ProgInz TG08.3 Home

Site location: Home Page - DuckI

- Filip Belina - Prject Lead
- Jakov Lovaković - Backend Team Lead
- Jan Lalić - Backend Engineer
- Leo Marušić - DevOps/Database Engineer
- Mislav Marinović - Lead design
- Jan Badel - Frontend Engineer
- Martin Šainčević - Frontend engineer

- Frontend - Bootstrap + jQuery
- Backend - ASP .NET
- Database - SQLite
- Project Management - JIRA + Confluence

Dev Guidelines

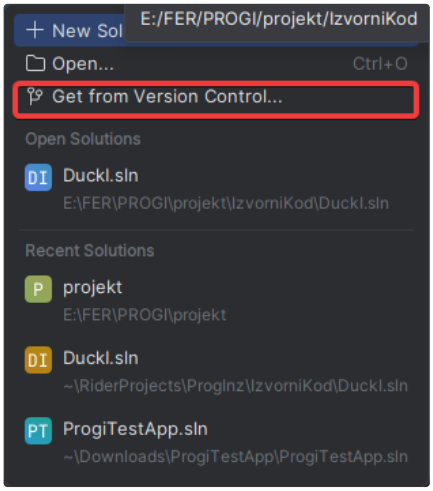
Git management

Ovaj page će opisati procese kojima koristimo github, ovaj prikaz govori o tome u sklopu Ridera, ali ideje su iste bez obzira na koji git managment tool bude odabran.

1. Kloniranje repozitorija

Ako repozitorij još nije kloniran na lokalno računalo:

- Otvori Rider i odaberi **Get from Version Control** s početnog zaslona.

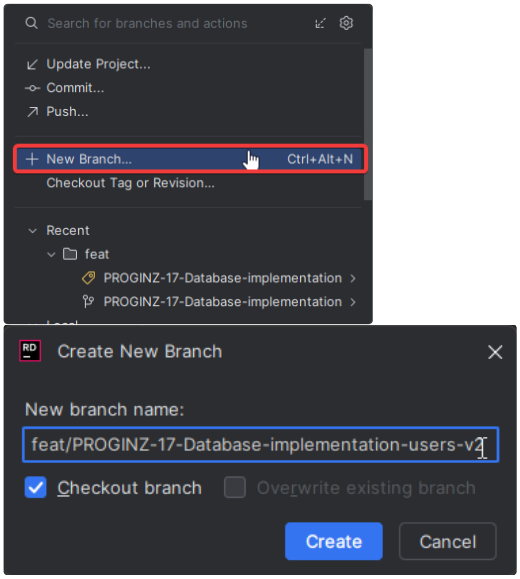


- Upiši URL GitHub repozitorija, odaberi lokalnu mapu za kloniranje i klikni na **Clone**.

2. Kreiranje nove grane iz *main* grane

Da bi kreirao novu granu:

- Provjeri da si na glavnoj (*main*) grani tako da u donjem desnom kutu Rider prozora vidiš naziv trenutne grane.
- Desnim klikom na trenutnu granu ili klikom na izbornik **Git** u alatnoj traci odaberi **New Branch...**.



- Unesi naziv nove grane, npr. `feat/TicketKey-TicketName`, i klikni **Create**. Rider će te automatski prebaciti na novu granu.
- Naziv grane uvijek mora biti jedan od 4:
  `feat/TicketKey-TicketName`,
  `bug/TicketKey-TicketName`,

`test/TicketKey-TicketName`,
`type/TicketName`

3. Dodavanje i commitanje promjena

Nakon što napraviš promjene u projektu:

- Otvori alatnu traku Git-a s **View > Tool Windows > Git** ili klikni na **Commit** ikonu u donjem dijelu zaslona.
- Označi promijenjene datoteke koje želiš dodati u commit ili označi sve promjene.
- Upiši opis promjena u polje za poruku.
- Klikni na **Commit** ili **Commit and Push** (ako želiš odmah poslati promjene na GitHub).

4. Slanje promjena na GitHub (push)

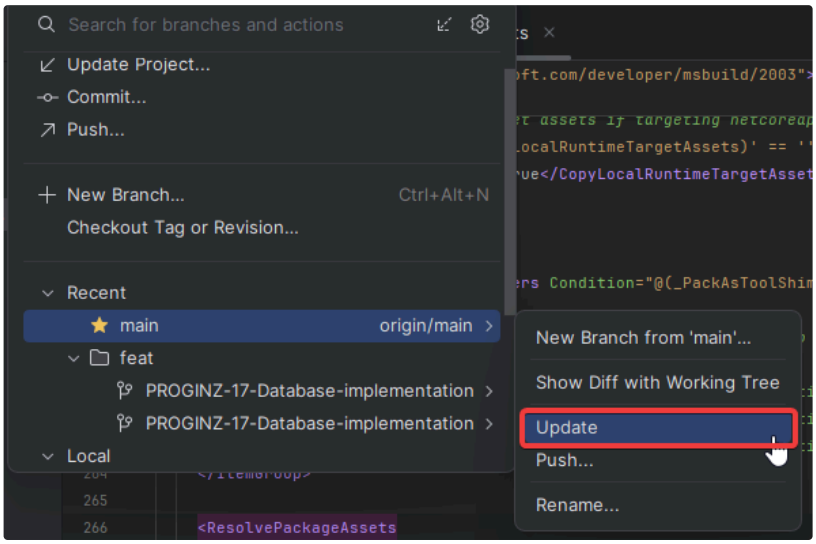
Ako si samo napravio commit i nisi još poslao promjene:

- Otvori **Git** alatnu traku.
- Klikni **Push** ili desni klik na granu i odaberi **Push**. Time ćeš poslati promjene na GitHub repozitorij.

5. Povlačenje najnovijih promjena s GitHub-a (pull)

Ako želiš povući najnovije promjene s *main* grane na GitHubu:

- Prebaci se na *main* granu kroz Git izbornik u donjem desnom kutu ili pomoću **Git** alatne trake.



- Klikni na **Fetch, Pull ili Update** kako bi povukao najnovije promjene u svoj lokalni repozitorij.

6. Spajanje promjena s *main* granom

Kad završiš rad na `nova-grana` i želiš je spojiti s *main*:

- Otvori Github te na projektu napravi novi pull request baziran na grani koju želiš spojiti u glavni projekt.
- Nakon pull requesta potrebno je čekati odobrenje 2 druga developera, ili dobiti explicitno dopuštenje za merganje od projekt leada
- Kada su svi uvijeti ispunjeni (dovoljan broj aproveova, bez ne razriješenih komentara, bez merge conflikata) dozvoljeno je dovršiti pr pritiskom na merge.

7. Nakon završetka na Jiri zatvoriti ticket

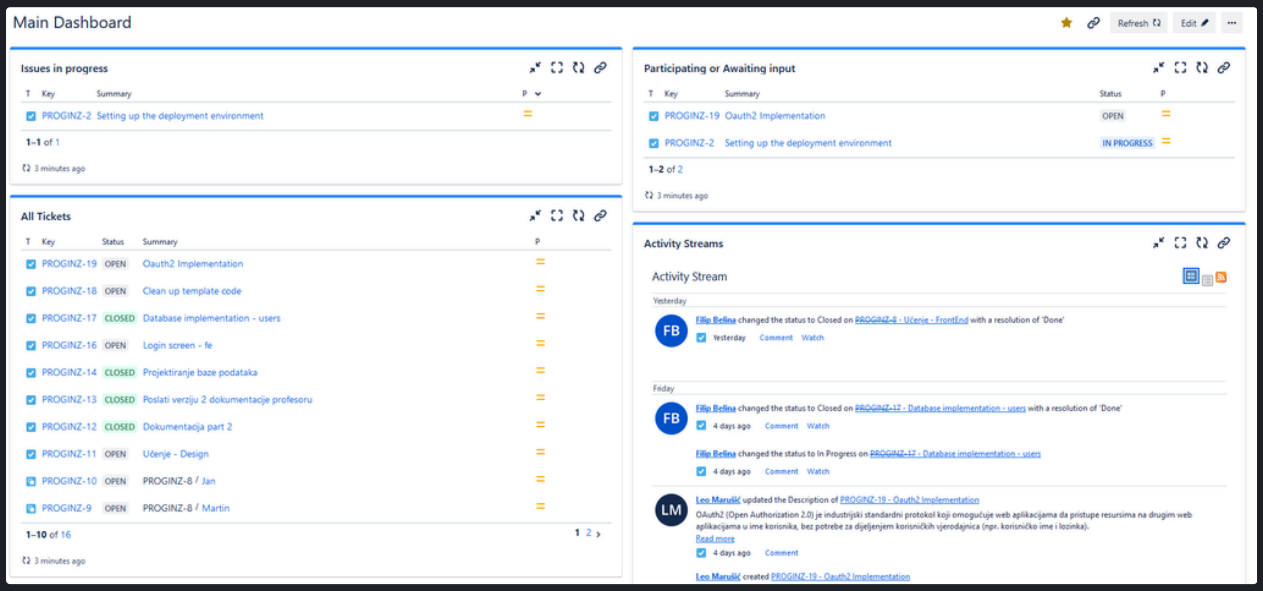
Potrebno je otiči na ticket i prebaciti njegov state na closed

JIRA management

Stranica za quick reference kako koristiti Jiru.

Main Dashboard

Kreiran je main dashboard koji odmah prikazuje informacije bitne za korisnika

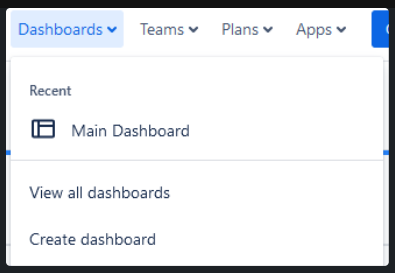


Dashboard je podijeljen na 4 dijela i prikazuje:

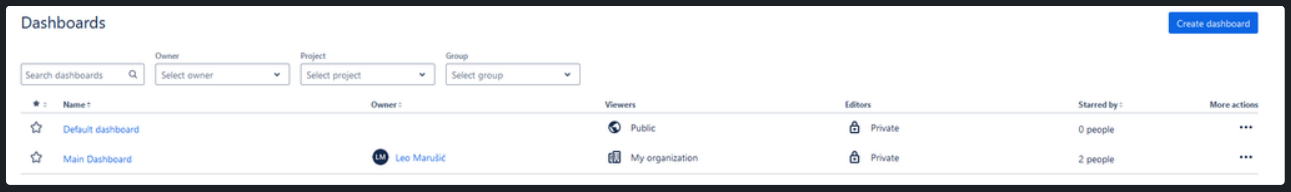
- Issues in progress - ticketi koji su u statusu “in progress“ - odnosi se na sve tikete
- All tickets - lista svih ticketa neovisno o statusu
- Participating or Awaiting input - ticketi u kojima je trenutni korisnik u “assingnee“ ili “participants“ polju, te ticketi koji zahtjevaju pažnju (vidi workflow)
- Activity stream - lista aktivnosti na Jiri i Confluence-u

Dodavanje Main Dashboard-a u quick access

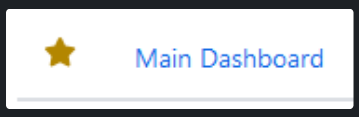
Klikom na Dashboards → View all dashboards



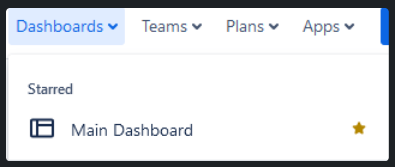
Odvest će se na stranicu svih dashboard-a, gdje se nalazi Main Dashboard.



Kliknuti na zvijezdicu za lakši pristup dashboard-u.

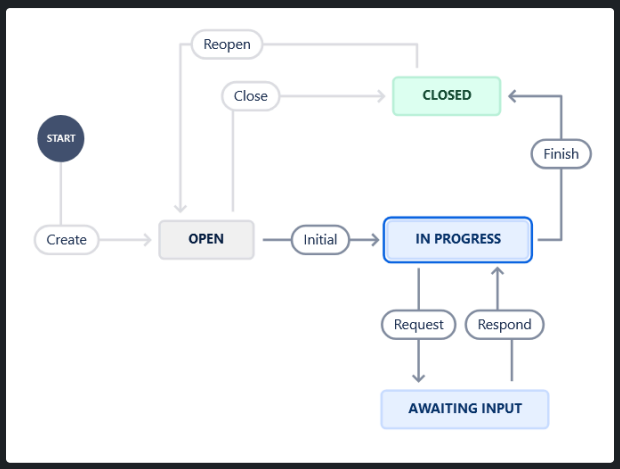


Te će onda Dashboards tab izgledati ovako:



Jira workflow

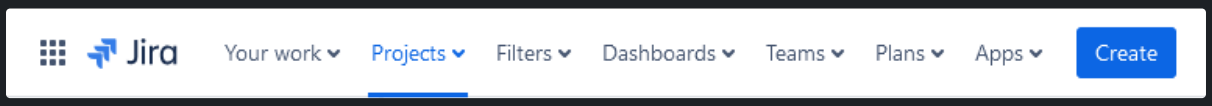
Workflow ticketa izgleda ovako:



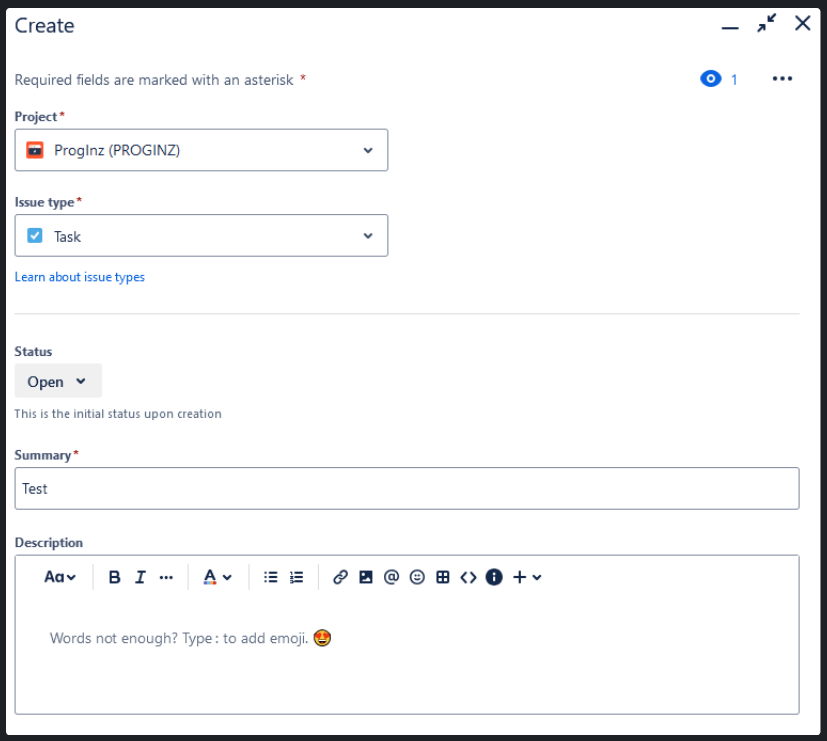
Kada se kreira ticket, započne odmah u “open“ statusu, zatim kada neko preuzme ticket, ta osoba stavlja ticket u “in progress“. Ako tijekom rada, ticket zahtjeva dodatan input od nekoga trećeg, stavlja se u “awaiting input“ status (koji onda svi vide u main dashboardu). Kada ta osoba odgovori ili se napravi posao koji se zahtjeva za nastavak ticketa, stavlja se nazad u “in progress“ i nastavlja se sa radom. Na kraju, ticket se stavlja u “closed“ status.

Otvaranje ticketa

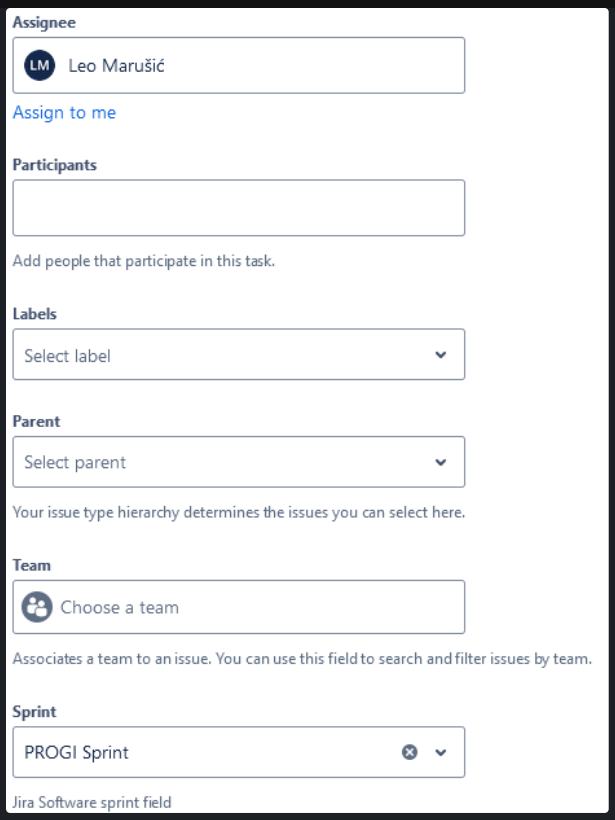
Klikom na Create gumb, otvara se prozor za izradu ticketa.



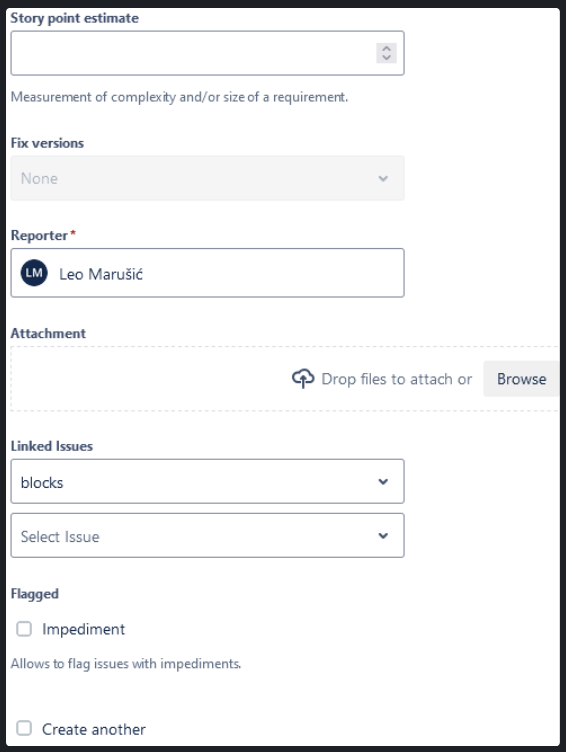
- Project: ProgInz (automatski se dodjeljuje)
- Issue type: najbolje ostaviti kao “Task“
- Summary: ime ticketa - treba popuniti
- Description: detalji ticketa, može ostati prazno u početku



- Assignee: dodjeliti sebi ili drugoj osobi za koju je taj ticket
- Participants: dodati ljude koji rade na ticketu
- Labels: može se dodati label po potrebi
- Parent: ignorirati
- Team: Određena su 2 tima, frontend i backend, može se njih ubaciti u to polje
- Sprint: **OBAVEZNO** postaviti na “PROGI Sprint“



- Story point estimate: procjena vremena u danima rada (1 dan = 8h)
- Fix versions: ignore
- Reporter: ostaviti default ili staviti na nekog drugog (npr. ako je team lead rekao da se napravi ticket)
- Attachment: upload za bilo kakav file vezan uz ticket
- Linked issued: postaviti ovisno o relaciji sa ostalim ticketima
- Flagged: ignore



Povezivanje sa Confluence-om

Ispod “Description“ polja u ticketu, nalazi se “Confluence content“ sekcija za povezivanje sa confluence-om.

Backend

FB Write a comment…

Deployment

All the information about the deployment of the web app.

Ubuntu server deployed on a Azure B1 virtual machine, with whole 1 CPU core and 1 Gib of RAM.
It runs an ASP .NET application as a *systemd* service which is interfaced with a *nginx* reverse proxy server.

Backend Guide

Svrha ovog priručnika je pojašnjenje backend koncepata koji se koriste u DuckI projektu.

Priručnik je pisan za ASP.NET Core, 26. 10. 2024.

Backend priručnik je podijeljen na više važnih cijelina:

- Modeli
- Konfiguracija entiteta
- DbContext
- Migracije
- Services
- Controllers

FB Write a comment…

Modeli

Unutar ,,Data" class project-a možemo pronaći direktorij ,,Modeli". U taj ćemo direktorij spremati klase koje će modelirati potrebne tablice u samoj bazi, te koje će se koristiti u nekim drugim dijelovima backenda, kao npr. ,,Services". Uz same modele tablice, u ovom folderu nalazi se i klasa koja nasljeđuje ,,DbContext".

Primjer klase `User`:

```
public class User
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public long IdUser { get; set; }

    public Guid Guid { get; set; }

    [Required]
    [StringLength(1000)]
    public string Password { get; set; }

    [Required]
    [StringLength(100)]
    public string Mail { get; set; }

    // navigational properties
    public Student? Student { get; set; }
    public Educator? Educator { get; set; }
    public Administrator? Administrator { get; set; }
    public Reviewer? Reviewer { get; set; }
}
```

Klasa User definira tablicu Users unutar baze. Ona sadrži više propertyja koji se prevode u stupce tablice: `IdUser`, `Guid`, `Password`, `Mail`. Svaki od ovih propertyja bit će kasnije preslikan u odgovarajući stupac tablice. Tip podataka koji će se spremati u određenom stupcu određuje tip propertyja. Npr., long se preslikava u bigint.

Propertyji Student, Educator, Administrator i Reviewer su navigacijski propertyji. Oni se ne preslikavaju u stupce tablice. Navigacijski propertyji služe za olakšavanja rada s podacima unutar ASP.NET Core-a. `?` na kraju svakog tipa navigacijskog propertyja označava da vrijednost tog propertyja može biti null.

Stvari unutar uglatih zagrada direktno iznad propertyja označavaju njegovo ograničenje ili svojstvo. Npr. `[Key]` označava primarni ključ, `[StringLength(1000)]` označava ograničenje na 1000 znakova.

Konfiguracija entiteta

Direktorij ,,Configuration” može biti pronađen unutar ,,Data” class project-a. Ovdje će se nalaziti sve klase koje će ,,manualno” puniti bazu podacima. Inače će korisnik puniti bazu podacima pozivajući odgovarajuće rute.

Primjer klase `UserConfiguration`:

```
public class UserConfiguration : IEntityTypeConfiguration<User>
{
    public void Configure(EntityTypeBuilder<User> builder)
    {
        builder.HasData(
            new User { IdUser = 1, Guid = Guid.NewGuid(), Password = "password1", Mail = "user1@example.com" },
            new User { IdUser = 2, Guid = Guid.NewGuid(), Password = "password2", Mail = "user2@example.com" },
            new User { IdUser = 3, Guid = Guid.NewGuid(), Password = "password3", Mail = "user3@example.com" },
            new User { IdUser = 4, Guid = Guid.NewGuid(), Password = "password4", Mail = "user4@example.com" },
            new User { IdUser = 5, Guid = Guid.NewGuid(), Password = "password5", Mail = "user5@example.com" }
        );
    }
}

```

Prvo što ovdje možemo primjetiti da klasa `UserConfiguration` nasljeđuje (implementira) sučelje `IEntityTypeConfiguration<User>`, parametrizirano s `User`. To je nešto što svaka klasa koja implementira nekakvu konfiguraciju opisanu u gornjem paragrafu mora imati. Klasa se parametrizira onim modelom entiteta u čiju tablicu ubacujemo podatke.

Klasa implementira metodu u `public void Configure(EntityTypeBuilder<User> builder)` u kojoj definiramo instance entiteta kao npr. `new User { IdUser = 1, Guid = Guid.NewGuid(), Password = "password1", Mail = "user1@example.com" }`.

DbContext

`DbContext` je jedna od najvažnijih klasa u cijelome backend dijelu projekta. `DbContext` koristi se za više stvari, primarno za povezivanje baze i za dohvat podataka ili uređivanje podataka iz baze.

Ova klasa implementira se pomoću klase `DataContext` koja direktno nasljeđuje `DbContext`. Klasa `DataContext` može se pronaći u direktoriju ,,Modeli", te je jedina klasa u tom direktoriju koja ne pripada entitetima.

Klasa `DataContext`:

```
public class DataContext : DbContext
{
    public DataContext(DbContextOptions options) : base(options) { }

    public DbSet<User>? Users { get; set; }
    public DbSet<Student>? Students { get; set; }
    public DbSet<Educator>? Educators { get; set; }
    public DbSet<Administrator>? Administrators { get; set; }
    public DbSet<Reviewer>? Reviewers { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.ApplyConfiguration(new StudentConfiguration());
        modelBuilder.ApplyConfiguration(new UserConfiguration());
        modelBuilder.ApplyConfiguration(new EducatorConfiguration());
        modelBuilder.ApplyConfiguration(new AdministratorConfiguration());
        modelBuilder.ApplyConfiguration(new ReviewerConfiguration());

        base.OnModelCreating(modelBuilder);
    }
}
```

Prvo imamo konstruktor `public DataContext(DbContextOptions options) : base(options) { }` koji call base konstruktor klase `DbContext`.

Nakon toga imamo `DbSet<T>` propertyje:

```
public DbSet<User>? Users { get; set; }
public DbSet<Student>? Students { get; set; }
public DbSet<Educator>? Educators { get; set; }
public DbSet<Administrator>? Administrators { get; set; }
public DbSet<Reviewer>? Reviewers { get; set; }
```

Za svaku tablicu koju kreiramo preko klasa entiteta u direktoriju ,,Models" moramo imati jedan `DbSet<T>` oblika iz gornjeg isječka.

Metoda:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
```

Služi za primjenjivanje konfiguracije koje smo spominjali u konfiguraciji entiteta.

FB Write a comment…

Migracije

Migracije služe kako bi primjenili sve važne konfiguracije kao npr. brisanje stupaca, dodavanje stupaca, dodavanje tablica i sl. na bazu podataka.

One se nalaze unutar ,,Data" class project-a.
Migracije se kreiraju pri svakoj manualnoj promijeni baze podataka, kao npr. mijenjanje strukture baze ili tablica, te seedanje podataka.

Migracije se kreiraju sljedećom naredbom u terminalu projekta:

```
Add-Migration Initial -Project Data -StartupProject DuckI.Server
```

Gdje -Project predstavlja projekt u kojem se nalazi migracija, dok StartupProject predstavlja projekt u kojem se nalazi `program.cs` u kojem je napravljena konekcija na bazu.

Ako se žele primjeniti migracije, u terminalu se pokreće `Update-Database`.

FB Write a comment…

Services

Servisi su usluge koje implementiramo za različite entitete. Servisi se nalaze u ,,Data" project-u.

Servisi započinju pisanjem interface-a. Unutar interface-a se zapišu definicije svih metoda koje servis implementira. Nakon toga se kreira klasa za odabrani servis, te se implementira napisani interface. Jedan interface može imati više metoda.

Klasa `UserService`:

```
using Data.Models;
using Microsoft.EntityFrameworkCore;

namespace Data.Services;

public interface IUserService
{
    Task<List<User>> GetUserRolesAsync();
}

public class UserService : IUserService
{
    private readonly DataContext _context;

    public UserService(DataContext context)
    {
        _context = context;
    }

    public async Task<List<User>> GetUserRolesAsync()
    {
        // implementacija funkcije, ugl. fetchanje podataka iz baze
    }
}

```

Dodatno, servis se mora registrirati u `program.cs`:

```
builder.Services.AddScoped<IUserService, UserService>();
```

FB Write a comment…

Controllers

Kontroleri služe za implementaciju ruta. Unutar njih ćemo pozivati metode koje implementiramo unutar servisa, te ćemo slati http response.

Mi ćemo kontrolere odvajati slično kao i servise, po entitetima, ali uz par iznimaka po potrebi.

Klasa `UserController`:

```
using Data.Dto;
using Data.Helpers;
using Data.Services;
using Microsoft.AspNetCore.Mvc;

namespace DuckI.Server.Controllers;

[ApiController]
[Route("api/users")]
public class UserController : ControllerBase
{
    private readonly IUserService _userRoleService;

    public UserController(IUserService userRoleService)
    {
        _userRoleService = userRoleService;
    }

    //[HttpGet("[action]")]  <-- almost the same
    [HttpGet("displayroles")]
    public async Task<IActionResult> DisplayRoles()
    {
        var users = // pozovi funkciju servisa
        var userRoles = // koristi dto
        return Ok(userRoles); // vrati http response
    }
}
```

Na početku se mogu primjetiti dva dekoratora, `[ApiController]` i `[Route("api/users")]`. Oni označavaju kontroler i default rutu za kontroler. Svaka metoda također mora biti dekorirana, kao npr. `[HttpGet("displayroles")]`. ,,displayroles" će ići na kraj api/users/ rute, ako se želi pozvati funkcija DisplayRoles. Dekoratori određuju rutu koja se koristi za poziv svake od funkcija.

Svaki kontroler sadrži člana koji je referenca na objekt prikladnog servisa.

FB Write a comment…

Frontend

FB Write a comment…

Općenito

Naš projekt koristi ASP.NET Razor Pages kao temelj za frontend dio aplikacije, omogućujući strukturiranu i modernu arhitekturu web stranica. Koristimo Bootstrap kao CSS framework za responzivni i estetski ugodan izgled stranice, olakšavajući rad na frontendu s unaprijed definiranim stilovima i komponentama.

Tehnologije:

Razor Pages: Stranični model programiranja unutar ASP.NET-a koji omogućava organizaciju logike i sadržaja pojedinačnih stranica bez korištenja klasične MVC arhitekture.

Bootstrap: CSS framework za responsive dizajn, koji osigurava da naša aplikacija izgleda jednako dobro na mobilnim uređajima, tabletima i desktop uređajima.

Struktura projekta

Pages direktorij: Sadrži .cshtml datoteke koje predstavljaju pojedinačne stranice, zajedno s pripadajućim .cshtml.cs datotekama za poslovnu logiku i povezivanje s backendom.

Index.cshtml: Početna stranica aplikacije.

Shared direktorij: Sadrži dijeljene komponente, uključujući Layout.cshtml koji definira izgled aplikacije, navigacijski izbornik i zaglavlje.

wwwroot direktorij: Sadrži sve statičke datoteke poput slika, stilskih datoteka i JavaScript datoteka.

CSS: Koristimo prilagođeni CSS (ako je potrebno) zajedno s Bootstrap CSS-om.

JS: Smještamo JavaScript datoteke (osim ako nisu uključene iz CDN-a).

_ViewImports.cshtml i _ViewStart.cshtml: Konfiguracijske datoteke za učitavanje često korištenih namespaceova i postavljanje početnih postavki.

Bootstrap integracija

Implementacija Bootstrapa omogućava nam brzo oblikovanje korisničkog sučelja bez previše prilagođavanja. Većina elemenata (forme, kartice, navigacija, gumbi) koristi Bootstrap komponente, što osigurava konzistentan izgled i jednostavniju prilagodbu.

Responzivni dizajn: Stranice su optimizirane za rad na različitim uređajima, čime poboljšavamo korisničko iskustvo bez potrebe za dodatnim stiliziranjem.

Komponente: Koristimo komponente kao što su modali, dropdown izbornici, tablice i forme kako bi aplikacija izgledala profesionalno i bila funkcionalna.