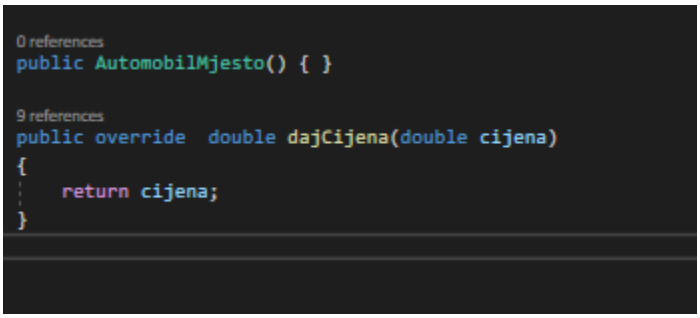
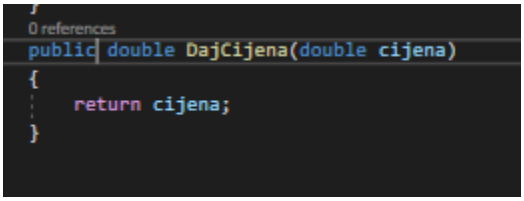
Refaktoring

-Loše imenovanje varijabli i metoda

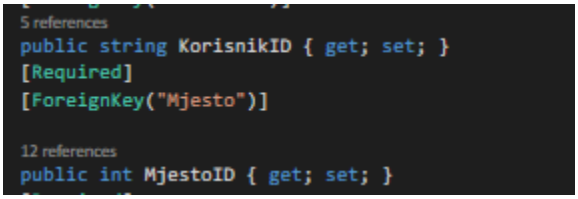
Kada su u pitanju best-practices i code-smells, prvi i najočigledniji code-smell u našoj aplikacija je imenovanje metoda i parametara.



U prethodnom primjeru vidimo da ime metode pocinje malim umjesto velikim slovom. Ispravno bi bilo:

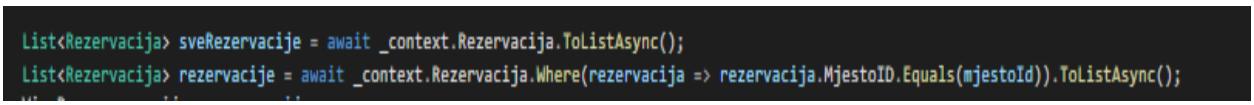


Kada su u pitanju parametri odnosno varijabli, imamo sljedeći primjer:



KorisnikID i MjestoID bi trebali biti KorisnikId i MjestoId.

Pored samog formata imena varijabli i metoda veliki problem je što veliki broj njih nije dovoljno self-describing. Štaviše, u dosta situacija je teško razlikovati dvije varijable, što se naročito javlja u kontrolerima.



U prethodnom primjeru vidimo da “rezervacije” se odnose na rezervacije nekog specifičnog mjesta, međutim ukoliko ne pogledamo samu inicijalizaciju ove varijable, na osnovu imena ne možemo zaključiti koja je svrha iste. Ispravno bi bilo npr. imenovati varijablu kao “rezervacijeMjestaId“.

-Anit-pattern

Pored imenovanja varijabli i metoda, jedna od greški je anti-pattern ponašanje kao u sljedećem primjeru gdje se očigledno krši factory-method pattern:

```
OsobaSInvaliditetomPopust osobaSInvaliditetomPopust = OsobaSInvaliditetomPopust.getInstance();
StalniGostMjesecnoPopust stalniGostMjesecnoPopust = StalniGostMjesecnoPopust.getInstance();
StalniGostUzastopnoPopust stalniGostUzastopnoPopust = StalniGostUzastopnoPopust.getInstance();

var user = await _userManager.GetUserAsync(User);
var trenutniKorisnik = await _context.Users.FirstOrDefaultAsync(usercic => usercic.UserName.Equals(user.UserName));



if (stalniGostMjesecnoPopust.dajKriterij() < trenutniKorisnik.ProvedenoVrijeme)
{
    rezervacija.Cijena = rezervacija.Cijena - stalniGostMjesecnoPopust.dajIznos() * rezervacija.Cijena;
}
if (stalniGostUzastopnoPopust.dajKriterij() < trenutniKorisnik.UzastopnoVrijeme)
{
    rezervacija.Cijena = rezervacija.Cijena - stalniGostUzastopnoPopust.dajIznos() * rezervacija.Cijena;
}
if (trenutniKorisnik.Invaliditet)
{
    rezervacija.Cijena = rezervacija.Cijena - osobaSInvaliditetomPopust.dajIznos() * rezervacija.Cijena;
}
```

Umjesto inicijalizacije navedinih klasa za popuste te zatim provjeravnaja kriterija, puno bi više smisla imalo da se prvo kriterij provjeri unutar neke *Factory* klase pa da se inicijalizuju klase u skladu s tim. Da su u pitanju veće klase ili da ih ima više, zauzimala bi se memorija bez potrebe.

-Ponavljanje koda

Česta je situacija gdje se iznova prave dijelovi metoda ili čitave metode koji su već napravljeni umjesto da se iskoriste već postojeće.

```
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var rezervacija = await _context.Rezervacija.FindAsync(id);
    var mjesto = await _context.Mjesto.FindAsync(rezervacija.MjestoID);
    List<Rezervacija> rezervacije = await _context.Rezervacija.Where(rezervacija => rezervacija.MjestoID.Equals(id)).ToListAsync();
    if (rezervacije.Count == 1)
    {
        mjesto.Zauzeto = false;
    }
    _context.Update(mjesto);
    _context.Rezervacija.Remove(rezervacija);
    await _context.SaveChangesAsync();
    return RedirectToAction("Index","Pocetna");
}
```

Na ovom mjestu je regulisano brisanje neke rezervacije iz baze, a dok na sljedećoj slici vidimo da se ponovo pišu dijelovi ove metode u odvojenom controlleru:

```
1 reference
public async Task<IActionResult> Index(int id)
{
    Parking parking = await _context.Parking.FindAsync(id);
    List<Mjesto> mjesta = await _context.Mjesto.Where(mjesto => mjesto.ParkingId.Equals(id)).ToListAsync();
    List<Rezervacija> rezervacije = await _context.Rezervacija.ToListAsync();
    List<Rezervacija> krajnjeRezervacije = new List<Rezervacija>();
    foreach (Rezervacija rezervacija in rezervacije)
    {
        if (rezervacija.VrijemeIsteka < System.DateTime.Now)
        {
            var mjesto = await _context.Mjesto.FindAsync(rezervacija.MjestoID);
            List<Rezervacija> rezervacijeMjesta = await _context.Rezervacija.Where(rezervacija => rezervacija.MjestoID.Equals(id)).ToListAsync();
            if (rezervacijeMjesta.Count == 1)
            {
                mjesto.Zauzeto = false;
            }
            _context.Update(mjesto);
            _context.Rezervacija.Remove(rezervacija);
            _context.SaveChangesAsync();

        }
        else
        {
            krajnjeRezervacije.Add(rezervacija);
        }
    }
```

Pored toga što ima nepotrebnog ponavljanja koda u prethodnom primjeru, klasa (controller) je zadužena za nešto što ne bi trebala biti zadužena. Pozivanje metoda iz drugih kontrolera nije baš zahvalno pa prethodno navedeno nije toliki problem koliki je situaciju (koju mi imamo) da se unutar istog controllera pišu dijelovi koda neke metode iz tog controllera što vidimo na sljedećoj slici

```
0 references
public async Task<IActionResult> PrikazRezervacija()
{
    var user = await _userManager.GetUserAsync(User);
    var trenutniKorisnik = await _context.Users.FirstOrDefaultAsync(usercic => usercic.UserName.Equals(user.UserName));
    List<Rezervacija> rezervacije = await _context.Rezervacija.Where(rezervacija => rezervacija.KorisnikID.Equals(trenutniKorisnik.Id)).ToListAsync();
    List<Rezervacija> izabraneRezervacija = new List<Rezervacija>();
    foreach (Rezervacija rezervacija in rezervacije)
    {
        if (rezervacija.VrijemeIsteka < System.DateTime.Now)
        {
            var mjesto = await _context.Mjesto.FindAsync(rezervacija.MjestoID);
            mjesto.Zauzeto = false;
            _context.Update(mjesto);
            _context.Rezervacija.Remove(rezervacija);
            _context.SaveChangesAsync();

        }
        else
        {
            izabraneRezervacija.Add(rezervacija);
        }
    }


    ViewBag.rezervacije = izabraneRezervacija;
    return View();
}
```

-Preduge rutine (metode)

Još jedna od zamjerki je prevelika metoda Rezervisanje u klasi RezervacijasController koja je napisana u 90 linija koda (zbog čega nismo u mogućnosti sliku metode prikazati), a zadužena je za primanje podataka sa odgovarajućeg view-a, provjeru validnosti tih podataka, provjeru podataka u skladu sa određenim kriterijima i što je najveći problem provjeru da li korisnik zadovoljava kriterije za popuste što bi se mogla vrlo lahko prebaciti u neku odvojenu metodu (iako bi to svakako bilo riješeno uvođenjem factory klase za popuste).