SUSTAV ZA AUTOMATSKO GENERIRANJE SUČELJA U WEB APLIKACIJAMA

Bebić, Filip

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Algebra University College / Visoko učilište Algebra**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:225:722694

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-25**

Repository / Repozitorij:

Algebra Univerity College - Repository of Algebra Univerity College

1.1.1.1 VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

SUSTAV ZA AUTOMATSKO GENERIRANJE SUČELJA U WEB APLIKACIJAMA

Filip Bebić

Zagreb, Prosinac 2019.

Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada.

U Zagrebu, 11.12.19.

Predgovor

U veljači 2018. godine zaposlio sam se u firmi koja se specijalizira za izradu programskih rješenja za upravljanje ljudskim resursima - HRPRO. Tada sam bio usred 2. godine preddiplomskog smjera programskog inženjerstva. Projekti su bili izazovni, ali i iznimno zanimljivi – ili sam barem tako mislio kad sam tek počeo raditi. Uskoro, nakon što je glavni programer na projektu podnio ostavku, našao sam se zadužen za izradu nekoliko web stranica za unos, pregled, mijenjanje i brisanje nekoliko desetaka setova podataka tj. „šifrarnika“.

Međutim, jako brzo, uslijedili su problemi. Pošto se projekt rasipao po velikom broju datoteka, svaka promjena na jednom mjestu bi zahtijevala replikaciju na minimalno 10 mjesta. Te promjene bi često bile odrađene nekonzistentno, tako da bi se iste greške pojavljivale nekoliko puta nakon što su „riješene“. U početku projekta smo mislili da smo korištenjem *Entity Framework-a* uštedili na vremenu ali kako je vrijeme protjecalo, postajalo je sve očitije da je ta odluka bila dvosjekli mač. Naime, kako se projekt razvijao, razvijao se i model baze, a svaka promjena na bazi morala se odraziti na generiranom kôdu *EF-a*. Poteškoće, uzrokovane kako našim manjkavim znanjem o *ORM*-ovima tako i ograničenjima alata koji smo imali uzrokovala su brojne poteškoće i kašnjenja.

Nakon jednog jako dugog dana punog hvatanja grešaka rasutih po kôdu, odlučio sam se napraviti vlastiti sustav čiji bi glavni zadatak bio centralizacija svog tog kôda na jedno mjesto. Cilj ovog projekta je bio i ostao osigurati da, kada developer jednom riješi grešku, ona ostane riješena.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

U ovom radu je opisan sustav za generiranje koda sučelja internetskih formi u svrhu ubrzavanja razvoja web aplikacija. Iako na tržištu postoje brojni alati koji postižu slične rezultate, trenutno niti jedan od njih nema značajne mogućnosti integracije u postojeće podatkovno okruženje. U svom radu pokazao sam mogućnost izrade ovakvog sustava sa minimalnom količinom kôda i sa potrebnim kapacitetima za proširenje funkcionalnosti kakav je nužan za praktično poslovanje.

Ključne riječi: ORM, automatizacija, MVC, Razor, šifrarnik, refleksija, atributi

Sadržaj

1. Uvod 1

2. Temeljni zahtjevi sustava i njihova problematika 2

2.1. Opseg 2

2.2. Razmatranje sustava za relacijsko objektno mapiranje**Error! Bookmark not defined.**

2.2.1. Problem prilagodbe u postojeće okruženje u pogonu 3

2.2.2. Razmatranje postojećih ORM sustava 3

3. Alati i mehanizmi rada 5

3.1. Refleksija 5

3.2. Programiranje orijentirano prema atributima 5

3.3. Nasljeđivanje i polimorfizam 7

4. Arhitektura sustava 8

4.1. Platforma ASP.NET Core 8

4.2. Razor sustav 8

4.3. Sustav za dostavljanje datoteka 9

4.4. Korišteni atributi 10

4.3.1 Atributi razreda 11

4.4.2 Atributi svojstava 13

5. Primjeri dinamičkih stranica 17

5.1. Jednostavni primjer 17

5.2. Realistični primjer 17

5.3. Kompleksni primjer 19

Zaključak 23

Popis kratica 24

Popis slika 25

Popis tablica 26

Popis kôdova 27

Literatura 28

Prilog 29

2. Uvod

Tokom svog rada na razvoju web aplikacija uočio sam da se velika količina programerskog vremena troši na pisanje sličnog, ako ne identičnog, kôda za grafičko sučelje. Zbog toga sam odlučio ispitati mogućnosti korištenja sustava za dinamično generiranje grafičkog sučelja za ubrzavanje razvoja web aplikacija i povećanje održivosti centralizacijom kôda velikog broja web stranica. U ovom radu ću opisati i u praktičnom radu implementirati izvedbu ovakvog sustava u .NET Core platformi na primjeru *enterprise level* aplikacije. Iako na tržištu već postoje neka gotova rješenja, ona obično pate od manjka proširivosti i prilagodljivosti što ih čini neprikladnim za implementaciju u kompleksne sustave kakve susrećemo u praksi. Ovaj rad će se fokusirati na osnovne operacije sa setom podataka – CRUD. Iako je pisanje kôda za grafičko sučelje za jednostavne zadatke poput promjene, brisanja, dodavanja i pregleda podataka relativno jednostavno, ono ipak oduzimanje značajan dio vremena prilikom razvoja skoro svake aplikacije. Štoviše, kôd napisan za tu svrhu je obično repetitivan, pa se pisanje istog sastoji od kopiranja i lijepljenja komada postojećeg kôda u nove dijelove aplikacije. Također, takvo pisanje repetitivnog kôda je percipirano od strane programera kao dosadno i kao takvo negativno utječe na moral u razvojnom timu. Uz sve gore napomenuto, taj kôd ima još jednu ključnu negativnu karakteristiku. Naime, uzastopnim kopiranjem istog kôda udaljavamo se od jednog od temeljnih koncepata u objektno orijentiranom programiranju - apstrakcija kôda. Ovakav sustav ublažuje taj problem centralizacijom svog kôda zajedničkog svim stranicama.

3. Temeljni zahtjevi sustava i njihova problematika

Svaki projekt mora počinjati sa izlaganjem njegovih ciljeva, a cilj uvođenja bilo kakve automatizacije je uvijek isti – ušteda vremena i novaca, smanjenje pogrešaka u radu i oslobađanje radnika za produktivniji posao. Cilj ovog projekta je sve to, međutim, automatiziranje je također zahtijevan posao koji ima svoju cijenu. Stoga je potrebno izračunati točan opseg projekta te njegove granice. Odmah na početku ovog rada odrediti ćemo iste.

3.1. Opseg

Teoretski, možemo generirati cijeli klijentski sloj u pozadini, međutim, bilo kakav proizvod koji se generira na takav način bi bio krajnje neprijatan za krajnjeg korisnika zbog nedostatka bilo kakve posebnosti i monotonog dizajna. Stoga ograničavamo ovaj projekt na vrlo specifičan dio web aplikacija – *šifrarnike*. Šifrarnik će za naše potrebe biti definiran kao jednostavni set podataka čiji je koji se sastoji od identifikacijskog kôda i skupom podataka koji se veže 1 na 1 uz taj kôd. Za uređivanje istog potrebna nam je tablica za pregled svih unosa, forma za pregled jednog unosa, stvaranje novog unosa, te forma za uređivanje i brisanje postojećeg unosa. Sustav će biti dovoljno fleksibilan da se može proširiti kôdom i na svakom sloju web aplikacije – klijentskom, aplikacijskom te podatkovnom.

3.2. Razmatranje sustava za relacijsko objektno mapiranje

Na tržištu već postoje neki alati koji postižu sličan cilj našeg sustava. Možemo ih podijeliti na ORM[1] alate kao *Hibernate* i *Entity Framework*(EF) te sustave za automatsko generiranje web formi. EF je naj sličniji onome što nastojimo postići tako da ću njega koristiti kao primjer postojećeg sustava.

[1] Object-relational mapping – Označava sustav za mapiranje setova podataka u paradigmi objektno orijentiranog programiranja

3.2.1. Problem prilagodbe u postojeće okruženje u pogonu

Prije nego što počnemo razmatrati postojeće sustave moramo se osvrnuti na probleme uvođenja ovakvog sustava u postojeće okruženje. Naime, svaki projekt je puno jednostavniji kada se njegovo planiranje počinje od nule, bez bremena postojeće infrastrukture koja mora biti podržana i održana bez obzira na uvođenje novih elemenata u postojeći sustav. Međutim, to u praksi često nije moguće, tako da koji god sustav odaberemo za automatsko generiranje sučelja, moramo imati na umu da taj sustav mora podržavati postojeći kôd, procedure u bazi, itd.

To se u praksi svodi na podržavanje ne optimalne strukture baze, različitih formata poziva podatkovnom sloju te raznog kôda koji je nastao prije no što su uvedene stroge kontrole kvalitete kôda. Najčešći primjeri ovakve problematike su:

1. Nestandardizirana forma slanja identifikacijskog ključa u bazu prilikom CRUD operacija. Nekada se šalje kao *output* parametar, nekada kao input, a nekada se uopće ne šalje.
2. Nepovezanost imena polja u web formi, objekata u aplikacijskom sloju, objekata u klijentskog sloju i objekata u podatkovnom sloju.
3. Nekonzistentno pretvaranje tipova iz aplikacijskog sloja u tipove podatkovnog sloja. npr. Nekada je tip *Boolean* u bazu zapisan kao *Integer*, a nekada kao *Bit.*

3.2.2. Razmatranje postojećih ORM sustava

Uzeti ćemo *Entity Framework* kao primjer postojećeg ORM sustava zato što je to jedan od naj razvijenijih i naj fleksibilnijih ORM sustava na tržištu.

PREDNOSTI

1. Brza izrada kôda – naravno, ovo je glavna prednost svakog ORM sustava, međutim, ona ima i svoju cijenu
2. Automatizirana migracija – Microsoft se pobrinuo da se na klik gumba kôd može ažurirati tako da podržava naj novije verzije povezanog *software*-a
3. Manje kôda – *Linq* jezik nam omogućuje lakše upite bazi s manje repetitivnog kôda.

NEDOSTATCI

1. Komplicira stvari – zbog načina na koji *EF* implementira keširanje i transakcije u svrhu optimizacije sustava, zahtjeva vrijeme za prilagodbu programskih inženjera svom načinu rada
2. Ograničenja na veličinu i strukturu baze podataka – *EF* zahtjeva potpuno standardiziran i jednostavan model tablica za normalnu *out-of-the-box* funkcionalnost.

4. Alati i mehanizmi rada

U izradi ovog sustava koristiti ćemo nekoliko alata i programskih mehanizama kako bi postigli cilj maksimalne automatizacije.

4.1. Refleksija

Refleksija je sposobnost aplikacije da pregledava i mijenja vlastitu strukturu kôda. Naj očitiji primjer refleksije u praksi je serijalizacija i deserijalizacija objekata u aplikaciji. U tom procesu, aplikacija čita svojstva na određenom razredu i sistematski ih zapisuje u nekom obliku iz kojeg ih može kasnije ponovo pročitati istim mehanizmom i tako obnoviti taj objekt. Objekti se obično serijaliziraju u binarnom obliku ako će ih ista aplikacija pročitati kasnije i u *JSON* ili *XML* obliku ako su ti objekti namijenjeni nekoj drugoj aplikaciji ili aplikacijskom sloju. U ovom radu, koristiti ćemo refleksiju za čitanje atributa razreda i svojstava, čitanje liste svojstava, utvrđivanje postojanja određenih metoda te pozivanje istih. Također, jedna od glavnih primjena refleksije je testiranje *mock-up* objektima. Mi ćemo je koristiti za istu svrhu, tako da ćemo gledati tip svojstava te njihove nazive kako bi generirali *mock-up*-e za učinkovitije i realističnije testiranje dinamičkih razreda.

4.2. Programiranje orijentirano prema atributima

Atributi nam daju jednostavan način da povežemo entitet(svojstvo, razred ili funkcija) s meta podacima i deklarativnim informacijama. Zapravo, glavna prednost atributa je da navedene ciljeve postignemo s manje kôda. Trenutno u c#-u, atributi mogu služiti samo za dohvat u pokrenutoj aplikaciji, a ne, kao naprimjer u Javi, za dohvat meta podataka pri kompilaciji aplikacije. Također, svojstva atributa moraju sadržavati samo jednostavne tipove podataka tj. konstante tipova *string, integer, boolean* etc.

Usprkos nedostatcima, zbog svoje sposobnosti da počisti i pojednostavni kôd, atributi se nameću kao preferirana metoda asocijacije entiteta s meta podacima. Naprimjer. Ukoliko želimo označiti neko polje kao obvezno, možemo napisati funkciju koja će provjeravati ispunjenost polja i dati joj listu imena polja koja mora provjeravati. U takvom sistemu, čitali bi imena svih svojstava objekta koji je došao na server sa forme refleksijom i to bi funkcioniralo. Međutim, imali bismo centraliziranu listu imena atributa na nekom mjestu, te

ista ne bi bila provjeravana tokom kompilacije aplikacije već tek tokom samog izvršavanja kôda. Nedostatci takvog pristupa su očiti. Prvi je nepreglednost takvog kôda tj. Vizualna nepovezanost te liste te samog svojstva na listi, a drugi je to što, ako promijenimo strukturu razreda, kompajler neće baciti grešku, već će kompilacija proći, a aplikacija će se raspasti tokom korištenja. Ovakve probleme možemo riješiti uvođenjem atributa kojeg ćemo koristiti iznad obveznog svojstva i jednostavno registrirati *middleware* koji će prilikom svakog dospijeća objekta na izloženu *HTTP* metodu u aplikaciji provjeravati ispunjenje onog svojstva dospjelog objekta koji je označen tim atributom. Dapače, takav atribut već postoji i dio je .NET i .NET Core sustava i mi ćemo ga koristiti u sustavu za automatsko generiranje sučelja.

Programiranje orijentirano prema atributima označava tehniku programiranja gdje programer nastoji što više jednostavnih funkcionalnosti postići preko atributa. Ova tehnika je sve popularnija zbog još jedne važne prednosti atributa, a to je da se pri asocijaciji meta podataka atributima, taj kôd nalazi točno tamo gdje se i logički očekuje, a to je neposredno uz entitet kojemu se podaci pripisuju.

U našem sustavu mi ćemo atribute koristiti za inicijalno prikupljanje podataka o dinamičkoj stranici te o pojedinim svojstvima. S ovim ćemo postići centraliziran kôd koji nema veliki utjecaj na jasnoću kôda, pošto su atributi pojedinog svojstva uvijek vizualno grupirani uz to svojstvo. Više o pojedinim atributima koje ćemo koristiti u sustavu te objašnjenje njihove funkcionalnosti se nalazi u poglavlju 4.4 Korišteni atributi.

Kako to dohvat atributa svojstva izgleda u kôdu:

```
public static Dictionary<string, int?> GetMaxLengths(Type razred)
{
    Dictionary<string, int?> maksimalneDuljine = new Dictionary<string, int?>();
    List<PropertyInfo> svojstva = razred.GetProperties().ToList();
    foreach (PropertyInfo svojstvo in svojstva)
    {
        MaxLengthAttribute atributMaksimalnaDuljina;
        atributMaksimalnaDuljina =svojstvo.GetCustomAttribute<MaxLengthAttribute>();
        maksimalneDuljine.Add(svojstvo.Name, atributMaksimalnaDuljina?.Length);
    }
    return maksimalneDuljine;
}
```

Kôd 1 – Primjer dohvata atributa svojstava refleksijom

4.3. Nasljeđivanje i polimorfizam

Nasljeđivanje je koncept u objektno orijentiranom programiranju koji se koristi za ponovno iskorištavanje napisanog kôda nasljeđivanjem razreda od strane drugih razreda. Razred koji nasljeđuje neki drugi zove se podrazred i on, činjenicom da mu je naslijeđeni razred nadrazred, automatski implementira sve metode i svojstva podrazreda. Ovo je iznimno korisno jer nam omogućuje, ne samo da zajednički kôd izlučimo u nadrazred, nego i da u taj nadrazred stavimo i neki osnovni kôd i ostavimo opciju da programer u specifičnom slučaju prepiše samo neke od metoda predloška. Takve metode, koje obavljaju različite funkcije ovisno o podrazredu, zovemo polimorfne metode. Metoda je polimorfna ako radnja koju metoda obavlja ovisi o objektu na kojeg se primjenjuje.[2] Polimorfnost je jedna od važnih osobina koje izdvajaju objektno orijentirano programiranje.

[2] Maja Čić, Uvod u programiranje, FESB

5. Arhitektura sustava

5.1. Platforma ASP.NET Core

Originalni .NET je sustav koji nadograđuje mogućnosti samog operativnog sustava(MS Windows). Radi se o posebnoj infrastrukturi koja programerima nudi gotova rješenja i funkcionalnosti da bi ubrzala i pojednostavila razvoj aplikacija svih vrsta i oblika. Najvažnija sastavnica .NET sustava zove se *Common Language Runtime* ili skraćeno CLR. CLR je softverski sustav u kojem se kôd izvršava. Kada korisnik pokrene aplikaciju pisanu za .NET Platformu, CLR ju izvršava kako bi joj osigurao stabilnost i funkcionalnost. Instrukcije u programu se u realnom vremenu prevode u izvorni strojni kôd koji razumije računalo. Za taj je posao zaslužan JIT-kompajler (eng. *Just In Time*).[3]

ASP.NET Core je platforma(eng. *Framework*) koji je izrastao iz .NET sustava, a koji kombinira najvažnije prakse iz Agile razvoja, urednost kôda MVC arhitekture, te najbolje dijelove .NET-a.[4] Glavna prednost .NET Core sustava jest njegova podrška za razne operacijske sustave i okoline, te njegova izoliranost od okruženja u kojem se aplikacija izvodi. Ja sam ga u ovom radu odabrao baš zbog tih karakteristika.

ASP.NET	**ASP.NET Core**
Može se koristi za razvoj WPF, Windows Forms aplikacija	Može se koristi za razvoj UWP aplikacija
Zatvoreni kôd	Otvoreni kôd
Glavna platforma je MS Windows	Podržava razvoj aplikacija za Linux

Tablica 1 - Razlike .NET i .NET Core sustava

5.2. Razor sustav

Razor(eng. *Razor Engine*) je sintaksa koja služi za uključivanje serverskog kôda u web stranice. Sintaksa Razor-a sastoji se od HTML-a, C# i Razor oznaka. Datoteke koje

[3] https://hr.wikipedia.org/wiki/.NET_Framework
[4] Adam Freeman; Pro ASP.NET Core MVC(6th edition); ISBN-13: 978-1484203989, ISBN-10: 1484203984

sadrže razor kôd u pravilu imaju ekstenziju .cshtml. Sastavljanjem Razor kôda dobivamo HTML koji se može prikazati u običnom web pregledniku.[5] U ovom radu koristit ću Razor za izradu posrednog kôda koji će se na posljetku sastaviti u HTML.

```
<ul>
  @foreach (var p in products) {
     <li>
         @p.ProductName

         @if (p.UnitsInStock == 0) {
            <text>(Out of stock!)</text>
         }
     </li>
  }
</ul>
```

Slika 1 - Primjer Razor sintakse

5.3. Sustav za dostavljanje datoteka

U .NET Core sustavu pogledi(eng. *Views*) se mogu sastaviti u DLL-ove radi bržeg pristupa, ali obično se spremaju u datoteke s cshtml ekstenzijom koji se dohvaćaju dok se aplikacija izvodi.[6] MVC nam omogućava da ga nadogradimo sa vlastitim sustavom za dohvat datoteka, a to će nam u ovom radu biti potrebno jer želimo „zavarati“ sustav da se naši dinamički generirani kôd nalazi u datotekama pogleda, jer tako ne ometamo izvršavanje centralnog kôda sustava. Novi sustav za dohvaćanje datoteka ćemo registrirati u StartUp.cs datoteci koja određuje postupak pri pokretanju aplikacije .NET Core sustava. A zatim ćemo za svaki zahtjev koji stigne poslužitelju pregledati traženu adresu te provjeriti da li je zatražena dinamička stranica i ukoliko je, generirati ćemo kôd koji nam treba i dostaviti ga MVC-u kao da je došao iz obične datoteke.

```
Startup
public void ConfigureServices(IServiceCollection services)
    services.Configure<RazorViewEngineOptions>(opts =>
        opts.FileProviders.Add(
            new DynamicFileProvider()
        )
    );
```

Slika 2 - Registracija poslužitelja datoteka

[5] Razor syntax reference for ASP.NET Core - docs.microsoft.com
[6] Adam Freeman; Pro ASP.NET Core MVC(6th edition); ISBN-13: 978-1484203989, ISBN-10: 1484203984

```
public class DynamicFileProvider : IFileProvider
{
    public IFileInfo GetFileInfo(string subpath)
    {
        if (!subpath.Equals("/Views/Dynamic/Grid.cshtml", StringComparison.InvariantCultureIgnoreCase)
            && !subpath.Equals("/Views/Dynamic/Edit.cshtml", StringComparison.InvariantCultureIgnoreCase)
            && !subpath.Equals("/Views/Dynamic/Select.cshtml", StringComparison.InvariantCultureIgnoreCase))
        {
            return new NotFoundFileInfo(subpath);
        }
        var result = new DynamicFileInfo(subpath);
        return result.Exists ? result as IFileInfo : new NotFoundFileInfo(subpath);
```

Slika 3 – Kôd koji određuje radi li se o zahtjevu za dinamičnom stranicom

```
public class DynamicFileInfo : IFileInfo
{
    public DynamicFileInfo(string viewPath)
    {
        _viewPath = viewPath;
        GetView(viewPath);
    }

    private void GetView(string viewPath)
    {
        try
        {
            Type currentClass = GetCurrentClassType();
            DynamicViews dynamicViews = GetDynamicViewsObject(currentClass);

            _exists = true;
            string nameOfFile = Path.GetFileName(_viewPath).ToLower();
            switch (nameOfFile)
            {
                case "select.cshtml":
                    _viewContent = Encoding.UTF8.GetBytes(dynamicViews.Select);
                    break;
                case "edit.cshtml":
                    _viewContent = Encoding.UTF8.GetBytes(dynamicViews.Edit);
                    break;
                default:
                    _viewContent = Encoding.UTF8.GetBytes(dynamicViews.Grid);
                    break;
```

Slika 4 Generiranje kôda, te slanje određenog dijela ovisno o zahtjevu

5.4. Korišteni atributi

Svi atributi imaju mehanizam postavljanja zadanih vrijednosti u slučaju nepostojanja atributa, pa će tako na primjer u slučaju odsustva atributa [ID], property koji sadrži riječ ID u nazivu biti ID svojstvo. Stoga nije potrebno eksplicitno implementirati sve atribute i njihove vrijednosti ukoliko koristimo zadane vrijednosti. Svi atributi koji su sastavni dio

MVC-a a koji služe za validaciju su podržani(*Required*, *MaxLength* itd.). Navedeni atributi sa vlastitim svojstvima u listi će imati primjer korištenja neposredno ispod.

4.3.1 Atributi razreda

Ovi atributi se odnose na razred koji implementira *IDynamicPage* sučelje. Implementiraju se jedan put po razredu.

1. **Serializable** – Atribut ugrađen u .Net Core koji označava da se klasa može serijalizirati. Obavezan na svakom dinamičkom razredu
2. **AdditionalStatics –** Pomoću ovog atributa je moguće odrediti dodatne statičke datoteke koje će se uključiti u generirani kôd dinamičke web stranice. Te datoteke mogu biti CSS ili JS

```
[AdditionalStatics (
   "~/Content/Webpages/KompleksniUnos.css",
   "~/Scripts/Webpages/KompleksniUnos.js")]
    )]
```

Kôd 2 Primjer atributa za dodavanje dodatnih statičkih datoteka

3. **Grid** – Atribut koji određuje postavke tablice na razini klase:
   a. **SelectionMode** – Enumeracija tipa *GridSelectionMode* koja služi za odabir onoga što će se dogoditi pri odabiru stavke na pregledu svih unosa na tablici. Moguće opcije su: onemogući odabir(*None*), odaberi jednu(*Single*) i omogući višestruki odabir(*Multiple*)
   b. **ShowGeneralSearch** – Prikazuje polje za pretraživanje cijele tablice
   c. **DoubleClickAction** – Enumeracija tipa *GridDoubleClickAction* koja određuje što će se dogoditi pri dvostrukom kliku na redak u tablici. Opcije su: *Select* – otvori prozor za prikaz, *Edit* – otvori prozor za uređivanje unosa i *None* – ne čini ništa
   d. **ColumnChooser**– Omogućuje promjenu prikazanih stupaca tj. omogućuje korisniku da sakriva i prikazuje pojedine stupce

```
[Grid(
    SelectionMode = GridSelectionMode.None,
    ColumnChooser = true,
    DoubleClickAction = GridDoubleClickAction.Select,
```

```
    ShowGeneralSearch = true
    )]
```

Kôd 3 Primjer atributa koji određuje izgled tablice

4. **GeneratorOptions** – Atribut koji opisuje mnoge opcije generatora:
    a. **GenerateHTMLSkeleton –** Opisuje da li se treba generirati puni html okvir za sve ekrane. Označava se ako će forme za uređivanje i prikaz biti na posebnim stranicama jer će im tada trebati potpuni html okvir za normalno funkcioniranje
    b. **Data_Binding –** Enumeracija tipa *DataBinding*. Označava tip vezanja svojstava među objektima
    c. **Form_Template –** Enumeracija tipa *FormTemplate*. Određuje format formi za pregled i uređivanje
    d. **Grid_Type –** Enumeracija tipa *GridType* koja određuje tip tablice koja će se generirati. Dinamička stranica može koristiti razne komponente za izradu tablice
    e. **Form_FieldType –** Određuje koja će se tehnologija koristiti za generiranje polja u formi za uređivanje. Sustav podržava i jednostavna HTML polja te napredne komponente kao *DevExtreme*
    f. **PopupViews –** Određuje da li će se koristiti *pop-up* dijalozi za uređivanje i pregled.
    g. **PopUp_Type –** Određuje tehnologiju koja će se koristi za generiranje pop-up prozora. Možemo koristiti obični *Bootstrap* modalni dijalog te napredne komponente za stvaranje prozora na web aplikacijama
    h. **ClassNameStringKey -** Služi za definiranje ključa u resursima koji će se koristiti za prikazivanje naslova dinamičke stranice

```
[GeneratorOptions(
        GenerateHTMLSkeleton = true,
        ClassNameStringKey = "Kompleksni unos",
        Data_Binding = DataBinding.MVC,
        Form_FieldType = FormFieldType.DevExpress,
        Form_Template = FormTemplate.BootstrapCards,
        Grid_Type = GridType.DevExpress,
        PopupViews = true,
        PopUp_Type = PopupType.BootstrapModal
        )]
```

Kôd 4 Primjer atributa koji određuje funkcionalnosti generatora dinamičke stranice

5. **SqlProcedures** –U ovom atributu se definiraju imena procedura za komunikaciju s bazom:
    a. **GetAll –** Ime procedure koja dohvaća sve unose u tablici dinamičke stranice

b. **Get** – Ime procedure koja dohvaća jedan unos iz tablice po ID-u. Nije obvezna. Ukoliko ovo svojstvo nije definirano šifrarnik će spremati listu unosa u sesiju i dobavljati pojedinačne unose iz iste
c. **Insert**– Ime procedure koja sprema novi unos u bazu
d. **Update**– Ime procedure koja ažurira unos u bazi
e. **Delete**– Ime procedure koja briše unos iz baze

```
[SqlProcedures(
    GetAll = "dbo.DobijKompleksne",
    Get = "dbo.DobijKompleksni",
    Insert = "dbo.DodajNoviKompleksni",
    Update = "dbo.UpdateKompleksni",
    Delete = "dbo.IzbrisiKompleksni"
       )]
```

Kôd 5 Primjer atributa koji određuje SQL procedure za rad s podacima

6. **ViewOnly** – Atribut koji označava da se radi o sistemskom setu podataka tj. onemogućava uređivanje i brisanje unosa

4.4.2 Atributi svojstava

Ovo su atributi koji se vežu na pojedina svojstva dinamičkog razreda. Svrha mnogih će biti ispunjena po zadanim postavkama čak i ako nisu prisutni.

1. **ID** – Služi za definiranje svojstva kao primarni ključ koji će se koristiti za identifikaciju unosa
2. **KeyString** – Služi za definiranje ključa u resursima koji će se koristiti za prikazivanje imena svojstva. Ukoliko ključ ne postoji ključ će se prikazati kao prevedena riječ/fraza.

```
[KeyString("Naziv radnog mjesta")]
public string PositionName { get; set; }
```

Kôd 6 Primjer atributa za označavanje imena ili ključa u resursima koji će biti korišten za ime polja svojstva

3. **GridColumn** – Služi za upravljanje prikazom svojstava na tablici. Ako niti jedno svojstvo nema ovaj atribut onda će se svi stupci prikazati u tablici, no ako postoji barem jedan ovaj atribut u razredu onda će se prikazivati samo ona svojstva koja imaju ovaj atribut definiran. Moguće je dodati prazan *GridColumn* atribut u kojem slučaju će se koristiti zadane postavke kao da isti ne postoji, međutim isti će naglasiti

da njegovo svojstvo treba biti prikazano na tablici. Atribut koji opisuje postavke svojstva u okviru tablice:

a. **Index –** Opisuje redoslijed svojstava u tablici. Svojstva bez ovog polja na atributu će se izredati nakon onih s definiranim indeksom po redoslijedu definiranja na razredu.
b. **CaptionKey –** Definira ključ u resursima koji služi za prikazivanje imena svojstava na tablici
c. **Format –** Enumeracija *GridColumnFormat* koja služi za određivanje oblika podatka u tablici(npr. prikaz datuma, valute itd.)
d. **Alignment –** Enumeracija tipa *GridHorizontalAlignment* koja služi za određivanje poravnanja u tablici
e. **GroupByMe –** Ako je *True* onda će se tablica grupirati po tom svojstvu

```
[GridColumn(
     Alignment = GridHorizontalAlignment.Left,
     CaptionKey = "Pos.Nm",
     Format = GridColumnFormat.LargeNumber,
     GroupByMe = false,
     Index = 1
            )]
public string PositionName { get; set; }
```

Kôd 7 Primjer atributa koji označava ulogu svojstva u tablici

4. **SqlParameter –** Definira ime parametra koje se koristi za označavanje svojstva u pozivima procedura na bazi. Npr. svojstvo se može zvati „Opis“, ali se u pozivima na bazu parametar zove „*Description*“. Ukoliko svojstvo nema ovaj atribut, pretpostavlja se da će se parametar zvati isto kao i svojstvo. Moguće je definirati i druge opcije vezane uz parametar koji predstavlja odabrani svojstvo

```
[SqlParameter("BrojRadnogMjesta")]
public int RedniBrojUnosa { get; set; }
```

Kôd 8 Primjer atributa koji određuje ime svojstva u bazi

5. **FormGroup –** Dopušta definiranje isključivo estetske grupe na formama za pregled i uređivanje. Atribut se koristi na dva načina: Na jednom svojstvu grupe se definira puni atribut(sva polja atributa), a zatim se, na ostalim članovima grupe, primjenjuje FormGroup atribut koji ima samo ime. Atribut ima 3 svojstva:
    - **GroupName** – Ime grupe, odabire se proizvoljno i mora biti isto na svim članovima grupe. Mora biti definirano na svakom FromGroup atributu

- **Accent** – Određuje izgled grupe. Može biti tradicionalni html fieldset ili samo estetska opcija. Može se definirati samo jedan put po grupi
- **SpecialColumn** – Određuje da li da se definira specijalni bootstrap stupac za ovu grupu. Može se definirati samo jedanput po grupi.

```
[FormGroup("Osnovne Informacije", FormGroupAccent.GlassPane, false)]
public string PositionName { get; set; }

[FormGroup("Osnovne Informacije")]
public bool IsManager { get; set; } = false;
```

Kôd 9 Primjer atributa za grupiranje svojstava

6. **FieldType –** Definira koji tip polja će se koristiti za uređivanje ovog svojstva na prikladnoj formi. Ovaj atribut obično nije potreban jer će generator sam odabrati prikladan tip polja ovisno o tipu svojstva. Npr. svojstvo tipa *string* će po zadanom biti neki tip *TextBox*-a, *Integer* će biti *NumberBox*, *Bool* će biti *CheckBox* itd. Atribut se najčešće koristi za definiranje tipova polja koji nisu jasni iz tipa svojstva. Npr. Definiranje svojstva tipa *String* kao *TextArea* umjesto *TextBox*-a

```
[FieldType(DataFieldTypes.TextArea)]
public string PositionName { get; set; }
```

Kôd 10 Primjer atributa koji određuje tip polja u kojem će svojstvo biti prikazano

7. **Uneditable –** Određuje da se ovo polje neće pojavljivati na formi za uređivanje. Prikladno za npr. polje koje sadržava datum unosa te identifikacijski broj odgovornog korisnika za sigurnosne svrhe
8. **HiddenInput –** Dio *AspNetCore.Mvc* paketa. Prilikom korištenja u generatoru, označeno polje sakrije sa svih formi i tablice.
9. **RadioButtonOption –** Ukoliko želimo da svojstvo bude prikazan u obliku radio dugmeta-a, dodamo mu *RadioButtonOption* opcije. Svojstvo će se prikazivati u obliku radio dugmadi ukoliko ima barem jedan atribut tipa *RadioButtonOption*, a svaki pojedini atribut predstavlja jednu opciju. Svaki atribut ima 2 polja:
   - **Value** – Vrijednost koja će biti korištena(može biti tipa *int* ili *string*)
   - **Text** – Tekst koji će biti prikazan za tu vrijednost

```
[RadioButtonOption("0", "Nikada")]
[RadioButtonOption("1", "Petkom")]
[RadioButtonOption("2", "Vikendom")]
public int Benefiti { get; set; }
```

Kôd 11 Primjer atributa za definiranje opcija u prikladnom polju

10. **DynamicDDL –** Označava da ovo svojstvo predstavlja vrijednost u listi vrijednosti i ključeva, te da će biti prikazan kao padajuća lista. Ove vrijednosti se u tablici automatski prevode u svoj pripadajući tekst. Vrijednosti se učitavaju iz baze kao lista *SelectListItem* objekata okidanjem funkcije na razredu s navedenim potpisom. Ovu funkciju je potrebno implementirati na razredu za svako svojstvo označeno *DynamicDDL* atributom.

```
public List<Microsoft.AspNetCore.Mvc.Rendering.SelectListItem> GetDDL[Ime
svojstva bez zagrada]()
```

Kôd 12 Primjer funkcije potrebne za definiranje izvora podataka za padajuću listu

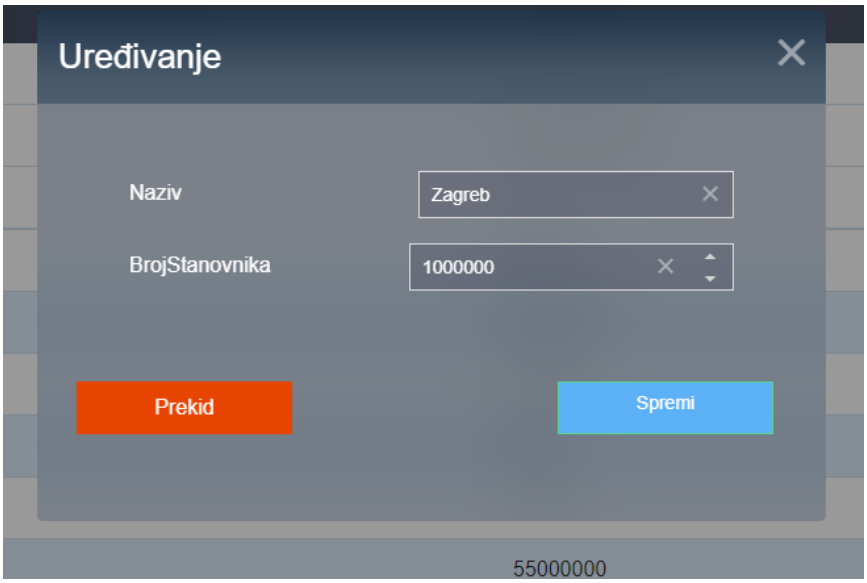
6. Primjeri dinamičkih stranica

6.1. Jednostavni primjer

U ovom primjeru ćemo proučiti naj jednostavniji mogući primjer dinamične stranice. Podatkovni set koji će ona obrađivati ima samo 2 polja.

```
    [Serializable]
    [GeneratorOptions(ClassNameStringKey ="Država")]
    public class SimpleEntry : DynamicPage, IDynamicPage
    {
        public int Id { get; set; }
        public string Naziv { get; set; }
        public int BrojStanovnika { get; set; }

    }
```

Kôd 13 Kôd jednostavnog primjera dinamičke stranice



Slika 5 Izgled forme za uređivanje jednostavnog unosa

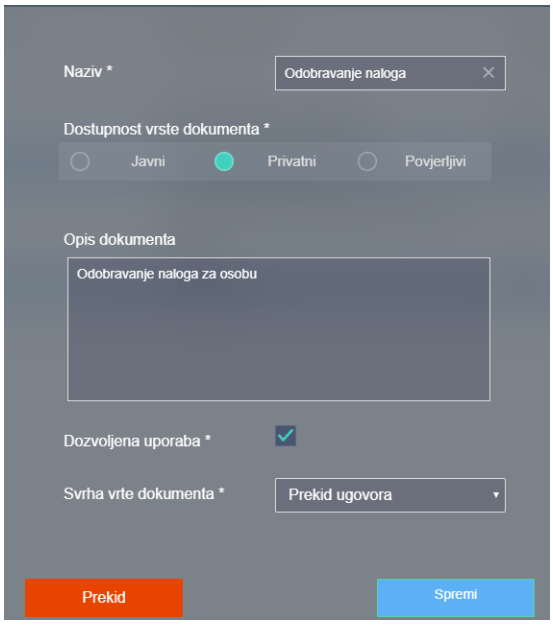
6.2. Realistični primjer

Ovaj primjer predstavlja realistični podatkovni set, te neke napredne funkcionalnosti našeg sustava.

```
[Serializable]
[GeneratorOptions(ClassNameStringKey ="Vrsta dokumenta")]
[Grid(GridSelectionMode.None, true, GridDoubleClickAction.Edit, true)]
public class NormalForm : DynamicPage, IDynamicPage
{
    [ID]
    public int Id { get; set; }
```

```
    [KeyString("Naziv")]
    [Required(AllowEmptyStrings =false, ErrorMessage = "Polje 'Naziv' je bavezno!")]
    public string Naziv { get; set; }

    [KeyString("Dostupnost vrste dokumenta")]
    [RadioButtonOption("1", "Javni")]
    [RadioButtonOption("2", "Privatni")]
    [RadioButtonOption("3", "Povjerljivi")]
    [Required(AllowEmptyStrings = false, ErrorMessage = "Polje 'Dostupnost vrste
dokumenta' je obavezno!")]
    public int DostupnostDokumenta { get; set; }

    [KeyString("Opis dokumenta")]
    [FieldType(DataFieldTypes.TextArea)]
    public string Opis { get; set; }

    [KeyString("Dozvoljena uporaba")]
    [Required(AllowEmptyStrings = false, ErrorMessage = "Polje 'Dozvoljena uporaba'
je obavezno!")]
    public bool Dozvoljen { get; set; }

    [KeyString("Svrha vrte dokumenta")]
    [DynamicDDL]
    [Required(AllowEmptyStrings = false, ErrorMessage = "Polje 'Svrha Dokumenta' je
obavezno!")]
    public int SvrhaDokumenta { get; set; }

    public List<Microsoft.AspNetCore.Mvc.Rendering.SelectListItem>
GetDDLSvrhaDokumenta()
        {
            SqlCommandWrapper sqlCommandWrapper = new
SqlCommandWrapper("dbo.SvrhaDokumentaGetAll");
            return sqlCommandWrapper.ExecuteSelectList("Id", "Naziv");

        }
```

Kôd 14 Prikazuje realnu implementaciju sustava



Slika 6 Prikazuje izgled forme za uređivanje realističnog podatkovnog seta

6.3. Kompleksni primjer

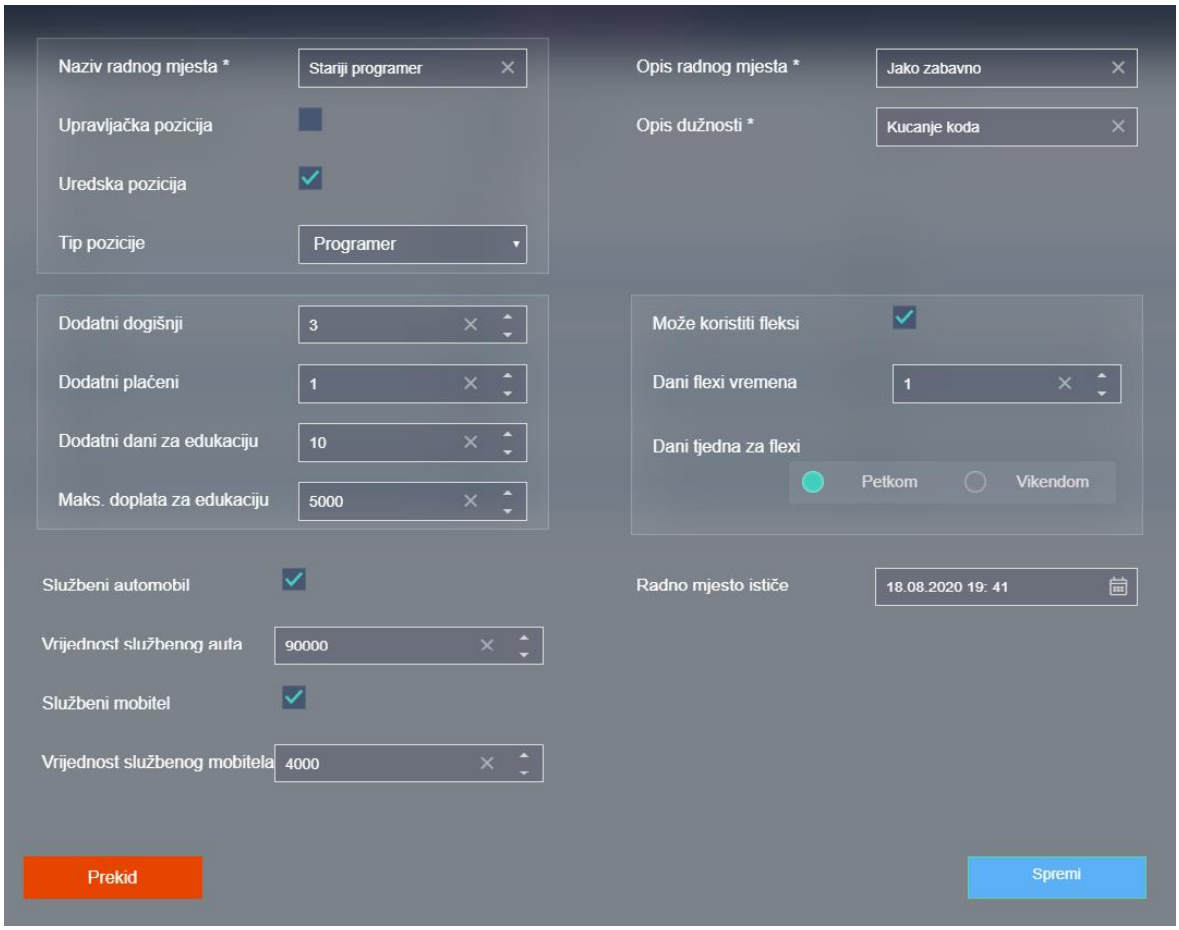
Ovaj primjer prikazuje sve sposobnosti proširivosti i prilagodljivosti sustava.

```
[SqlProcedures(
    GetAll = "dbo.DobijKompleksne",
    Get = "dbo.DobijKompleksni",
    Insert = "dbo.DodajNoviKompleksni",
    Update = "dbo.UpdateKompleksni",
    Delete = "dbo.IzbrisiKompleksni")]
    [AdditionalStatics(
    "~/Content/Webpages/KompleksniUnos.css",
    "~/Scripts/Webpages/KompleksniUnos.js")]
    [GeneratorOptions(ClassNameStringKey = "Radno mjesto")]
    [Grid(
        SelectionMode = GridSelectionMode.None,
        ColumnChooser = true,
        DoubleClickAction = GridDoubleClickAction.Select,
        ShowGeneralSearch = true
        )]
    [Serializable]
    public class ComplexForm : DynamicPage, IDynamicPage
    {
        [ID]
        [SqlParameter("BrojRadnogMjesta")]
        [HiddenInput]
        public int RedniBrojUnosa { get; set; }

        [KeyString("Naziv radnog mjesta")]
        [Required(AllowEmptyStrings = false, ErrorMessage = "Radno mjesto mora imati
Naziv!")]
        [FormGroup("Osnovne Informacije", FormGroupAccent.GlassPane, false)]
        [GridColumn(
            Alignment = GridHorizontalAlignment.Left,
            CaptionKey = "Pos.Nm",
            Format = GridColumnFormat.LargeNumber,
            GroupByMe = false,
            Index = 1
            )]
        public string PositionName { get; set; }

        [FormGroup("Osnovne Informacije")]
        [KeyString("Upravljačka pozicija")]
        [GridColumn(Index = 2)]
        public bool IsManager { get; set; } = false;

        [FormGroup("Osnovne Informacije")]
        [KeyString("Uredska pozicija")]
        [GridColumn(Index = 3)]
        public bool OfficeJob { get; set; } = true;

        [FormGroup("Osnovne Informacije")]
        [KeyString("Tip pozicije")]
        [GridColumn(Index = 4)]
        [DynamicDDL]
        public int PostionType { get; set; }

        [KeyString("Opis radnog mjesta")]
        [Required(AllowEmptyStrings = false, ErrorMessage = "Radno mjesto mora imati
opis!")]
        public string OpisRadnogMjesta { get; set; }
```

```
        [KeyString("Opis dužnosti")]
        [Required(AllowEmptyStrings = false, ErrorMessage = "Radno mjesto mora imati
opis dužnosti!")]
        public string OpisDuznosti { get; set; }

        [KeyString("Dodatni dogišnji")]
        [FormGroup("Benefiti", FormGroupAccent.GlassPane, false)]
        public int AdditionalVacation { get; set; } = 0;

        [KeyString("Dodatni plaćeni")]
        [FormGroup("Benefiti")]
        public int AdditionalPaidLeave { get; set; } = 0;

        [KeyString("Dodatni dani za edukaciju")]
        [FormGroup("Benefiti")]
        public int AdditionalEducation { get; set; } = 0;

        [KeyString("Maks. doplata za edukaciju")]
        [FormGroup("Benefiti")]
        [FieldType(DataFieldTypes.Decimal)]
        public double EducationBudget { get; set; } = 0;

        [KeyString("Može koristiti fleksi")]
        [FormGroup("Fleksibilno vrijeme", FormGroupAccent.GlassPane, true)]
        public bool HasFlexiTime { get; set; } = false;

        [KeyString("Dani flexi vremena")]
        [FormGroup("Fleksibilno vrijeme")]
        public int FlexiDays { get; set; }

        [KeyString("Dani tjedna za flexi")]
        [RadioButtonOption("1", "Petkom")]
        [RadioButtonOption("2", "Vikendom")]
        [FormGroup("Fleksibilno vrijeme")]
        public int DaniZaFlexi { get; set; }

        [KeyString("Službeni automobil")]
        public bool ImaAuto { get; set; }

        [KeyString("Vrijednost službenog auta")]
        public double VrijednostAuto { get; set; }

        [KeyString("Službeni mobitel")]
        public bool ImaMobitel { get; set; }

        [KeyString("Vrijednost službenog mobitela")]
        public double VrijednostMobitel { get; set; }

        [KeyString("Radno mjesto ističe")]
        [FieldType(DataFieldTypes.DateTime)]
        [GridColumn(Index = 5)]
        public DateTime PositionExpires { get; set; } = DateTime.Now;

        [Uneditable]
        [FieldType(DataFieldTypes.Date)]
        [KeyString("Radno mjesto stvoreno")]
        public DateTime Created { get; set; } = DateTime.Now;

        [HiddenInput]
        public DateTime Modified { get; set; } = DateTime.Now;
```

```
        public List<Microsoft.AspNetCore.Mvc.Rendering.SelectListItem>
GetDDLPostionType()
        {
            SqlCommandWrapper sqlCommandWrapper = new
SqlCommandWrapper("dbo.TipoviRadnogMjestaGetAll");
            return sqlCommandWrapper.ExecuteSelectList("id", "ime");
        }




        public new object GetObject(IHandleErrors dynamicController, string id) {
            SqlCommandWrapper sqlCommandWrapper =
DynamicRepoHelper.GetWrapper(dynamicController, GetType(), CRUD.Get,
StandardParameters());
            DynamicRepoHelper.AttachIdPropertyAsParam(sqlCommandWrapper, GetType(),
id);
            return sqlCommandWrapper.ExecuteObject<ComplexForm>();
        }

        public new bool InsertObject(IHandleErrors dynamicController, object obj)
        {
            Created = DateTime.Now;
            Modified = DateTime.Now;

            SqlCommandWrapper sqlCommandWrapper =
DynamicRepoHelper.GetWrapper(dynamicController, GetType(), CRUD.Insert, null);
            DynamicRepoHelper.AttachAllPropertiesAsParams(sqlCommandWrapper, obj,
true);
            DynamicRepoHelper.AttachIdPropertyAsOutputParam(sqlCommandWrapper,
GetType());
            return sqlCommandWrapper.ExecuteNonQuery() == 0 ? false : true;
        }

        public new bool UpdateObject(IHandleErrors dynamicController, object obj)
        {
            Modified = DateTime.Now;

            SqlCommandWrapper sqlCommandWrapper =
DynamicRepoHelper.GetWrapper(dynamicController, GetType(), CRUD.Update, null);
            DynamicRepoHelper.AttachAllPropertiesAsParams(sqlCommandWrapper, obj,
false);
            return sqlCommandWrapper.ExecuteNonQuery() == 0 ? false : true;
        }


    }
```

Kôd 15 Prikazuje implementaciju dinamičke stranice za kompleksni set podataka



Slika 7 Izgled forme za uređivanje kompleksnog podatkovnog seta

Zaključak

Tema ovog rada je izvedba sustava za automatsko generiranje sučelja u web aplikacijama. Izvedbu smo postigli koristeći .NET Core platformu i MVC. Sustav ima značajne mogućnosti proširenja i prilagodljivosti koje nadilaze slične postojeće sustave na tržištu. Među glavnim značajkama prilagodljivosti sustava jest mogućnost integracije sa postojećim podatkovnim okruženjem.

Glavne prednosti implementiranog sustava su centralizacija kôda, standardizacija metoda za upravljanje podatkovnim setovima u aplikaciji te značajno ubrzanje razvoja CRUD formi.

Naša, kao i svaka, centralizacija koda ima svoju manu, a to je da stvara jedinstvenu točku kvara(eng. *Single Point Of Failure*). Ukoliko se dogodi greška na kodu koji generira stanice, niti jedna stranica koja koristi ovaj sustav neće funkcionirati ispravno. To je ujedno i glavna mana ovakvog sustava.

Popis kratica

MVC	*Model View Controller*	Arhitektura web aplikacija
MS	*Microsoft*	Tvrtka koja stoji iza Windows OS-a
DLL	*Dynamic Linking Library*	Tip biblioteke za računala
HTML	*Hyper Text Markup Language*	Programski jezik za web preglednike
SQL	*Structured Query Language*	Upitni jezik

Popis slika

Slika 1 Primjer Razor sintakse 9

Slika 2 Registracija poslužitelja datoteka 9

Slika 3 Kôd koji određuje radi li se o zahtjevu za dinamičnom stranicom 10

Slika 4 Generiranje kôda, te slanje određenog dijela ovisno o zahtjevu 10

Slika 5 Izgled forme za uređivanje jednostavnog unosa 17

Slika 6 Prikazuje izgled forme za uređivanje realističnog podatkovnog seta 18

Slika 7 Izgled forme za uređivanje kompleksnog podatkovnog seta 22

Popis tablica

Tablica 1 - Razlike .NET i .NET Core sustava 8

Popis kôdova

Kôd 1 Primjer dohvata atributa svojstava refleksijom 6

Kôd 2 Primjer atributa za dodavanje dodatnih statičkih datoteka 11

Kôd 3 Primjer atributa koji određuje izgled tablice 12

Kôd 4 Primjer atributa koji određuje funkcionalnosti generatora dinamičke stranice 12

Kôd 5 Primjer atributa koji određuje SQL procedure za rad s podacima..... 13

Kôd 6 Primjer atributa za označavanje imena ili ključa u resursima koji će biti korišten za ime polja svojstva 13

Kôd 7 Primjer atributa koji označava ulogu svojstva u tablici..... 14

Kôd 8 Primjer atributa koji određuje ime svojstva u bazi 14

Kôd 9 Primjer atributa za grupiranje svojstava 15

Kôd 10 Primjer atributa koji određuje tip polja u kojem će svojstvo biti prikazano..... 15

Kôd 11 Primjer atributa za definiranje opcija u prikladnom polju..... 16

Kôd 12 Primjer funkcije potrebne za definiranje izvora podataka za padajuću listu 16

Kôd 13 Kôd jednostavnog primjera dinamičke stranice 17

Kôd 14 Prikazuje realnu implementaciju sustava..... 18

Kôd 15 Prikazuje implementaciju dinamičke stranice za kompleksni set podataka 21

Literatura

[1] ADAM FREEMAN; PRO ASP.NET CORE MVC(6TH EDITION); ISBN-13: 978-1484203989, ISBN-10: 1484203984

[2] JOHN SHARP; MICROSOFT VISUAL C# 2012 STEP BY STEP; ISBN-13: 978-0735668010, ISBN-10: 9780735668010

[3] MAJA ČIĆ, UVOD U PROGRAMIRANJE, FESB

[4] RAZOR SYNTAX REFERENCE FOR ASP.NET CORE - DOCS.MICROSOFT.COM.

Prilog

Uz ovaj rad priložena je aplikacija koja u praksi prikazuje rad sustava za dinamičko generiranje web stranica.

Preduvjeti za pokretanje aplikacija su .NET Core Runtime minimalne verzije 2.2. te baza podataka koja može biti rekonstruirana iz priložene .bak datoteke. Također, potrebna je valjana DevExtreme licenca.

ALGEBRA
VISOKO UČILIŠTE

SUSTAV ZA AUTOMATSKO GENERIRANJE SUČELJA U WEB APLIKACIJAMA

Pristupnik: Filip Bebić, 0036489099

Mentor: prof. Danijel Kučak