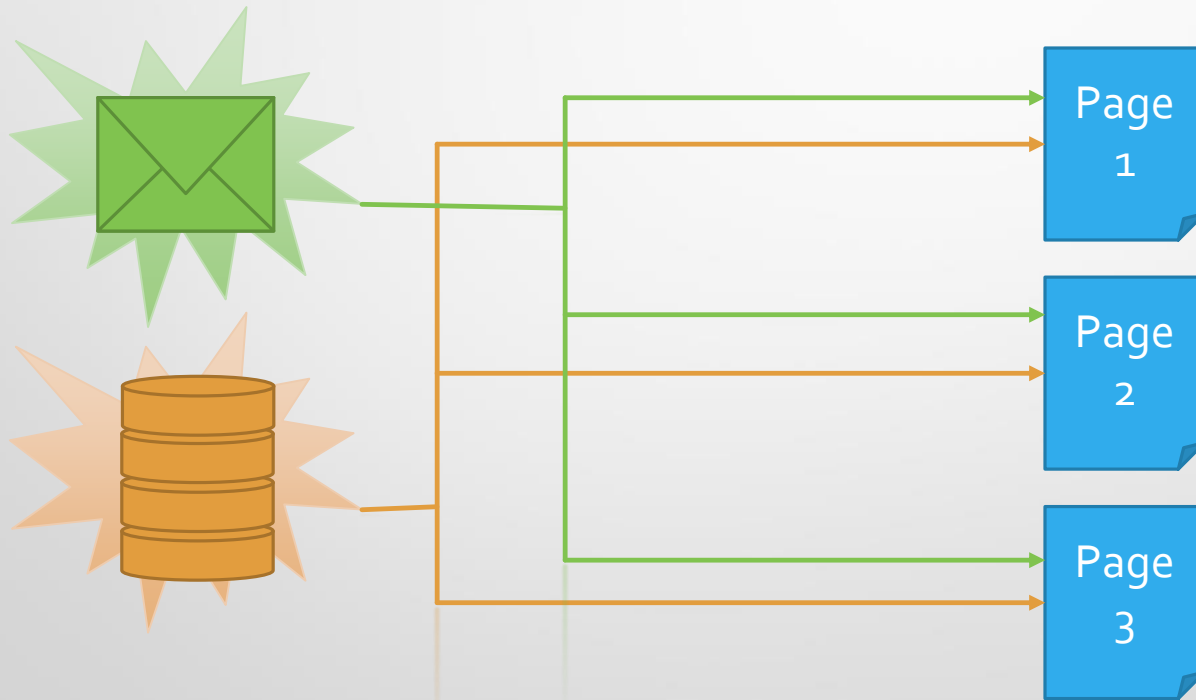
Dependency Injection

- Ovo se postiže uvođenjem interfejsa za servis, nakon čega se sve reference ka tom servisu čuvaju kao atributi klasa tipa tog interfejsa (a ne konkretnog servisa).
- Omogućava postojanje više različitih implementacija datog tipa servisa
- Mehanizam putem kojeg se konkretna implementacija datog servisa prosleđuje klasi koja ga koristi naziva se Dependency Injection
- U ASP.NET Core implementaciji - prosleđivanje servisa u konstruktor kontrolera

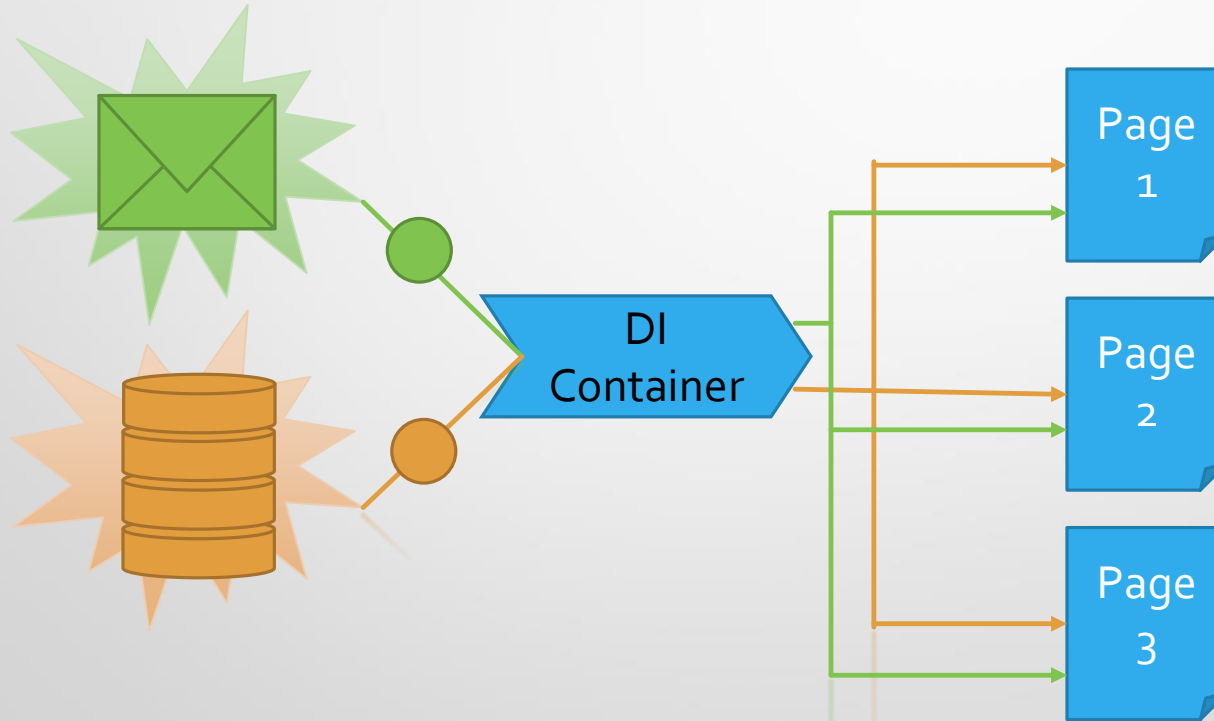
Dependency Injection



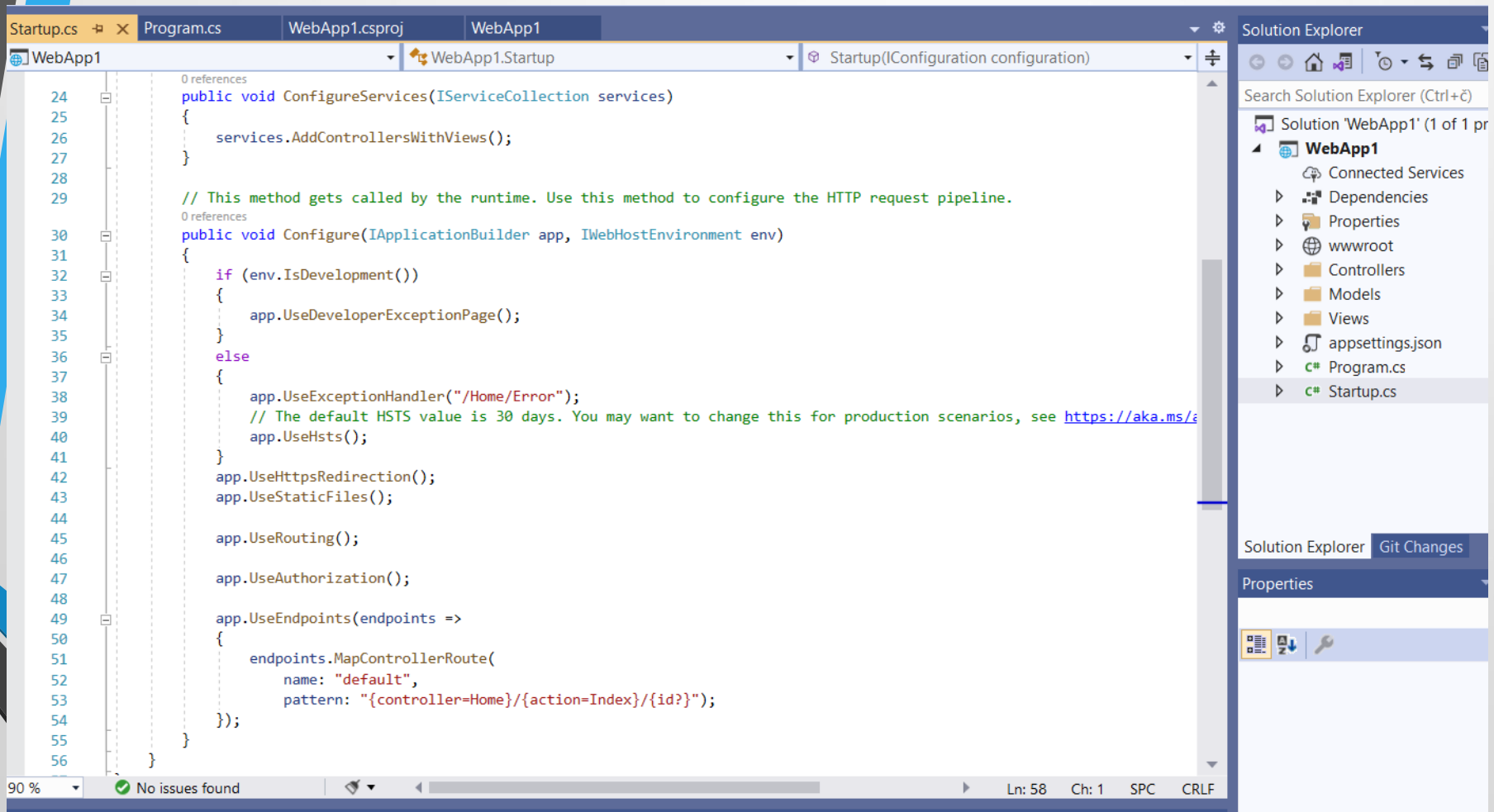
Problem



Rešenje: Dependency Injection

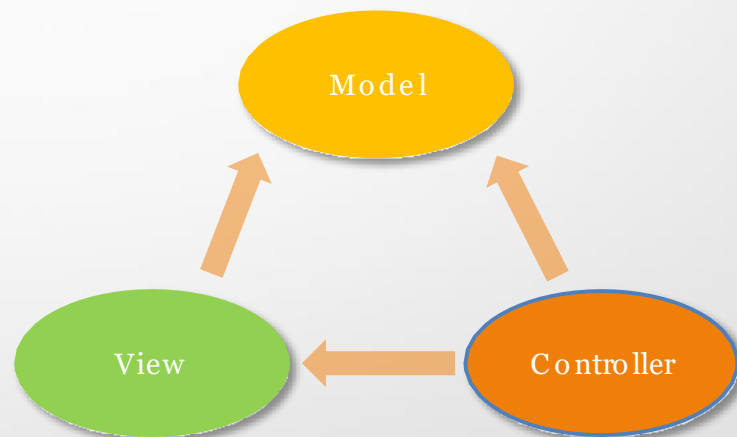


ASP.NET Core aplikacija – Startup.cs



MVC Arhitektura

- ASP.NET Core koristi patern koji se zove MVC – model-view-controller
- Pomaže u kreiranju oblika web aplikacije i Interakcije između komponenti koje koristi
- Patern još iz 1978. godine
- Prirodni ciklus: korisnik preduzima akciju, a kao odgovor aplikacija menja model podataka i predstavlja korisniku ažurirani pogled. Ciklus se dalje ponavlja...
- ASP.NET Core implementira varijantu MVC-a koja je specijalno pogodna za web aplikacije



MVC

- **Modeli** Sadrže reprezentaciju podataka sa kojima korisnik radi
- **View** se Koriste za renderovanje nekog dela modela kao korisnički interfejs
- **Kontroleri** Procesiraju dolazne zahteve i izvode operacije nad modelima i biraju view koji se renderuje korisniku

Model

View

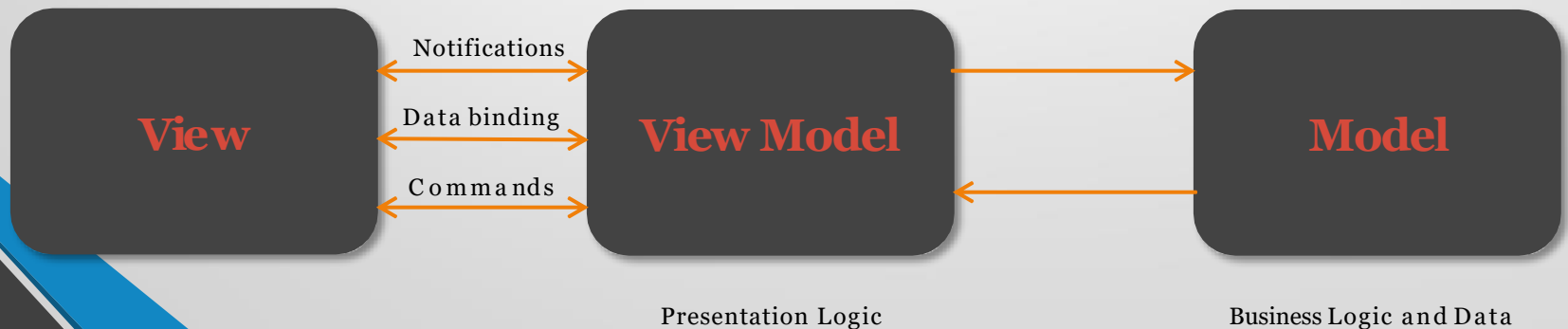
Controller

MVC

- Logika koja manipuliše podacima u modelu se nalazi samo u modelu
- Logika koja prikazuje podatke je samo u view-u
- Kod koji obrađuje korisničke zahteve se nalazi samo u kontroleru
- Sa ovako jasno razdvojenim segmentima, ASP.NET Core aplikaciju je lakše održavati i proširavati vremenom, bez obzira na to koliko velika postane.

MVC

- Iako moćan MVC svejedno ima sopstvene nedostatke. Ove nedostatke rešavaju neke od nadogradnji MVC pattern-a kao što su MVVM (model-view-viewmodel) i druge.
- ASP.NET Core implementira osnovnu varijantu MVC arhitekturnog obrasca koju krajnji korisnik (programer) može nadograditi po sopstvenim potrebama.



Modeli

Modeli u MVC-u sadrže podatke sa kojima korisnici rade.

Postoje dva tipa modela:

1. View model – koji prikazuje podatke prenete od kontrolera pogledu
2. Domain models – koji sadrže podatke u biznis domenu, zajedno sa operacijama, transformacijama i pravilima za kreiranje, smeštanje i manipulisanje tim podacima, koji se jednim imenom nazivaju logikom modela.

Modeli

Model treba da:

- Sadrži domenske podatke
- Sadrži logiku za kreiranje, upravljanje i modifikovanje domenskih podataka
- Obezbedi čist API koji prezentuje model podataka i operacije nad njim

Modeli

Model ne treba da:

- Otkriva detalje o tome kako je model podataka pribavljen ili kako se njime upravlja
- Sadrži logiku za transformisanje modela na osnovu interakcije korisnika (u kontroler!)
- Sadrži logiku za prikazivanje podataka korisniku (u view!)

Kontroleri

Kontroleri su spona u MVC paternu i koordiniraju između modela, podataka i pogleda.

- Kontroler treba da:
 - definiše akcije koje obezbeđuju biznis logiku koja upravlja modelom podataka i da obezbede podatke koje pogledi prikazuju korisniku.
 - sadrži akcije koje ažuriraju model na osnovu korisničkih interakcija.
- Kontroler ne treba da:
 - Sadrži logiku koja upravlja izgledom podataka
 - Sadrži logiku koja upravlja perzistencijom podataka

Pogledi

- Pogled - sadrži logiku koja je potrebna da bi se podaci prikazali korisniku
- Koriste se za prikupljanje podataka od korisnika da bi dalje mogli da se obrađuju od strane kontrolera.

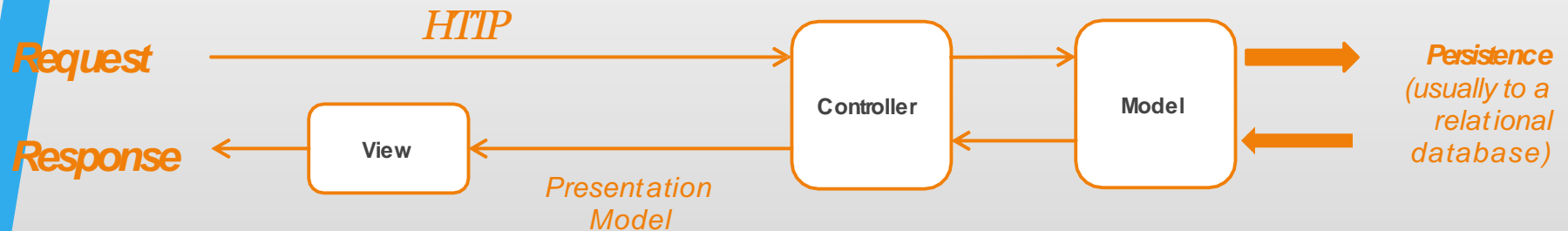
Pogledi

- Pogledi treba da:
 - Sadrže logiku i oznake za prezentovanje podataka korisniku
- Pogledi ne treba da:
 - Sadrže kompleksnu logiku
 - Sadrže logiku koja kreira, smešta i manipuliše modelom podataka

Pogledi

- U kontekstu ASP.NET Core aplikacije, pogledi predstavljaju najčešće specijalno formatirane html datoteke tako da ih može obraditi Razor template engine.
- Ovo ne mora biti nužno ovako – REST API treba da vraća JSON, a ne HTML!

ASP.NET Implementacija MVC-a



ASP.NET Implementacija MVC-a

- U ASP.NET Core-u su kontroleri C# klase
- Svaka javna metoda u klasi izvedenoj iz Controller-a je akcija, kojoj je pridružen URL.
- Kada se pošalje zahtev na URL povezan sa akcijom, naredbe u okviru te akcije se izvršavaju s namerom da se obavi neka operacija nad modelom i da se nakon toga selektuje pogled koji će te podatke prikazati klijentu.

Konvencija ne konfiguracija

- ASP.NET Core ima bogate sposobnosti za konfiguraciju sistema.
- Da li ovo znači da treba svaku komponentu konfigurisati individualno ??
 - NE!
- ASP.NET Core poseduje ustanovljene konvencije koje u praksi igraju ulogu podrazumevanih konfiguracija – ako pratimo konvenciju isti posao uradimo sa manje koda

ASP.NET Implementacija MVC-a

- ASP.NET Core MVC koristi view engine, koji se zove Razor - komponenta odgovorna za procesiranje pogleda da bi se generisao odgovor za web pretraživač.
- Razor pogledi su HTML templejti koji sadrže C# logiku koja se koristi da bismo obradili model podataka i tako kreirali dinamički sadržaj koji odgovara na promene u modelu.

ASP.NET Implementacija MVC-a

- Modeli su POCO klase – Plain Old C Sharp klase
- Postoji izričita potreba da se entiteti kojima aplikacija barata na neki način perzistiraju
 - Entity framework (data access layer) u ASP.NET-u
 - Kod Asp.Net Core-a situacija se razlikuje – entity framework core
- Ne poseduje interaktivan graficki interfejs koji uproscava rad, vec programer mora sam da odabere jedan od pristupa za uspostavljanje komunikacije sa bazom
- Code-First pristup – domain driven design – fokus na domenu aplikacije, počinje se kreiranjem klasa za naš domen pre dizajniranja baze

Implementacija kontrolera

- Osnovna klasa za sve kontrolere je *ControllerBase*
- Klasa *ApiController* je zamenjena atributom *ApiControllerAttribute*
- *ControllerBase* u sebi sadrži implementaciju čitave infrastrukture koja je potrebna za rad kontrolera
 - neki od nasleđenih atributa su *User*, *HttpContext*...
 - Neke od metoda *Ok*, *JsonResult*...
- Jedna od bitnih promena koja se izdvaja je direktna podrška za *Dependency Injection Pattern*

Implementacija kontrolera

- Omogućava Pisanje modularnih i testabilnih kontrolera
- Primeri injektiranih servisa su repozitorijumi entiteta, kontekst baze podataka, User manager servis...
- Kontroleri suštinski treba da predstave (eng. expose) skupove metoda kojima je moguće pristupiti standardnim http zahtevima na adekvatnim rutama.
- Dekorisanje metoda atributima – `AuthorizeAttribute`, `HttpGet`, `HttpPost`, `HttpPut`...
- Moguće je implementirati i nove attribute nasleđivanjem klase *Attribute*

View komponeta

- Kako bi se obezbedilo lako i efektivno manipulisanje podacima, MVC obrazac predviđa specijalne komponente za prezentaciju interfejsa – view componente
- Rad sa podacima je odvojen od prezentacije podataka
 - Omogućava brzu i elegantnu zamenu razlicitih klijentskih aplikacija
- ASP.NET Core nudi različite templejte projekata - nudi se direktna podrška za najpopularnije SPA frejmvorke (Angular, React, Vue.js)
- Majkrosoftov html templejt engine (razor), kao i od skora za novu tehnologiju zasnovanu na web assembly standardu – blazor.

Repozitorijumi

- Use case za Dependency Injection – **Repozitorijumi**
- Specijalan tip servisa čija je osnovna namena apstrakcija upita
- Ovo je izuzetno poželjno za moderne poslovne aplikacije - koriste se da kreiraju neophodne upite ka bazi i da nakon toga omoguće njihovo višestruko korišćenje u aplikaciji
- Ukoliko je kontroleru za neku od svojih metoda neophodan upit koji repozitorijum implementira, dovoljno je samo da injektira repozitorijum preko svog konstruktora. Nakon toga mu je dostupan u metodama
- Omogućavaju da se razdvoji logika obrade zahteva od logike za pribavljanje i perzistiranje entiteta.
- Omogućava da se piše DRY (Don't repeat yourself) kod

Repozitorijumi

- Repozičtorijumi omogućavaju centralizaciju ovih upita, pri čemu se svaka eventualna promena u strukturi jednog upita automatski propagira ka svim konkretnim instancama datog upita
- Još jedna povoljnost ovog pristupa je testabilnost – moguće je lako mokovati individualne rezultate upita, pri čemu se lakše izoluju funkcionalnosti Komponenti koje ih koriste.



Repozitorijumi

- Repozitorijumi i svi ostali servisi koji se koriste u aplikaciji u nekom trenutku moraju biti instancirani od strane sistema
- Krajnji programer pristupa samo interfejsu datog servisa (Dependency Inversion) - negde u programu se mora naglasiti koja će se konkretna implementacija servisa koristiti

Repozitorijumi

- ASP.NET Core nudi jedinstvenu lokaciju za registraciju svih servisa koji će biti korišćeni u aplikaciji – Startup klasa deklarirana u Startup.cs fajlu
- Ovde je moguće registrovati više različitih servisa i izvršiti njihovu konfiguraciju dinamički
- Ovaj pristup je na engleskom set it and forget it – jednom kada se izvrši konfiguracija servisa - on se može koristiti u nedogled!
- Za enterprise projekte ovaj pristup obezbeđuje veliku vremensku uštedu

Implementacija Repozitorijuma

```
public interface IRepository
{
    public IEnumerable<Model> Get();
    public Model GetById(int id);
    public Model Update(Model m);
}
```

```
public class MyRepository : IRepository
{
    private readonly DbContext ctx;

    public MyRepository(DbContext ctx)
    {
        this.ctx = ctx;
    }

    public IEnumerable<Model> Get()
    {
        return this.ctx.Models.ToList();
    }
}
```


NuGet

- Celokupna arhitektura ASP.NET Core-a je takva da se mogu lako primeniti principi modularnog programiranja - izdvajanja mislenih celina koja se kasnije mogu koristiti ne samo u tekućim, već i u kasnijim projektima
- .NET Core ima ugrađen menadžer modula (paketa) koji se zove NuGet
- Korišćenjem NuGet-a, moguće je instalirati veoma veliki broj dodatnih korisnih paketa koji implementiraju različite funkcionalnosti
- Neki od najkorisnijih paketa su Entity Framework Core, Identity...
- Ovaj stil programiranja omogućava veoma visok stepen recikliranja koda - minimizuje boilerplate kod

Entity Framework Core

- EF Core predstavlja paket koji krajnjem programeru nudi čist interfejs ka bazi i direktnu integraciju sa LINQ-om
- Najčešće se koristi u tandemu sa Entity Framework Tools- ima – skup alata koji automatizuju veliki deo interakcije sa EF Core-om i nudi funkcionalnosti kao što su kreiranje konteksta, migracija, pokretanje migracija, i sl.
- EF Core za razliku od klasičnog EF-a nudi veći broj API-jeva za rad sa modelom
- Fluent API za konfiguraciju EF-a i Code-First pristup zasnovan na migracijama
- Još jedna razlika u odnosu na regularan EF je činjenica da se ovde kontekst baze podataka koristi kao servis – može se injektirati u kontroler ili u drugi servis po potrebi, nakon čega mu se može pristupiti.

Nswag

- Nswag - popularna biblioteka koja se koristi za veći broj platformi (ASP.NET, ASP.NET Core i Node.js)
- Osnovna uloga Nswag-a je automatsko generisanje dokumentacije po OpenAPI I swagger standardima
- Nswag predstavlja niz alata koji omogućava automatsko kreiranje dokumentacije na osnovu source koda
- Omogućava i automatsko kreiranje većeg broja klijenata na osnovu već gotove dokumentacije – ovo mu faktički omogućava da brzo i elegantno generiše gotovog klijenta za proizvoljan API

Identity Core

- Identity Core predstavlja Core varijantu Identity servisa iz ASP.NET-a
- Ovde se podržava veoma veliki broj načina autorizacije i autentifikacije
- Jedna od osnovnih prednosti celokupnog Identity Core Framework-a je činjenica da je proširljiv i da je moguće ugraditi proizvoljan protokol u njega
- Neki od gotovih tipova navedenih servisa su oni bazirani na kolačićima, na jwt (json web token) standardu...
- Podržava se Roles based autentifikacija (na osnovu rola), Claims based (na osnovu iskaza/tvrdnji) i Policy based (ovde se definišu specijalne polise koje mogu da predstavljaju kombinacije Roles based, Claims based, kao i drugih proizvoljnih sistema koje je moguće implementirati).

Primer – TicketMaster

- Ticket Master – aplikacija koja omogućava prodaju karata
- Organizacije mogu da registruju svoje naloge, kasnije da se uloguju i da kreiraju svoje oglase za prodaju karata
- Korisnik može da se registruje, uloguje i da nakon toga pregleda sve moguće karte koje organizacije prodaju i da kupi neku od njih
- TicketMaster Code -
<https://github.com/teodorislava/ticket-master>

Za vežbu

- <https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-3.1&tabs=visual-studio>

Proći kroz tutorijal i isprobati kreiranje ASP.NET Core REST servisa