



Univerzitet u Sarajevu
Elektrotehnički fakultet u Sarajevu
Odsjek za računarstvo i informatiku



Refaktoring

-BrainBoost-



Članovi:

Amar Grizović

Imran Spahić

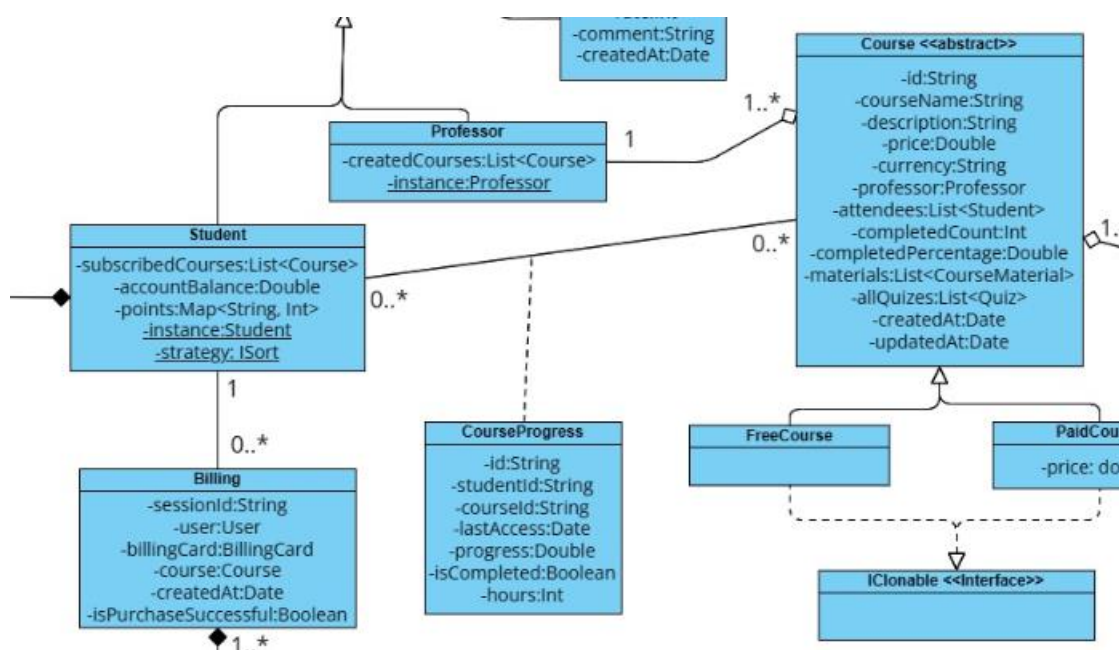
Berin Karahodžić

Asmir Prašović

Nedim Hošić

Refaktoring

Prvi problem sa kojim smo se susreli bio je način implementacije veze više naprema više. Prvobitno smo imali dvije klase Course i Student koje su zamišljene na način da jedan student može biti prijavljen na više kurseva, a također na jednom kursu može biti više studenata. Ubrzo smo uvidjeli da će nam biti potrebna međuklasa koja će sadržavati identifikacione brojeve studenta i kursa i preko nje je moguće uspješno uspostaviti relaciju između pomenutih klasa. Ova međuklasa koju smo nazvali CourseProgress može posjedovati i druge atribute koji su nam korisni prilikom implementacije.



Naredne dvije stavke koje smo izdvojili u refaktoringu se tiču estetike i pisanja clean koda. Prvobitno je cilj tima bio da se napravi funkcionalna cjelina i nismo detaljno obraćali pažnju na način njegove realizacije. Kada smo završili sa implementacijom novih funkcionalnosti, počeli smo uviđati mnoga ponavljanja koda i pisanja bespotrebnih grananja. U nastavku smo izdvojili neke od njih i vizuelni prikaz kako smo riješili iste.

```
public async Task<IActionResult> Index()
{
    bool isAuthenticated = User.Identity.IsAuthenticated;
    System.Diagnostics.Debug.WriteLine(isAuthenticated);
    if (User.IsInRole("Professor"))
    {
        if(isAuthenticated)
            TempData["KLjuc"] = _context.Professor.FirstOrDefault(p => p.Username == User.Identity.Name).UserId;
    }
    else
    {
        if(isAuthenticated)
            TempData["KLjuc"] = _context.Student.FirstOrDefault(p => p.Username == User.Identity.Name).UserId;
    }
    if (isAuthenticated)
    {

```

Na prethodnoj slici možemo vidjeti kako se prvo vrši provjera u zavisnosti od toga koja je trenutno ulogovana rola na aplikaciju. Zatim smo zbog logike koja je implementirana, a neće biti trenutno obrazložena, morali vršiti provjeru da li je uopšte korisnik ulogovan da aplikacija ne bi krahirala. Kada smo sve to provjerili postavila se odgovarajuća vrijednost varijable na view-u koji je potrebno učitati, nastavlja se dalji tok programa. Pošto smo radili u timu, neko od kolega koji nije bio zadužen za ovu funkcionalnost nije obraćao pažnju na napisani kod, i vrši opet provjeru da li je korisnik ulogovan na aplikaciju, iako je to u prethodnom koraku već utvrđeno. U prilogu slijedi kako je prepravljen loše napisani kod.

```
public async Task<IActionResult> Index()
{
    bool isAuthenticated = User.Identity.IsAuthenticated;
    System.Diagnostics.Debug.WriteLine(isAuthenticated);
    if (isAuthenticated)
    {
        if (User.IsInRole("Professor"))
        {
            TempData["Kljuc"] = _context.Professor.FirstOrDefault(p => p.Username == User.Identity.Name).UserId;
        }
        else
        {
            TempData["Kljuc"] = _context.Student.FirstOrDefault(p => p.Username == User.Identity.Name).UserId;
        }
    }
}
```

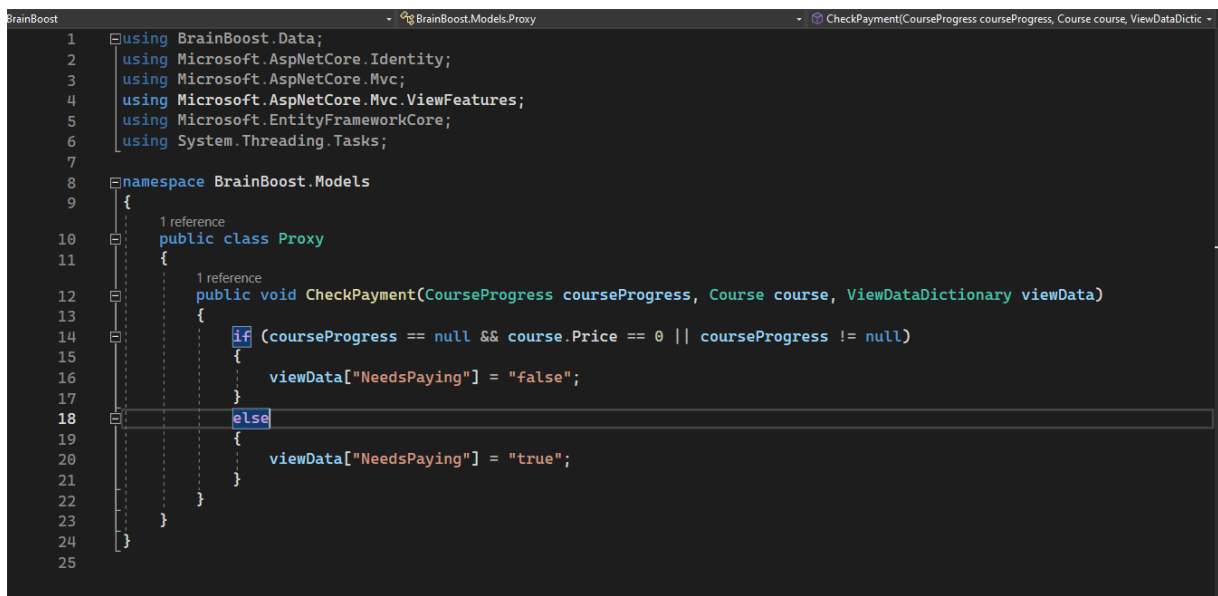
Sljedeća uočena mana napisanog koda bila je ponavljanje. Potrebno nam je bilo u backend logici registracije da spasimo korisnika koji se registruje u našu tabelu User iz koje su izvedene Student i Professor. Ovo se desilo zbog nepredviđenih okolnosti korištenja Identity ASPNetUser tabele koja spašava novog usera, a evidencija u našoj tabeli je morala biti interno implementirana. U zavisnosti od provjere koja se rola registruje na sistem, tako je pravljen slog koji će biti mapiran u bazu podataka. Problem je uočen tek kada je kod završen gdje se većina atributa koji se dodjeljuju ponavlja, jedina razlika je u kreiranju instance varijable.

```
string combo = Request.Form["role"].ToString();
if (combo == "Student")
{
    Models.User u = new Models.Student();
    u.FirstName = Input.FirstName;
    u.LastName = Input.LastName;
    u.Email = Input.Email;
    u.BirthDate = Input.BirthDate;
    u.CreatedAt = DateTime.Now;
    u.IsVerified = true;
    u.Username = Input.Username;
    context.User.Add(u);
}
else
{
    Models.User u = new Models.Professor();
    u.FirstName = Input.FirstName;
    u.LastName = Input.LastName;
    u.Email = Input.Email;
    u.BirthDate = Input.BirthDate;
    u.CreatedAt = DateTime.Now;
    u.IsVerified = true;
    u.Username = Input.Username;
    context.User.Add(u);
}
context.SaveChanges();
```

Ovo smo riješili na način da smo sve dodjele koje se ponavljaju u oba dijela if uslova izdvojili izvan uslova, a samo određivanje tipa instance ostavili unutar. U prilogu slika sa koje se jasno vidi promjena.

```
string combo = Request.Form["role"].ToString();
Models.User u;
if (combo == "Student")
{
    u = new Models.Student();
}
else
{
    u = new Models.Professor();
}
u.FirstName = Input.FirstName;
u.LastName = Input.LastName;
u.Email = Input.Email;
u.BirthDate = Input.BirthDate;
u.CreatedAt = DateTime.Now;
u.IsVerified = true;
u.Username = Input.Username;
context.User.Add(u);
context.SaveChanges();
```

U sklopu pisanja clean koda neophodno je spomenuti dizajn patterne. Njihova svrha je pisanje efektivnog koda u karakterističnim situacijama kada je moguće primijeniti iste. Naš tim je vješto implementirao dizajn patterne, od kojih ćemo navesti u nastavku obrazloženje jednog i njegovu primjenu, ostali imaju sličnu namjenu i implementaciju. Iskoristili smo proxy pattern za provjeru da li korisnik ima pristup sadržaju kursa, tj. da li se kurs plaća, i ako da, da li je plaćen od strane korisnika. U samu metodu se mogao slati samo context te da proxy pristupa bazi umjesto kontrolera, ipak smo odlučili to da ne radimo, pošto je samom kontroleru potreban pristup bazi, tako da ne pristupamo dva puta. U nastavku su priložene slike koje pokazuju njegovu implementaciju u našoj aplikaciji.



```
1 using BrainBoost.Data;
2 using Microsoft.AspNetCore.Identity;
3 using Microsoft.AspNetCore.Mvc;
4 using Microsoft.AspNetCore.Mvc.ViewFeatures;
5 using Microsoft.EntityFrameworkCore;
6 using System.Threading.Tasks;
7
8 namespace BrainBoost.Models
9 {
10     1 reference
11     public class Proxy
12     {
13         1 reference
14         public void CheckPayment(CourseProgress courseProgress, Course course, ViewDataDictionary viewData)
15         {
16             if (courseProgress == null && course.Price == 0 || courseProgress != null)
17             {
18                 viewData["NeedsPaying"] = "false";
19             }
20             else
21             {
22                 viewData["NeedsPaying"] = "true";
23             }
24         }
25     }
26 }
```

```
        ViewData["action"] = "CourseBidding";  
    }  
  
    }  
  
    var courseMaterials = await _context.CourseMaterial  
        .Where(cm => cm.CourseId == id)  
        .ToListAsync();  
    var proxy = new Proxy();  
    proxy.CheckPayment(courseProgress, course, ViewData);  
    ViewData["CourseMaterials"] = courseMaterials;  
    return View(course);  
}
```

Ovdje su pobrojani samo neki od primjera refaktoringa u našoj aplikaciji, oni koje smo mi smatrali reprezentativnim za prikaz.