Implementacija pristupa bazi podataka i povezivanje modela sa pravom bazom

MVC aplikaciju koju smo kreirali povezali smo sa statičkim podacima kreiranim u okviru jednog od kontrolera koji smo upotrebljavali. Ukoliko bi želeli da povežemo našu MVC aplikaciju sa bazom podataka, koju smo već kreirali i smestili na SQL Server (baza *Skola* koja sadrži samo jednu tabelu *Studenti*) potrebno je napraviti neke promene koda.

Najpre moramo instalirati *Entity Framework.dll* biblioteku, jer nam ona automatski obezbeđuje konekciju do baze i omogućuje da rad sa bazom bude što više nezavisan od same baze. Tu biblioteku u okviru projekta, u Visual Studiju, možemo instalirati koristeći opciju ***Tools* > *NuGet Package Manager* > *Manage NuGet Packages for Solution...***

Po instalaciji *Entity Framework.dll* biblioteke, potrebno je popraviti kontroler *StudentController.cs* i njegov pogled.

U kontroleru *StudentController.cs* dodajemo metodu (*StudentiContext \_context)* koja ga povezuje sa klasom *StudentiContext.cs,* koju ćemo formirati u folderu Models*.* To je *DbContext* klasa koja sadrži generički tip *DbSet<Studenti>* za uzimanje podataka iz formirane tabele u bazi podataka. Osim ove dodajemo još GET i POST metode za ažuriranje podataka (CRUD operacije).

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Skola.Models;

namespace Skola.Controllers

{

public class StudentController : Controller

{

private readonly StudentiContext \_context;

public StudentController(StudentiContext context)

{

\_context = context;

}

// GET: Student

public async Task<IActionResult> Index()

{

return View(await \_context.Studenti.ToListAsync());

}

**StudentController.cs**

// GET: Student/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var studenti = await \_context.Studenti

.FirstOrDefaultAsync(m => m.Id == id);

if (studenti == null)

{

return NotFound();

}

return View(studenti);

}

// GET: Student/Create

public IActionResult Create()

{

return View();

}

// POST: Student/Create

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Ime,Prezime,JMBG,MestoRođenja")] Studenti studenti)

{

if (ModelState.IsValid)

{

\_context.Add(studenti);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(studenti);

}

// GET: Student/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var studenti = await \_context.Studenti.FindAsync(id);

if (studenti == null)

{

return NotFound();

}

return View(studenti);

}

// POST: Student/Edit/5

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Ime,Prezime,JMBG,MestoRođenja")] Studenti studenti)

{

if (id != studenti.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(studenti);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!StudentiExists(studenti.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(studenti);

}

// GET: Student/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var studenti = await \_context.Studenti

.FirstOrDefaultAsync(m => m.Id == id);

if (studenti == null)

{

return NotFound();

}

return View(studenti);

}

// POST: Student/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var studenti = await \_context.Studenti.FindAsync(id);

\_context.Studenti.Remove(studenti);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool StudentiExists(int id)

{

return \_context.Studenti.Any(e => e.Id == id);

}

}

}

**StudentiContext.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

namespace Skola.Models

{

public class StudentiContext : DbContext

{

public StudentiContext (DbContextOptions<StudentiContext> options)

: base(options)

{

}

public DbSet<Skola.Models.Studenti> Studenti { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

if (!optionsBuilder.IsConfigured)

{

#warning To protect potentially sensitive information in your connection string, you should move it out of source code. See http://go.microsoft.com/fwlink/?LinkId=723263 for guidance on storing connection strings.

optionsBuilder.UseSqlServer("Server=DESKTOP-013A96I\\SQLEXPRESS;Database=Skola;Trusted\_Connection=True;");

}

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.HasAnnotation("ProductVersion", "2.2.6-servicing-10079");

modelBuilder.Entity<Studenti>(entity =>

{

entity.HasKey(e => e.Id);

entity.Property(e => e.Id).HasColumnName("Id");

entity.Property(e => e.JMBG).HasColumnName("JMBG");

entity.Property(e => e.Ime).HasColumnName("Ime").HasMaxLength(15);

entity.Property(e => e.MestoRođenja).HasColumnName("MestoRođenja").HasMaxLength(30);

entity.Property(e => e.Prezime).HasColumnName("Prezime").HasMaxLength(25);

});

}

}

}

Po sređivanju kontrolera prelazimo na sređivanje pogleda vezanog za taj kontroler u folderu Views/*Student,*  kao i kreiranju novog foldera *Shared* sa fajlom *Layout.cshtml.*

@model IEnumerable<Skola.Models.Studenti>

@{

ViewData["Title"] = "Index";

}

<meta charset=”utf-8″>

<meta http-equiv=”X-UA-Compatible” content=”IE =edge”>

<meta name=”viewport” content=”width=device-width, initial-scale=1″>

<link href=”~/lib/bootstrap/dist/css/bootstrap.min.css” rel=”stylesheet”>

<script src=”~/lib/jquery/dist/jquery.min.js”></script>

<script src=”~/lib/bootstrap/dist/js/bootstrap.min.js”></script>

<h1>Index</h1>

<p>

<a **asp-action**="Create">Create New</a>

</p>

<table class="table">

<thead class="thead-dark">

<tr>

<th>

@Html.DisplayNameFor(model => model.Id)

</th>

<th>

@Html.DisplayNameFor(model => model.Ime)

</th>

<th>

@Html.DisplayNameFor(model => model.Prezime)

</th>

<th>

@Html.DisplayNameFor(model => model.JMBG)

</th>

<th>

@Html.DisplayNameFor(model => model.MestoRođenja)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Id)

</td>

<td>

@Html.DisplayFor(modelItem => item.Ime)

</td>

<td>

@Html.DisplayFor(modelItem => item.Prezime)

</td>

<td>

@Html.DisplayFor(modelItem => item.JMBG)

</td>

<td>

@Html.DisplayFor(modelItem => item.MestoRođenja)

</td>

<td>

<a **asp-action**="Edit" **asp-route-id**="@item.Id">Edit</a> |

<a **asp-action**="Details" **asp-route-id**="@item.Id">Details</a> |

<a **asp-action**="Delete" **asp-route-id**="@item.Id">Delete</a>

</td>

</tr>

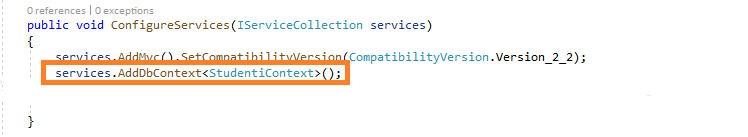
}

</tbody>

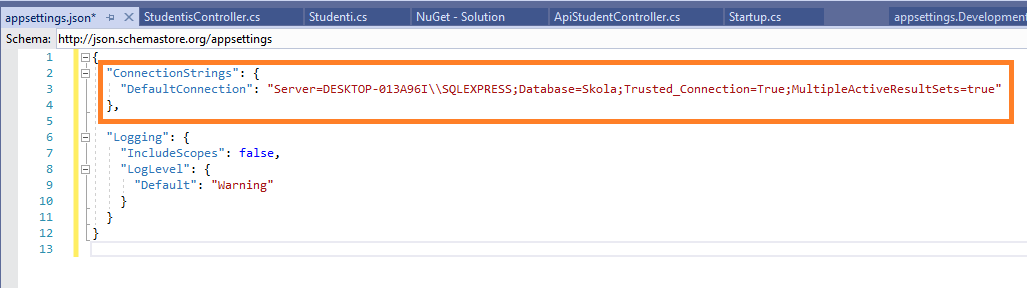
</table>

**Index.cshtml**

U folderu Student možemo dodati strane za ažuriranje podataka (Edit, Create, Delete, Details) i modifikovati *Index.cshtml* stranu, tako što podatke koji se unose u tabelu povežemo sa kreiranom bazom podataka.

Po sređivanju kontrolera i pogleda potrebno je u klasi *Startup.cs* dodati novi servis *AddDbContext<StudentiContext>()*.

Na kraju je potrebno podesiti string za konekciju. U fajlu *appsettings.json* dodaje se sledeća vrednost za konekcijski string.



Za *Server* izabiramo naš SQL Server na kojem smo smestili našu bazu, čije ime upisujemo na mesto *Database.*

Konačni izgled početne strane (*Index*) i novo formirane strane (*Create*) mogu se videti na slikama ispod.

