# Konkurentan pristup resursima u bazi

## Konflikt 1

### Vlasnik vikendice/broda ili instruktor ne može da napravi rezervaciju u isto vreme kad i drugi klijent

Prva konfliktna situacija nastaje kada vlasnik vikendice/broda ili instruktor pecanja pokuša da napravi rezervaciju za nekog svog klijenta u istom ili preklapajućem periodu kao neki drugi klijent. Prilikom obrade rezervacije sistem u oba slučaja dostavlja identičnu listu slobodnih termina iz baze. Čuvanje prve rezervacije dovodi do nekonzistentnosti između stanja u bazi i podataka koje sistem ima pri obradi druge rezervacije. Pri pokušaju čuvanja druge rezervacije dolazi do konflikta.

### Solucija:

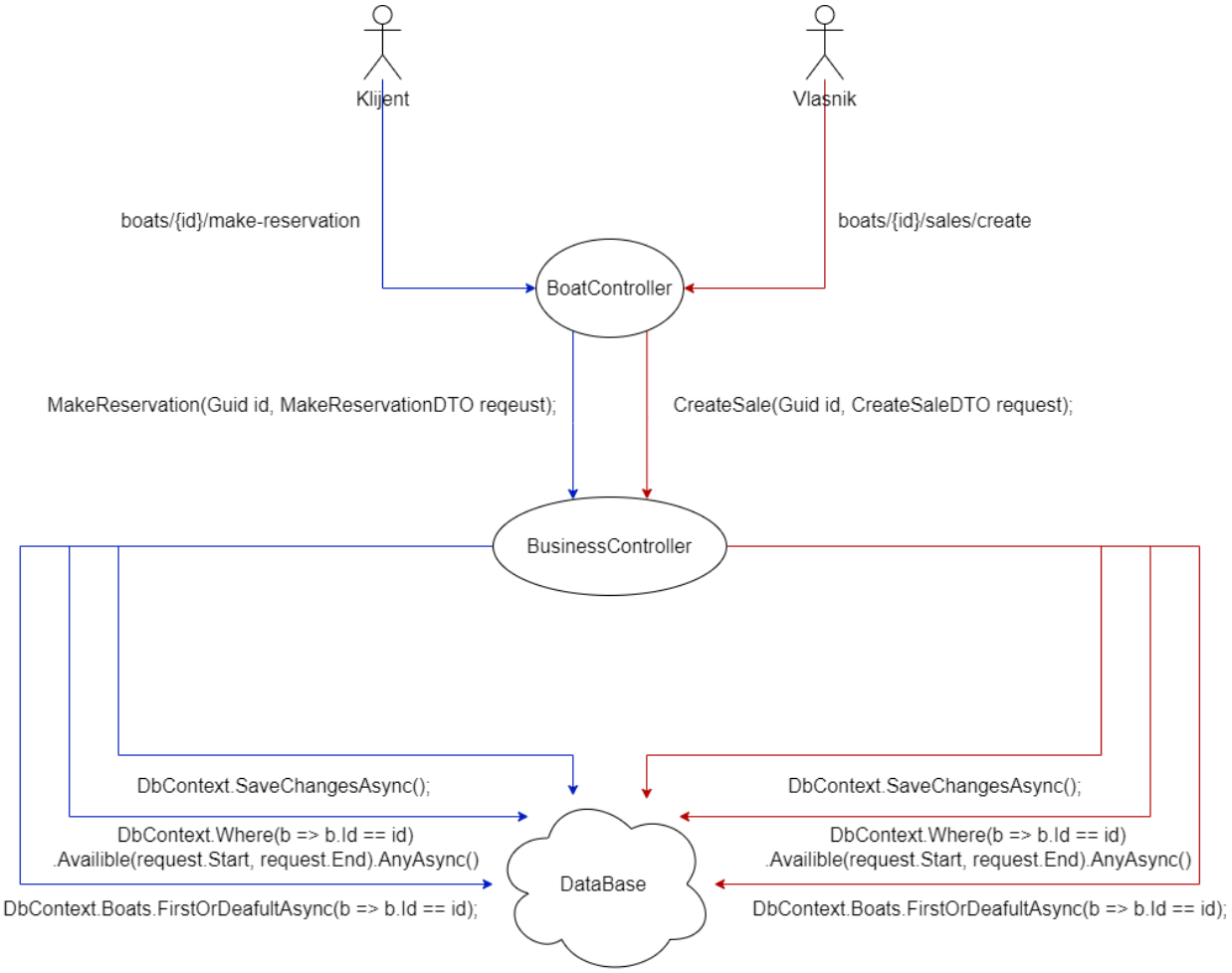
Problem rešavamo postavljanjem pesimističkog zaključavanja na metodu za kreiranje rezervacija kao i na metodu za produzavanje rezervacjie, sa nivoom izolacije *Serializable.* Time će samo jedan korisnik moći da pristupi entitetu, dok će drugi morati da pokušaju ponovo. Pesimistično zaključavanje se koristi kako bi se osiguralo korišćenje najnovijih podataka pri proveri dostupnosti.

```
await using var transaction = await Context.Database.BeginTransactionAsync(IsolationLevel.Serializable);
try
{
    var business = await Context.Set<TBusiness>()
            .Include(b => b.Owner)
            .Include(b => b.Services)
            .Include(b => b.Subscribers)
            .FirstOrDefaultAsync(b => b.Id == id);

    if (business is null)
    {
        return BadRequest($"The specified {BusinessType} does not exist.");
    }
```

```
        return Ok(sale.Adapt<SaleDto>());
    }
    catch (Exception)
    {
        await transaction.RollbackAsync();
        return BadRequest();
    }
}
```

## Tok zahteva



# Konflikt 2

## Vlasnik vikendice/broda ili instruktor pecanja ne može da napravi akciju u isto vreme kad neki klijent vrši rezervaciju postojećeg entiteta

Druga konfliktna situacija nastaje kada istovremeno vlasnik vikendice/broda ili instruktor pokuša da napravi termin za brzu rezervaciju u istom ili preklapajućem periodu za koji klijent pokuša da rezerviše isti entitet. Kako ove funkcionalnosti imaju identičan način validacije doći će do konfliktne situacije slične prvoj.

Na dijagramu prikazan je za slučaj rezervacije i kreiranja akcije kod brodova, isti postupak je i za ostale entitete. Klijent i vlasnik istovremeno upućuju zahtev BoatController-u, ali ovaj put su zahtevi različiti. Klijent šalje zahtev za rezervaciju, a vlasnik šalje zahtev za kreiranje termina za brzu rezervaciju. Kontroler obrađuje zahteve i komunicira sa bazom. Problem nastaje kod provere dostupnosti termina gde oba dobijaju odgovor da je termin slobodan. Dalje prilikom čuvanja podataka u bazi nastaje problem jer će vikendica biti zauzeta i imati slobodnu akciju u istom ili preklapajućem terminu.

### Solucija:

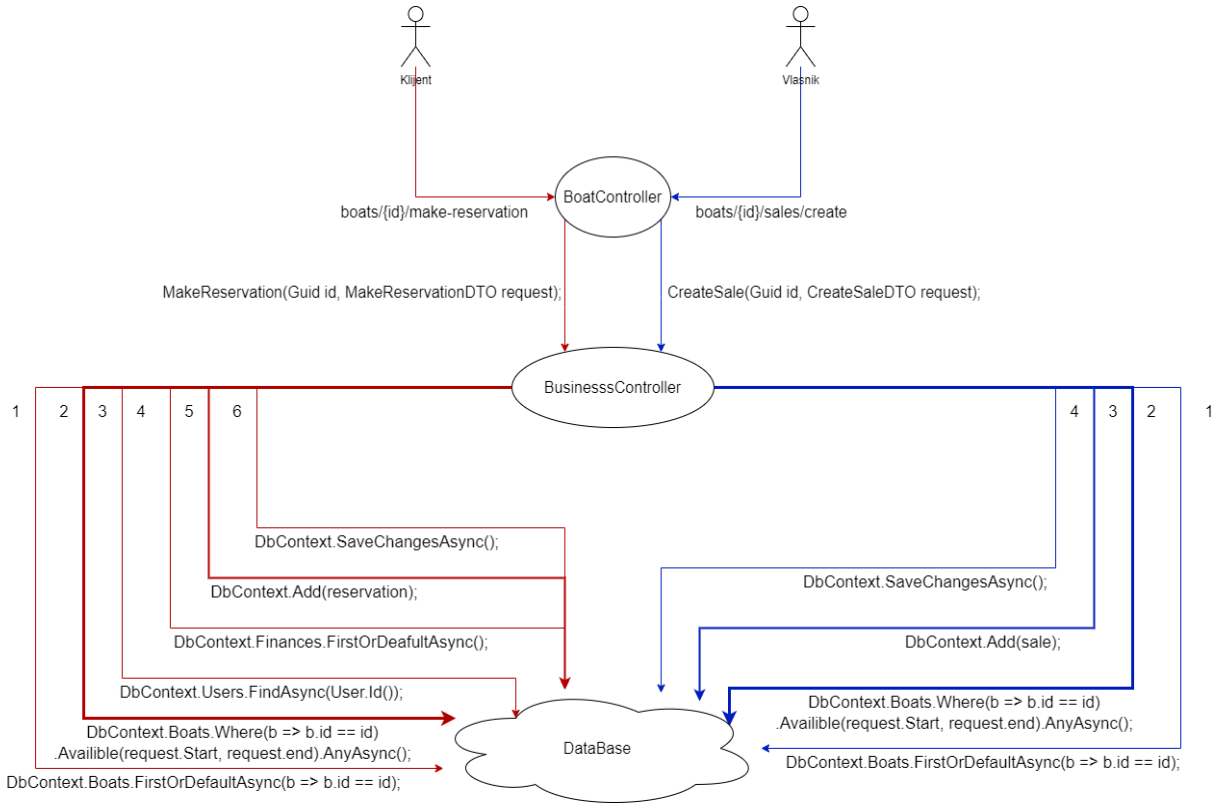
Problem rešavamo postavljanjem pesimističkog zaključavanja na metodu za kreiranje rezervacija kao i na metodu za produzavanje rezervacije, sa nivoom izolacije *Serializable.* Time će samo jedan korisnik moći da pristupi entitetu, dok će drugi morati da pokušaju ponovo. Pesimistično zaključavanje se koristi kako bi se osiguralo korišćenje najnovijih podataka pri proveri dostupnosti.

```
await using var transaction = await Context.Database.BeginTransactionAsync(IsolationLevel.Serializable);
try
{
    var business = await Context.Set<TBusiness>()
            .Include(b => b.Owner)
            .Include(b => b.Services)
            .Include(b => b.Subscribers)
            .FirstOrDefaultAsync(b => b.Id == id);

    if (business is null)
    {
        return BadRequest($"The specified {BusinessType} does not exist.");
    }
```

```
    }
    catch (Exception)
    {
        await transaction.RollbackAsync();
        return BadRequest();
    }
}
```

Tok zahteva



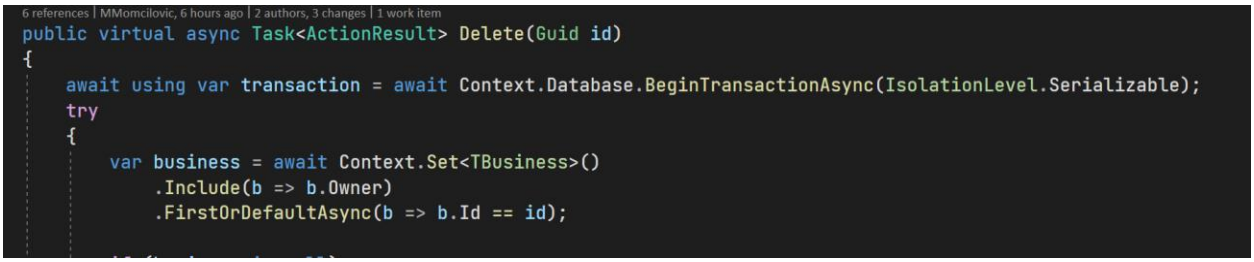

# Konflikt 3

## Klijent ne može da rezerviše vikendicu istovremeno kada vlasnik isti entitet briše

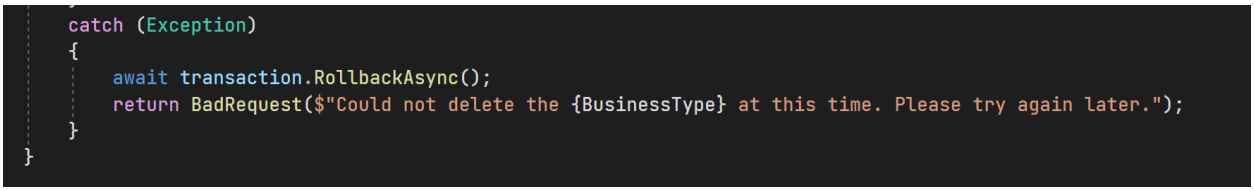
Treća konfliktna situacija nastaje kada klijent pokuša da rezerviše entitet dok u isto vreme vlasnik entiteta pokušava isti da obriše. Na dijagramu je prikazan slučaj za vikendicu i njegovog vlasnika, ali je za ostala dva entiteta slučaj ekvivalentan. Klijent i vlasnik istovremeno upućuju zahtev CabinController-u. Problem nastaje pri proveri postojanja entiteta koji klijent želi da rezerviše obzirom da on još uvek postoji u bazi. Nastaje konfliktna rezervacija jer ne može da postoji aktivna rezervacija entiteta koji je obrisan.

## Solucija:

Problem rešavamo postavljanjem pesimističkog zaključavanja na metodu za kreiranje rezervacija kao i na metodu za brisanje entiteta, sa nivoom izolacije *Serializable.* Time će samo jedan korisnik moći da pristupi entitetu, dok će drugi morati da pokušaju ponovo. Pesimistično zaključavanje se koristi kako bi se osiguralo korišćenje najnovijih podataka pri proveri dostupnosti.

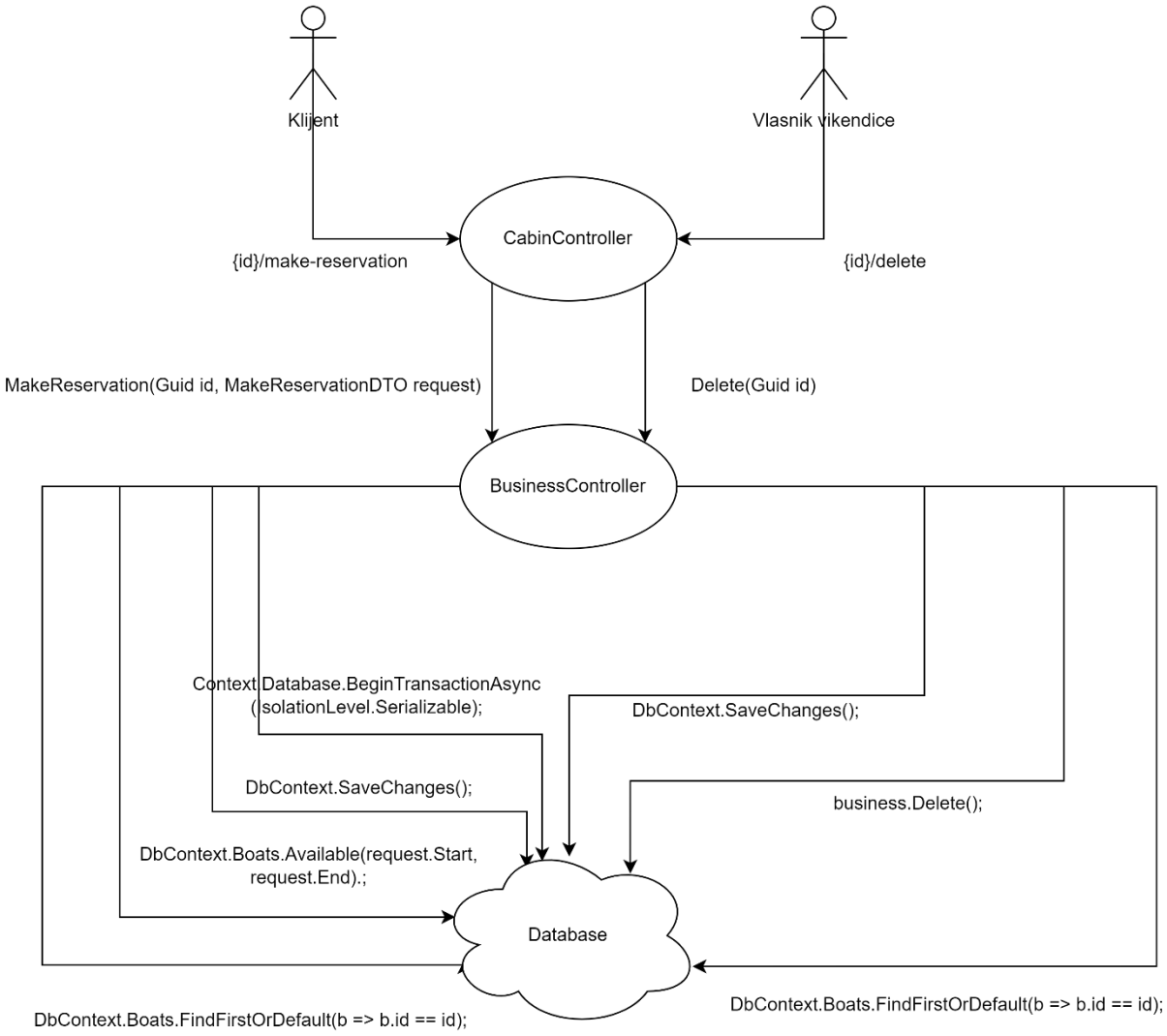


```
6 references | MMomcilovic, 6 hours ago | 2 authors, 3 changes | 1 work item
public virtual async Task<ActionResult> Delete(Guid id)
{
    await using var transaction = await Context.Database.BeginTransactionAsync(IsolationLevel.Serializable);
    try
    {
        var business = await Context.Set<TBusiness>()
            .Include(b => b.Owner)
            .FirstOrDefaultAsync(b => b.Id == id);
```



```
    catch (Exception)
    {
        await transaction.RollbackAsync();
        return BadRequest($"Could not delete the {BusinessType} at this time. Please try again later.");
    }
}
```

## Tok zahteva


Klijent
Vlasnik vikendice
CabinController
{id}/make-reservation
{id}/delete
MakeReservation(Guid id, MakeReservationDTO request)
Delete(Guid id)
BusinessController
Context.Database.BeginTransactionAsync
(IsolationLevel.Serializable);
DbContext.SaveChanges();
DbContext.SaveChanges();
business.Delete();
DbContext.Boats.Available(request.Start,
request.End).;
Database
DbContext.Boats.FindFirstOrDefault(b => b.id == id);
DbContext.Boats.FindFirstOrDefault(b => b.id == id);