

Elektrotehnički fakultet Sarajevo

EBANK

INTERNA DOKUMENTACIJA

**Predmet: Objektno orijentisana analiza
i dizajn**

Grupa: **Bankari**

Enver Suljić

Berina Suljić

Amina Šiljak

10. maj 2020. u Sarajevu

Interna Dokumentacija

Prilikom testne upotrebe našeg bankarskog sistema možete se poslužiti predefinisanim login podacima:

Administrator

Username: admin

Password: admin

Bankar

Username: bankar

Password: bankar

Klijent

Username: klijent

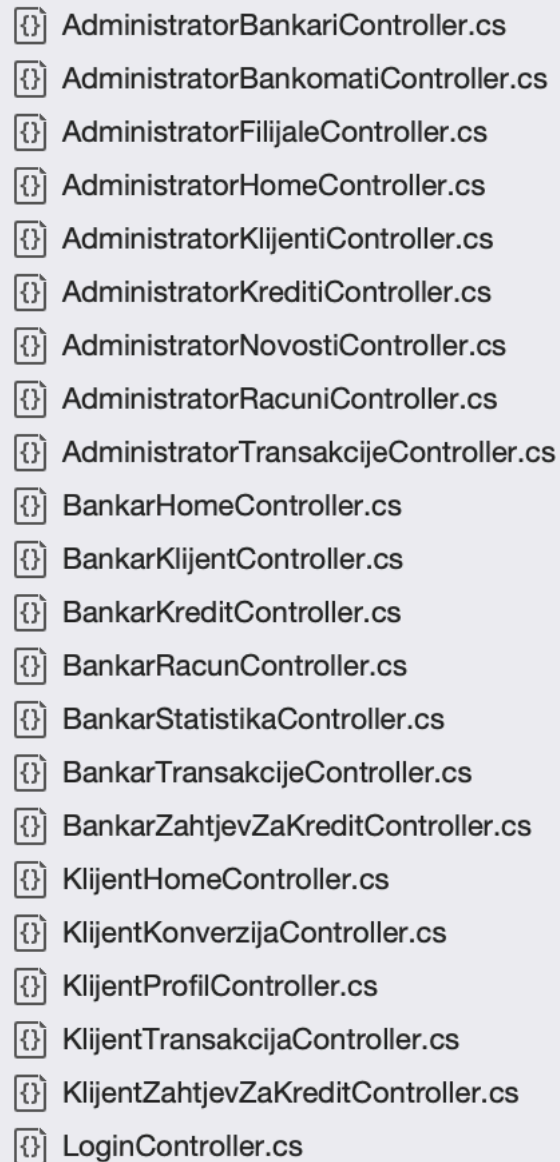
Password: klijent

Naš projekat je strukturiran na sljedeći način:

- Controllers
- Data
- Migrations
- Models
- Utils
- Views
- wwwroot

Controllers

Unutar ovog paketa se nalaze svi naši kontroleri:



A list of 20 C# Controller files, each preceded by a small folder icon. The files are arranged in a single column and include:

- AdministratorBankariController.cs
- AdministratorBankomatiController.cs
- AdministratorFilijaleController.cs
- AdministratorHomeController.cs
- AdministratorKlijentiController.cs
- AdministratorKreditController.cs
- AdministratorNovostiController.cs
- AdministratorRacuniController.cs
- AdministratorTransakcijeController.cs
- BankarHomeController.cs
- BankarKlijentController.cs
- BankarKreditController.cs
- BankarRacunController.cs
- BankarStatistikaController.cs
- BankarTransakcijeController.cs
- BankarZahtjevZaKreditController.cs
- KlijentHomeController.cs
- KlijentKonverzijaController.cs
- KlijentProfilController.cs
- KlijentTransakcijaController.cs
- KlijentZahtjevZaKreditController.cs
- LoginController.cs

Svaki od njih je struktuiran na sličan način:

- implementiraju *Controller* klasu
- u atributima se nalaze proxy klase, instanca trenutno logovanog *Korisnik*-a, i instanca *OOADContext*-a
- u konstruktoru se instanciraju sve proxy klase, i čuva kontekst da bi se proslijedio modulu za autentikaciju korisnika
- akcije kao što su **Index**, **Details**, **Edit**...

```

public class AdministratorBankariController : Controller
{
    private BankariProxy _bankari;
    private Korisnik korisnik;
    private OOADContext Context;

    public AdministratorBankariController(OOADContext context)
    {
        _bankari = new BankariProxy(context);
        Context = context;
    }
}

```

Unutar svake akcije se vrši autentikacija korisnika:

```

public async Task<IActionResult> Index()
{
    korisnik = await LoginUtils.Authenticate(Request, Context, this);
    if (korisnik == null) return RedirectToAction("Logout", "Login", new { area = "" });

    _bankari.Pristupi(korisnik);

    ViewData["Ime"] = korisnik.Ime;
    return View(await _bankari.DajSveBankare());
}

```

Ukoliko LoginUtils.Authenticate() vrati null vrijednost, znači da korisnik nije logovan, ili nešto drugo nije u redu, pa se preusmjerava na Login stranicu, i to na akciju Logout kako bi se očistili podaci o loginu.

Osim toga, na osnovu korisnika se pozivaju Pristupi metode na svim Proxy klasama koje su potrebne za funkcionisanje tog dijela stranice. To je potrebno kako bi se isti repository-ji mogli koristiti u različitim dijelovima aplikacije, za različite vrste korisnika (Administrator, Bankar, Klijent), uz regulisanje prava svakog od njih.

Svi kontroleri su slični, i razlikuju se samo u proxyjima koje koriste, i metodama koje implementiraju. Sva logika se dešava u drugim klasama.

Data

U ovom paketu su Context klase koje se koriste za definisanje structure baze. U OOADContext klasi su definisane sve tabele, i na osnovu kojih modela.

```
public OOADContext(DbContextOptions<OOADContext> options) : base(options)
{
}
public DbSet<Klijent> Klijent { get; set; }
public DbSet<Bankar> Bankar { get; set; }
public DbSet<Administrator> Administrator { get; set; }
public DbSet<Adresa> Adresa { get; set; }
public DbSet<Bankomat> Bankomat { get; set; }
public DbSet<Filijala> Filijala { get; set; }
public DbSet<Kredit> Kredit { get; set; }
public DbSet<Novost> Novost { get; set; }
public DbSet<Racun> Racun { get; set; }
public DbSet<Transakcija> Transakcija { get; set; }
public DbSet<ZahtjevZaKredit> ZahtjevZaKredit { get; set; }
//Ova funkcija se koriste da bi se ukinulo automatsko dodavanje mnozine u nazive
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    //modelBuilder.Entity<Korisnik>().ToTable("Korisnik");
    modelBuilder.Entity<Adresa>().ToTable("Adresa");
    modelBuilder.Entity<Bankomat>().ToTable("Bankomat");
    modelBuilder.Entity<Filijala>().ToTable("Filijala");
    modelBuilder.Entity<Kredit>().ToTable("Kredit");
    modelBuilder.Entity<Novost>().ToTable("Novost");
    modelBuilder.Entity<Racun>().ToTable("Racun");
    modelBuilder.Entity<Transakcija>().ToTable("Transakcija");
    modelBuilder.Entity<ZahtjevZaKredit>().ToTable("ZahtjevZaKredit");
}
```

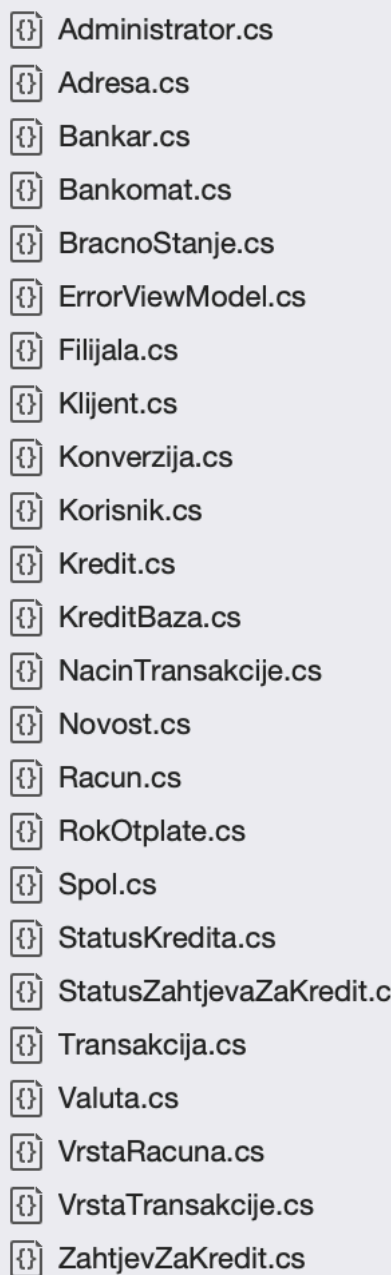
Migrations

Tu su definisane migracije baze. Pošto prvi put publishamo aplikaciju, dovoljno je imati jednu migraciju koja će kreirati bazu. Ukoliko dođe do potrebe da se baza mijenja u toku produkcije, bit će potrebno praviti migraciju kako se podaci ne bi izgubili.

Models

U paketu models imamo sve modele, ka oi njihove repository-je.

Naši modeli u projektu su:



- Administrator.cs
- Adresa.cs
- Bankar.cs
- Bankomat.cs
- BracnoStanje.cs
- ErrorViewModel.cs
- Filijala.cs
- Klijent.cs
- Konverzija.cs
- Korisnik.cs
- Kredit.cs
- KreditBaza.cs
- NacinTransakcije.cs
- Novost.cs
- Racun.cs
- RokOtplate.cs
- Spol.cs
- StatusKredita.cs
- StatusZahtjevaZaKredit.c
- Transakcija.cs
- Valuta.cs
- VrstaRacuna.cs
- VrstaTransakcije.cs
- ZahtjevZaKredit.cs

Za primjer ćemo prikazati model Adresa.

```
public class Adresa
{
    [ScaffoldColumn(false)]
    public int Id { get; set; }
    [Required]
    public float Latitude { get; set; }
    [Required]
    public float Longitude { get; set; }
    [Required]
    [DisplayName("Ulica")]
    public string Naziv { get; set; }
}
```

Kao što vidimo, u njoj imamo četiri atributa:

- Id
- Latitude
- Longitude
- Naziv

Atribut Id je označen anotacijom ScaffoldColumn(false), što znači da prilikom automatskog generisanja kontrolera i viewa taj atribut neće biti prikazan korisniku.

Atributi Latitude, Longitude i Naziv su označeni anotacijom Required, koja govori da se ti atributi moraju unijeti prilikom kreiranja novog objekta.

Vidimo da je za atribut Naziv postavljena anotacija DisplayName("Ulica"), koji govori ime koje će se koristiti prilikom prikazivanja tog polja u Viewima.

Repository klase

Za većinu klasa imamo njegovu repository klasu koja komunicira sa bazom, i ima metode kao što su DajAdministradora() ili DodajRacun(). Sve funkcije repository preklapa iz svog interfejsa. Jedan od njih izgleda ovako:

```
interface IKlijenti
{
    public Task DodajKlijenta(Klijent klijent);
    public Task UrediKlijenta(Klijent klijent);
    public Task UkloniKlijenta(int? id);
    public Task<List<Klijent>> DajSveKlijente();
    public Task<Klijent> DajKlijenta(int? id);
    public Task<Klijent> DajKlijentaLK(string brojLicneKarte);
    public bool DaLiPostojiKlijent(int? id);
    public Task<Klijent> DajKlijenta(string korisnickoIme);
}
```

Vidimo da se tu nalaze sve potrebne metode za rad sa bazom.

Osim Repository klase, ovaj interfejs implementira i njihova Proxy klasa. Proxy pattern nam ovdje omogućava da na osnovu korisnika omogućimo da se mogu izvršiti tačno određene akcije. Na primjer, Klijent ne može uređivati Filijale i Bankomate, ali ih može pregledati.

Utils

U ovom paketu se nalaze pomoćne klase LoginUtils i Konvertor.

LoginUtils je klasa koja omogućava autentifikaciju korisnika za svaku akciju u kontroleru, ii ma jednu funkciju Authenticate koja izgleda ovako:

```
public static async Task<Korisnik> Authenticate(HttpRequest Request, OOADContext Context, Controller Controller)
{
    IAdministratori _administratori = new Administratori(Context);
    IBankari _bankari = new Bankari(Context);
    IKlijenti _klijenti = new Klijenti(Context);

    if (Request.Cookies["userId"] == null || Request.Cookies["userId"].Equals("")) return null;

    var userId = int.Parse(Request.Cookies["userId"]);
    var role = Request.Cookies["role"];

    if (role == "Administrator") return await _administratori.DajAdministratora(userId);
    else if (role == "Bankar") return await _bankari.DajBankara(userId);
    else if (role == "Klijent") return await _klijenti.DajKlijenta(userId);
    else return null;
}
```

Ova funkcija vraća korisnika, koji se kasnije proslijeđuje u Pristupi() metode proxy klasa.

Druga pomoćna klasa u našem sistemu je **Konvertor**, koja komunicira sa fixer.io API-jem, pomoću kojeg se konvertuju valute na osnovu realtime vrijednosti.

Funkcija koja obavlja konvertovanje je konvertujDevizuAsync():

```
public async Task<float> konvertujDevizuAsync(float iznos, string izValute, string uValutu)
{
    using (var client = new HttpClient())
    {
        client.BaseAddress = new Uri("http://data.fixer.io/api/latest" +
            "?access_key=ba8e84f219054816408d7814d5b43b0c");

        var result = await client.GetAsync("");
        if (result.IsSuccessStatusCode)
        {
            var response = await result.Content.ReadAsStringAsync();
            var rates = JsonConvert.DeserializeObject<JToken>(response).Root.Element("rates");
            var from = float.Parse(rates.Element(izValute).Value);
            var to = float.Parse(rates.Element(uValutu).Value);
            return iznos * to / from;
        }
    }

    return 0;
}
```


Views

U paketi Views se nalaze svi naši pogledi. Struktuirani su u podpakete za svaki kontroler. Svaki od viewa je na početku generisan Scaffoldom, ali je dosta izmijenjen u odnosu na početni izgled.

Neki od pogleda su potpuno custom napravljeni, kao što su Home stranice za svakog od korisnika (Administrators, Bankara i Klijenta).

Također, na nekim od pogleda nije bilo dovoljno raditi samo sa modelom tog pogleda, nego su se iz kontrolera morale prosljeđivati razne druge vrijednosti. Primjer toga su prosljeđene liste bankomata i filijala u KlijentHome pogledu. Da bi se one koristile, prvo ih je potrebno castati:

```
@{  
    var bankomati = (List<Bankomat>) ViewData["bankomati"];  
    var filijale = (List<Filijala>) ViewData["filijale"];  
}
```

Njih smo iskoristili da prikazemo markere bitnih objekata banke na google mapi. Za to smo koristili googleov api, i instrukcije sa njihove stranice, uz malu modifikaciju sa razorom i javascriptom. Sljedeći dio koda dodaje filijale na mapu:

```
var infowindow = new google.maps.InfoWindow();  
  
var marker, i;  
  
for (i = 0; i < bankomati.length; i++) {  
    marker = new google.maps.Marker({  
        position: new google.maps.LatLng(bankomati[i][1], bankomati[i][2]),  
        map: map  
    });  
  
    google.maps.event.addListener(marker, 'click', (function (marker, i) {  
        return function () {  
            infowindow.setContent("Bankomat: " + bankomati[i][0]);  
            infowindow.open(map, marker);  
        }  
    })(marker, i));  
}  
  
for (i = 0; i < filijale.length; i++) {  
    marker = new google.maps.Marker({  
        position: new google.maps.LatLng(filijale[i][1], filijale[i][2]),  
        map: map  
    });  
  
    google.maps.event.addListener(marker, 'click', (function (marker, i) {  
        return function () {  
            infowindow.setContent("Filijala: " + filijale[i][0] + " (" + filijale[i][4] + ")");  
            infowindow.open(map, marker);  
        }  
    })(marker, i));  
}
```

Vidimo kako će se klikom na marker neke filijale prikazati njeno ime i kontakt broj.

Još jedan bitan dio Views paketa su Shared pogledi. To su okviri za stranice unutar kojih se prikazuju pojedinačne stranice. Pa tako za home stranice imamo poseban layout u shared folderu, jer on ne treba imati sidebar, dok onaj za ostale poglede treba.

Ovdje se izdvaja Login pogled, koji se prikazuje u layoutu _Empty, jer on ne zahtjeva ništa od okvira osim pozadinske slike i footera.

wwwroot

Ovaj paket sadrži cijeli izvorni kod stranice koji se sastoji od css fajlova, javascript skripti, slika, fontova...

Koristili smo bootstrap SB Admin template, koji smo uređivali po našim potrebama.