Konkurentan pristup resursima u bazi

Konflikt 1

Više istovremenih klijenata ne mogu da naprave rezervaciju istog entiteta u isto (ili preklapajuće) vreme

Prva konfliktna situacija jeste situacija u kojoj više korisnika istovremeno pokušava da rezerviše isti entiteta. Ukoliko bi oba korisnika za isto ili barem preklapajuće vreme odradila proveru dostupnosti jednog entiteta, obe provere bi vratile validan rezultat, te bismo bili u riziku *dualne zauzetosti* jednog objekta ili avanture. U idealnoj situaciji, ovaj problem bio bi regulisan time što bi samo jedan korisnik uspešno izvršio rezervisanje entiteta, dok bi drugi bili obavešteni o grešci.

Kako se ovaj problem javlja na tri ekvivalentna mesta, tj. prilikom rezervacije brodova, vikendica i avantura, dijagramom ćemo prikazati prvu situaciju. Postupak se odvija ekvivalentno za druge tipove, razlika je u endpointu i kontroleru koji se poziva. Radi pojednostavljenja dijagram i očuvanja konteksta problema, proces slanja mejla o rezervaciji je izostavljen *[Strana 2, ilustracija 1].*

Problem se dešava jer dva paralelna rezervisanja nisu svesna jedno drugog. Metoda proverava da li je entitet slobodan u izabranom terminu, međutim ona ne može znati za paralelno rezervisanje koje bi trebalo sprečiti.

Rešenje:

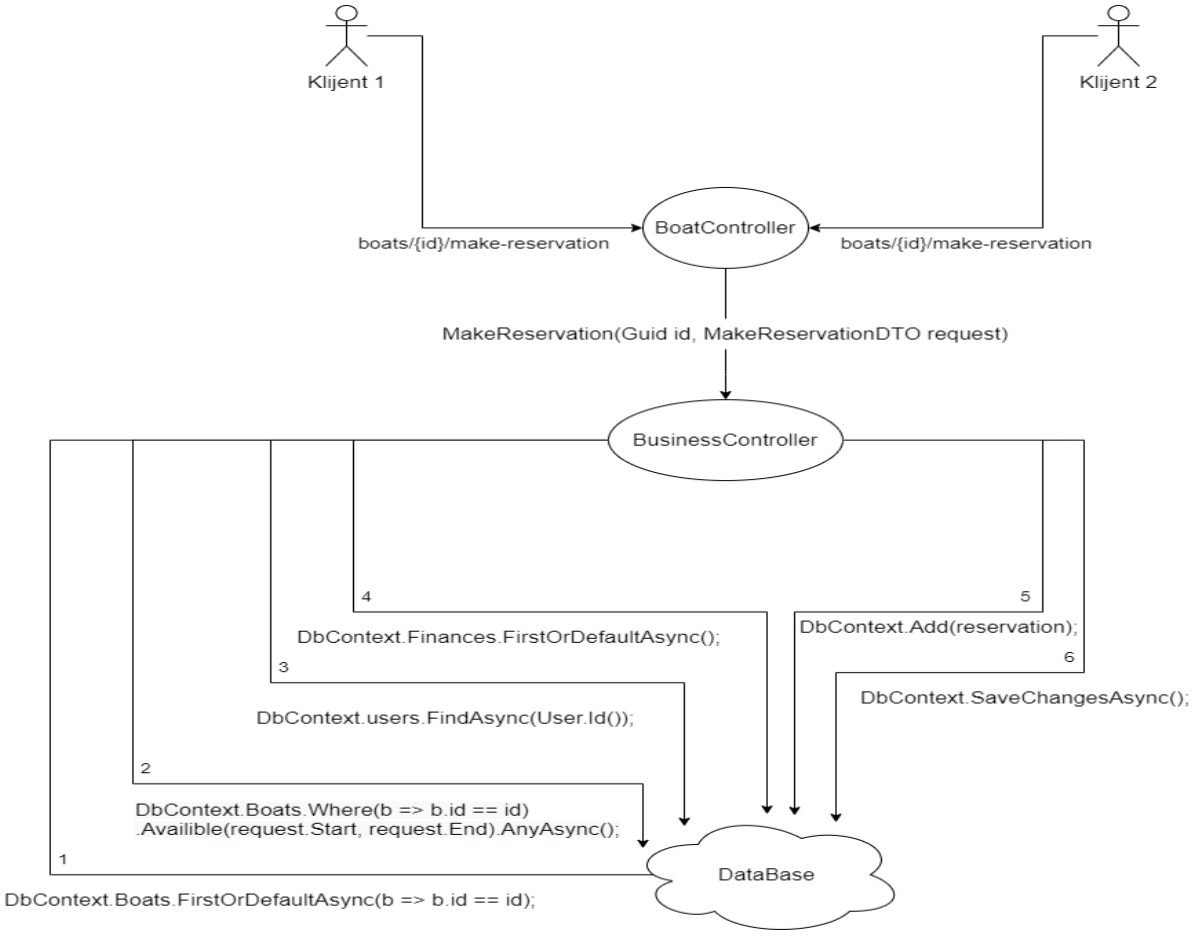
Problem rešavamo postavljanjem pesimističkog zaključavanja na metodu za kreiranje rezervacija, sa nivoom izolacije *Serializable.* Time će samo jedan korisnik moći da pristupi entitetu, dok će drugi biti informisani o grešci i moraće da pokušaju ponovo. Pesimistično zaključavanje održava konzistentnost u bazi tako što onemogućava istovremeno zauzimanje termina nekog entiteta.

```
// Conflicting Situation 1.1
[Authorize(Roles = Role.Customer)]
[HttpPost("{id}/make-reservation")]
0 references
public async Task<ActionResult> MakeReservation([FromRoute] Guid id, [FromBody] MakeReservationDTO request)
{
    await using var transaction = await Context.Database.BeginTransactionAsync(IsolationLevel.Serializable);
    try
    {
        await Context.SaveChangesAsync();
        await transaction.CommitAsync();

        _mailer.Send(user, new ReservationCreated(
            reservation,
            ImageUrl(reservation.Business.Id, reservation.Business.Images.FirstOrDefault()),
            "#contactUrl"
         ));

        return Ok(reservation.Id);
    }
    catch (Exception)
    {
        await transaction.RollbackAsync();
        return BadRequest($"Could not make a reservation at this time. Please try again later.");
    }
}
```

Tok zahteva



Ilustracija 1- Rezervacija broda

Konflikt 2

Više istovremenih klijenata ne mogu da naprave rezervaciju istog entiteta na akciji u isto vreme

Kako su akcijske ponude dostupne svim prijavljenim klijentima za brzu rezervaciju, lako je moguće da dva klijenta u isto, ili približno isto vreme pokušaju da rezervišu jednu ponudu. Ovaj slučaj doveo bi do nekonzistentnosti baze sa mogućnostima realnog sistema, te je u cilju isti izbeći.

Na dijagramu prikazan je tok zahteva prilikom pravljenja brze rezervacije *[Strana 4, ilustracija 2].*
Problem se dešava jer sistem ne zna za paralelne pokušaje rezervisanja jedne akcije.

Rešenje:

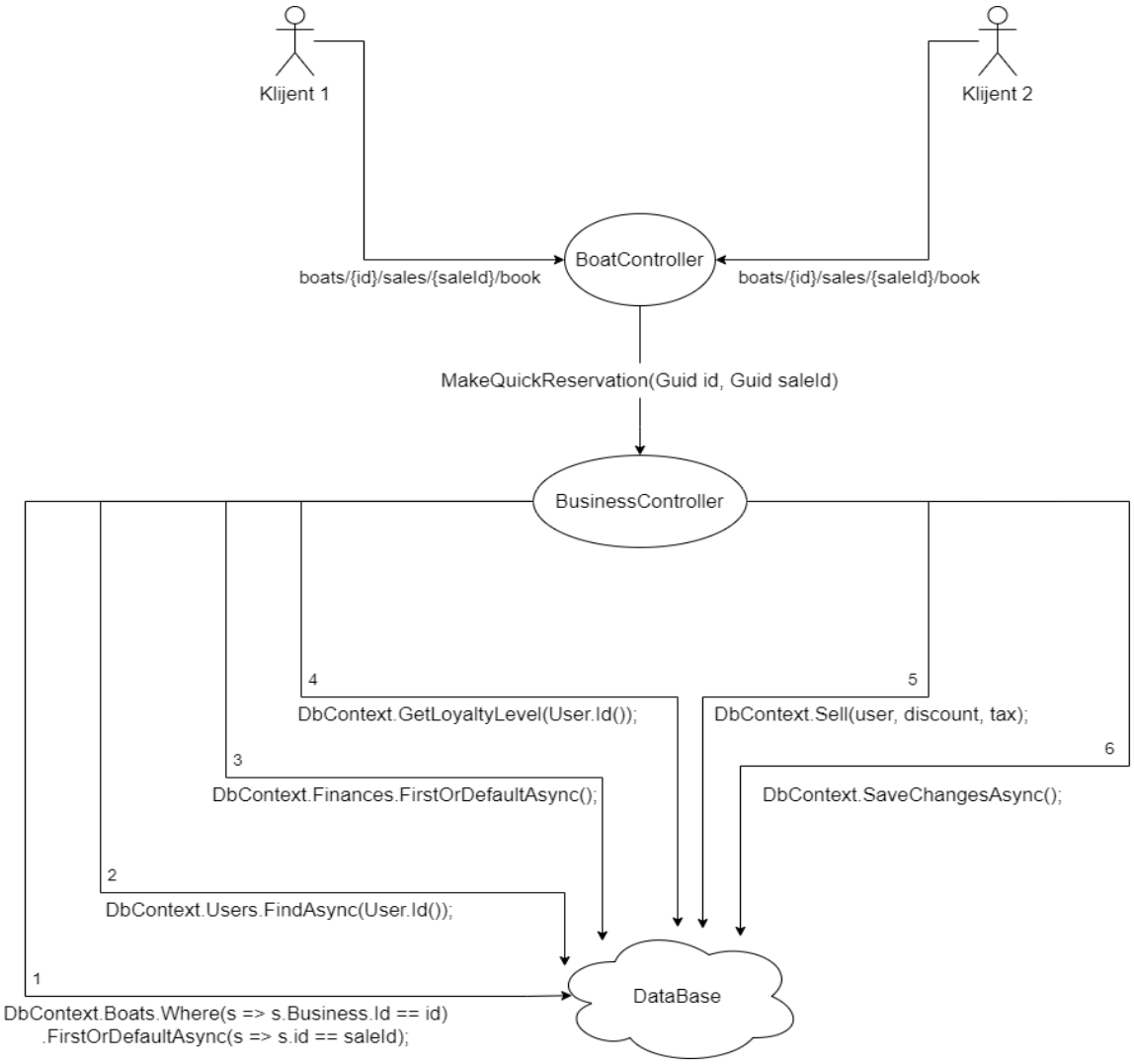
Problem rešavamo postavljanjem pesimističkog zaključavanja na metodu za kreiranje rezervacije na akciji, sa nivoom izolacije *Serializable.* Time će samo jedan korisnik moći da pristupi entitetu, dok će drugi morati da pokušaju ponovo. Ukoliko klijent pokuša da napravi rezervaciju entiteta na akciji dok traje druga transakcija njegovog rezervisanja, biće informisan o grešći i moraće da pokuša ponovo nakon završetka transakcije.

```
// Conflicting Situation 1.2
[Authorize(Roles = Role.Customer)]
[HttpPost("{id}/sales/{saleId}/book")]
0 references
public async Task<ActionResult> MakeQuickReservation([FromRoute] Guid id, [FromRoute] Guid saleId)
{
    await using var transaction = await Context.Database.BeginTransactionAsync(IsolationLevel.Serializable);
    try
    {
        sale.Sell(user, loyalty?.DiscountPercentage ?? 0, finance.TaxPercentage);
        user.LoyaltyPoints += finance.CustomerPoints;
        await Context.SaveChangesAsync();
        await transaction.CommitAsync();

        _mailer.Send(user, new ReservationCreated(
            sale,
            ImageUrl(sale.Business.Id, sale.Business.Images.FirstOrDefault()),
            "#contactUrl"
         ));

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

Tok zahteva



Ilustracija 2 – Brza rezervacija broda

Konflikt 3

Klijent ne može da napravi rezervaciju istovremeno dok vlasnik definiše nedostupnost svog objekta ili avanture u isto ili preklapajuće vreme

Svaki vlasnik u bilo kom momentu može da definiše nedostupnost svog objekta ili avanture. Definisanjem nedostupnosti, svi klijentima onemogućava rezervisanje objekta ili avanture u izabranom periodu. Prilikom definisanja nedostupnosti može se desiti konflikt ukoliko klijent u isto vreme pokuša da napravi rezervaciju u istom ili preklapajućem terminu. Na dijagramu je pokazan samo slučaj istovremenog brisanja i rezervisanja broda, ali za ostale entitete tok zahteva je ekvivalentan.

Ovaj problem može se rešiti optimističnim zaključavanjem.

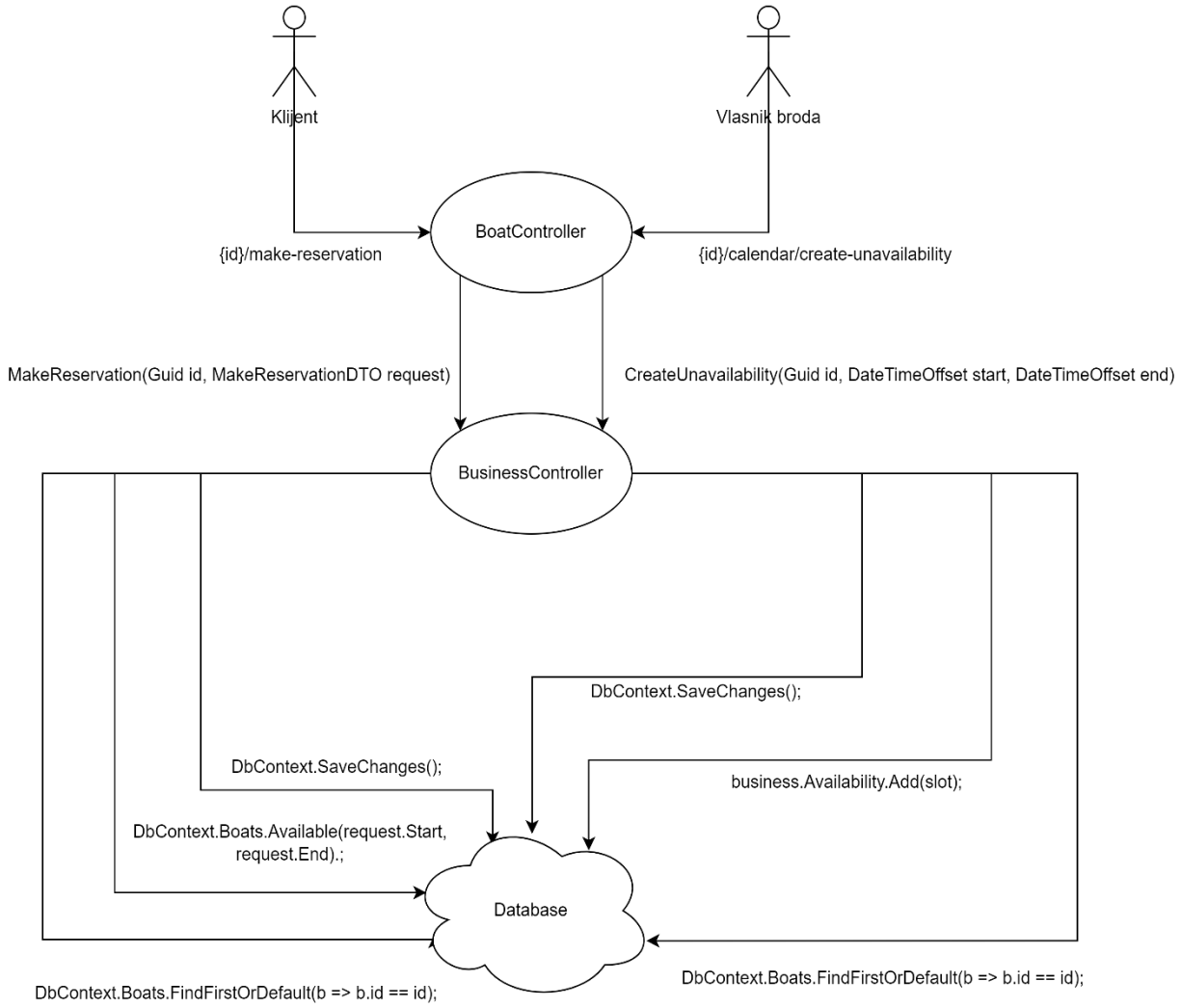
Rešenje:

Problem rešavamo postavljanjem pesimističkog zaključavanja na metodu za kreiranje perioda nedostupnosti, sa nivoom izolacije *Serializable.* Time će samo jedan korisnik, ili vlasnik ili klijent, onaj koji je započeo izvršavanje zahteva prvi, moći da promeni podatke. Ovim postupkom osiguravamo da neće doći do nekonzistentnog stanja u bazi.

```
// Conflicting Situation 1.3
[HttpPost("{id}/calendar/create-unavailability")]
6 references
public virtual async Task<ActionResult> CreateUnavailability(Guid id, DateTimeOffset start, DateTimeOffset end)
{
    await using var transaction = await Context.Database.BeginTransactionAsync(IsolationLevel.Serializable);
    try
    {
        business.Availability.Add(slot);
        await Context.SaveChangesAsync();
        await transaction.CommitAsync();

        return Ok(new
        {
            slot.Id,
            slot.Start,
            slot.End,
            Type = "unavailable",
            Name = "Unavailable"
        });
    }
    catch (Exception)
    {
        await transaction.RollbackAsync();
        return BadRequest("Unavailability could not be created at this time. Please try again later.");
    }
}
```

Tok zahteva



Ilustracija 3- Konflikt pri rezervisanju i definisanju nedostupnosti