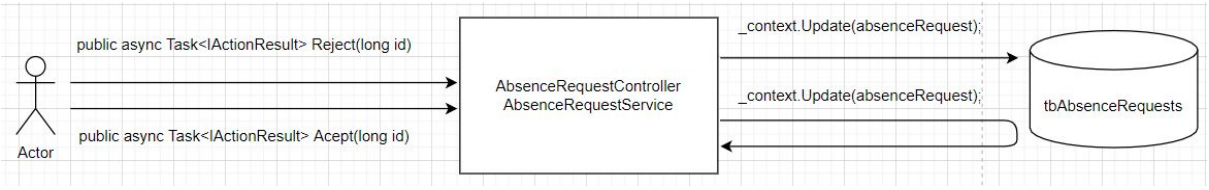
Konkurentni pristup resursima u bazi

Filip Zdelar IN11-2017, internet sofrverska arhitektura

1. Prihvatanje zahteva za godišnji odmor

Funkcionalnost: Medicinski radnik radnik šalje zahtev za godišnji odmor u određenom vremenskom intervalu za koji nema zakazan pregled. Administrator apoteke u zavisnosti od stanja daje odgovor, da potvrđuje ili odbija zahtev. Zahtev se time ažurira i smešta u bazu.

Problem: Prilikom vešestukog odgovora na zahtev za godišnji odmor, može doći do štetnog preplitanja niti koje su menjale stanje objeta u tom sistemu. Ukoliko jedna nit promeni stanje objekta, druga nit može da prepiše zatečeno stanje, čineći enitet baze podataka u suštinski invalidnom nekonzistentnom stanju.



Rešenje: Za realizaciju ovog rešenja odlučeno je da se prvi poslat zahtev za promeni stanja tipa entiteta prihvati kao konačno stanje baze. To je omogućeno vođenjem evidentcije o redosledu upisanog sloga, pa ukoliko se pojavi zahtev sa istim evidecionim brojem, niti onemogućava se pristup i proverava se postojanje datog identifikacionog obeležja u bazi. Razlog ove realizacije je je što su ovi pozivi ređe učestalosti od drug poziva sistema, a i iz razloga što ga odobrava isti administrator apoteke, te drugi pozive može da se protumači kao greška korisnika. Prihvatiti zahtev za odmor, iako ga je trebalo odbiti, je nefektivnija alternativa, zato što će uticati na ostala prepačunvanja, dok ukoliko je zahtev odbijem, a nije trebao, administrator apoteke ima mogućnost da obrazloži mejlom koji sledi odbijanje godišnjeg odmora. Ova varijacija bi se mogla dodatno uzeti u obir, iako u konkretnoj implementaciji sotfvera nije. [*endpoint* /AbsenceRequests/Accept/{Id}]

```
public async Task<IActionResult> Reject(long id)
{
    var absenceRequest = _context.tbAbsenceRequests.Find(id);
    absenceRequest.Approved = false;

    try
    {
        _context.Update(absenceRequest);
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!AbsenceRequestExists(absenceRequest.Id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }
    return RedirectToAction("RejectText", "AbsenceRequests", new { id = absenceRequest.Id});
}
```

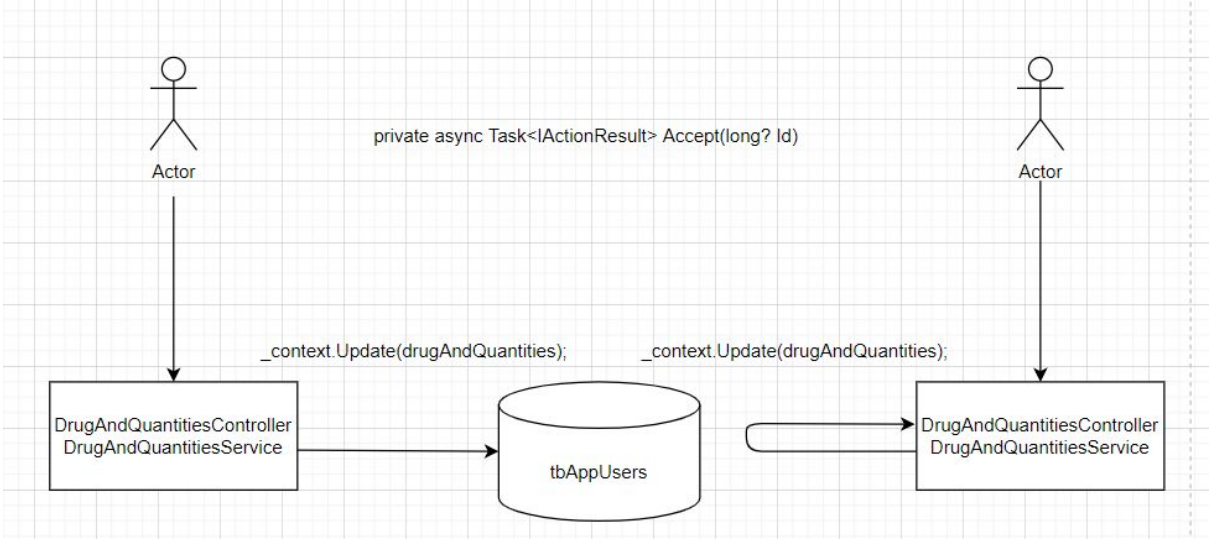
```
public async Task<IActionResult> Approve(long id)
{

    var absenceRequest = await _context.tbAbsenceRequests.FindAsync(id);
    absenceRequest.Approved = true;

        try
        {
            _context.Update(absenceRequest);

            var user = await _userManager.GetUserAsync(User);
            await _emailSender.SendEmailAsync(user.Email, "Absence Requiest Respons of Admin",
                $"Your absence Requiest has been approved");
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!AbsenceRequestExists(absenceRequest.Id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
}
```

2. Vođenje evidencije o količini leka u apoteci

Funkcionalnost: Administrator apoteke ima mogućnost da naruči određenu količinu lekova i samim tim poveća trenutnu vresnot leka u apoteci. Korisnici naručuju lekove i smanjuju isti broj lekova iz apoteke.

Problem: Prilikom menjanja količine leka može se dogoditi da administrator apoteke upiše povećanu vresnot, pre nego što ju je korisnik smanjio na

određenu količinu. Samim tim se stvorio višak lekova, a manjak sredstava u apoteci. Takođe postoje štetna preplitanja između dva korisnika, kada umesto da se smanji količina zbiru naručenih lekova, smanji naizmeniča količina nekog pacijenta. Štetno preplitnanje među administartorima nije moguđe usped jedinstvenosti administratora po apoteci.



Rešenje: Za realizaciju ovog rešenja odlučen je optimističan pristup (isto kao i u prethodnom problemu). Dodaje se novi atrubut - niz bajtova koji vodi evidenciju o rednom broju upisanog sloga u bazu podataka. Prilikom svakog upisa

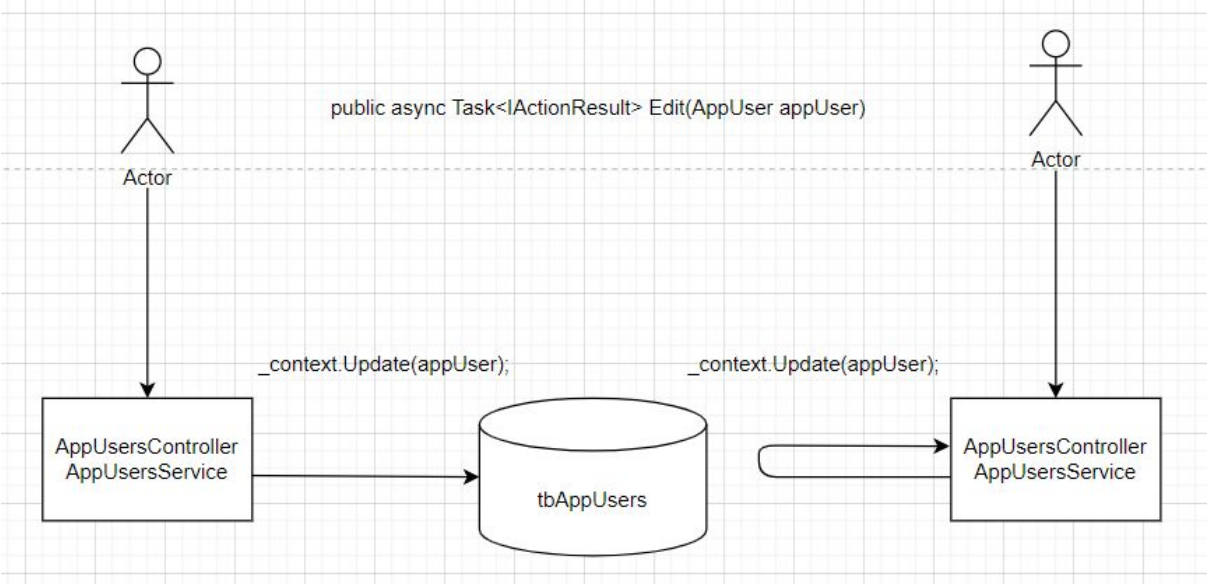
[*endpoint* /SupplyOffers/Accept/{Id}]

```
for (int j = 0; j < allSupplyItems.Count(); j++)
{
    for (int k = 0; k < drugsAndQ.Count(); k++)
    {
        if (allSupplyItems[j].DrugId == drugsAndQ[k].Drug.Id)
        {
            drugsAndQ[k].Quantity += (uint)allSupplyItems[j].ExtraQuantity;

            try
            {
                _ = await _context.AddAsync(drugsAndQ[k]);
                _ = await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                return NotFound();
            }
            break;
        }
    }

}
```

3. Menjanje ličnih podataka farmaceuta

Funkcionalnost: Svaki administrator sistema može da menja podatke farmaceuta koji rade u njegovoj apoteci. Farmacuti mogu da menjaju svoje lične podatke.

Problem: Prilikom menjanja ličnih podataka, može se dogoditi da farmaceut prepiše podatke apotekara, kao i da apotekar može da prepiše podatke farmacuta.



Rešenje: Za dat problem opredeljen je optimističan pristup bazi podataka. Mogućnost preplitanja za ovu vrstu je izuzetno mala i zbog toga je opredeljno za ovaj pristup. Implementacija ovog rešenja je slična kao i u prethodna dva problema.
[*endpoint* /AppUsers/Edit/{Id}]