Konkurentan pristup resursima u bazi

Srđan Šuković RA3/2018 - student 3

Rešene konfliktne situacije:

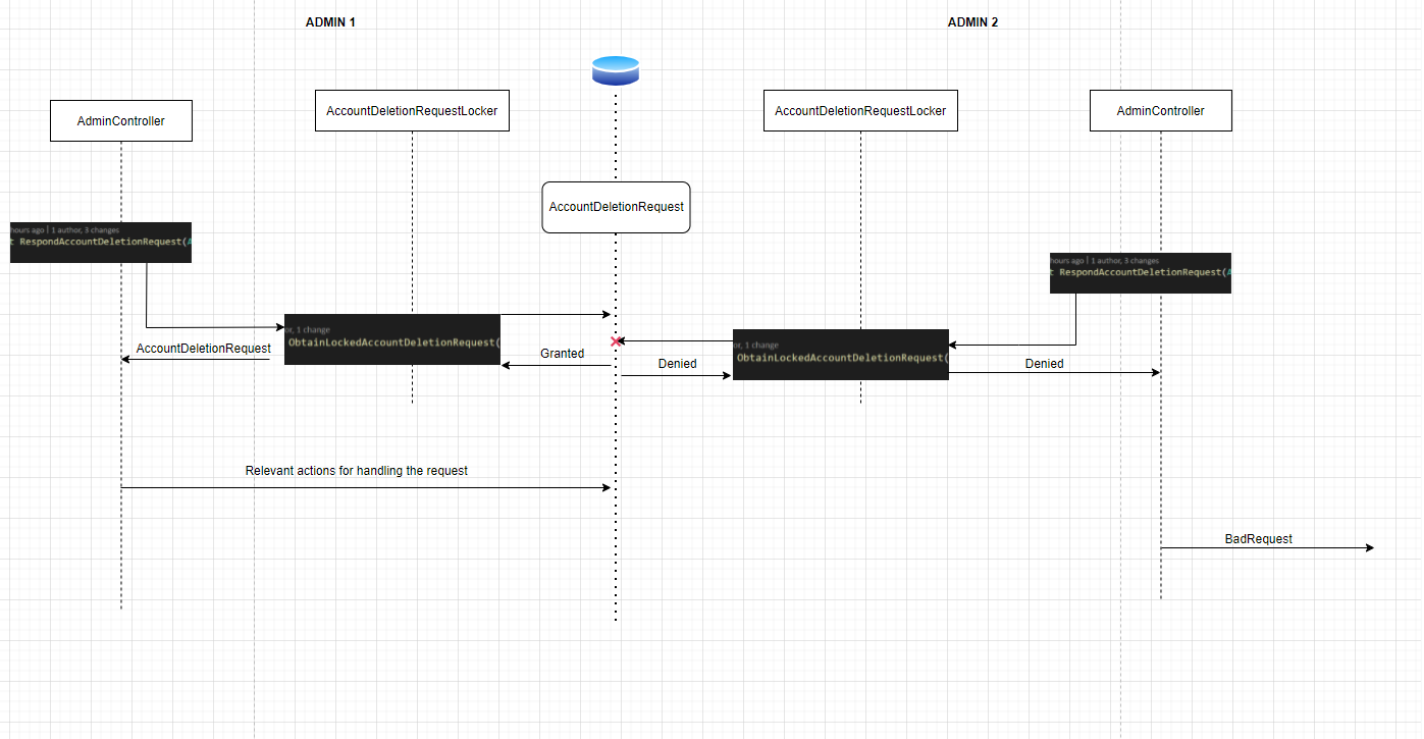
- Na jedan zahtev za brisanje naloga može da odgovori samo jedan administrator sistema
- Na jednu žalbu može da odgovori samo jedan administrator sistema
- Na jedan zahtev za odobrenje ocene može da odgovori samo jedan administrator sistema

Na jedan zahtev za brisanje naloga može da odgovori samo jedan administrator sistema

- **Opis scenarija**

Administrator sistema ima pristup zahtevima korisnika za brisanje naloga. Ima opciju da ih prihvati ili odbije. Na kraju obe akcije se šalje potvrda korisniku na mejl. Ukoliko bi dva administratora sistema u isto vreme pokušala da interaguju sa istim zahtevom za brisanje naloga, došlo bi do nepredviđenog ponašanja i stoga se ovakav slučaj mora regulisati.

- **Crtež tokova**



- **Opis rešenja**

Opisan problem je rešen primenom pesimističkog zaključavanja. Implementacija rešenja ovog problema nalazi se u klasama:

- **AdminController** - metoda **RespondAccountDeletionRequest**
- **AccountDeletionRequestLocker** - metoda **ObtainLockedAccountDeletionRequest**

Neophodno je zaključati AccountDeletionRequest kada se radi sa njime, kako nijedna druga nit ne bi mogla da nastavi da radi dalje sa konkretno tim servisom. Implementirana je klasa AccountDeletionRequestLocker.

```
namespace Services
{
    2 references | srdjansukovic, 2 days ago | 1 author, 1 change
    public class AccountDeletionRequestLocker
    {
        private readonly IUnitOfWork uow;
        private int timeout;
        private readonly int pollingInterval;

        1 reference | srdjansukovic, 2 days ago | 1 author, 1 change
        public AccountDeletionRequestLocker(IUnitOfWork uow)
        {
            this.uow = uow;
            timeout = uow.GetRepository<ISystemConfigReadRepository>().GetValue<int>("LockingInterval");
            pollingInterval = uow.GetRepository<ISystemConfigReadRepository>().GetValue<int>("PollingInterval");
        }

        1 reference | srdjansukovic, 2 days ago | 1 author, 1 change
        public AccountDeletionRequest ObtainLockedAccountDeletionRequest(int userId)
        {
            while (timeout != 0)
            {
                timeout -= pollingInterval;
                try
                {
                    var request = uow.GetRepository<IAccountDeletionRequestReadRepository>()
                        .GetById(userId, LockMode.UpgradeNoWait);
                    return request;
                }
                catch
                {
                    Thread.Sleep(pollingInterval);
                }
            }

            throw new ADOException();
        }
    }
}
```

Ideja iza ovog servisa jeste, da ukoliko neko želi da radi sa zahtevom, mora prvo da dobije ključ za njega, ili da ga zaključa. U bazi su upisane dve vrednosti – timeout i polling interval. Timeout predstavlja maksimalno čekanje neke niti pre pucanja. Polling interval je vreme koje nit čeka kako bi opet pokušala da otključa AccountDeletionRequest.

Aplikacija koristi Microsoft SQL bazu i ona ima ugrađenu podršku za Lockmode.Upgrade, ali odlučili smo da implementiramo ovako ukoliko bi se nekad odradio prelazak na neku drugu bazu koja nema ugrađenu podršku za ovo.

Primer korišćenja ovog AccountDeletionRequestLocker-a za zaključavanje zahteva:

```
try
{
    oldRequest = new AccountDeletionRequestLocker(UoW).ObtainLockedAccountDeletionRequest(request.UserId);
}
catch
{
    return BadRequest(Responses.UnavailableRightNow);
}
```

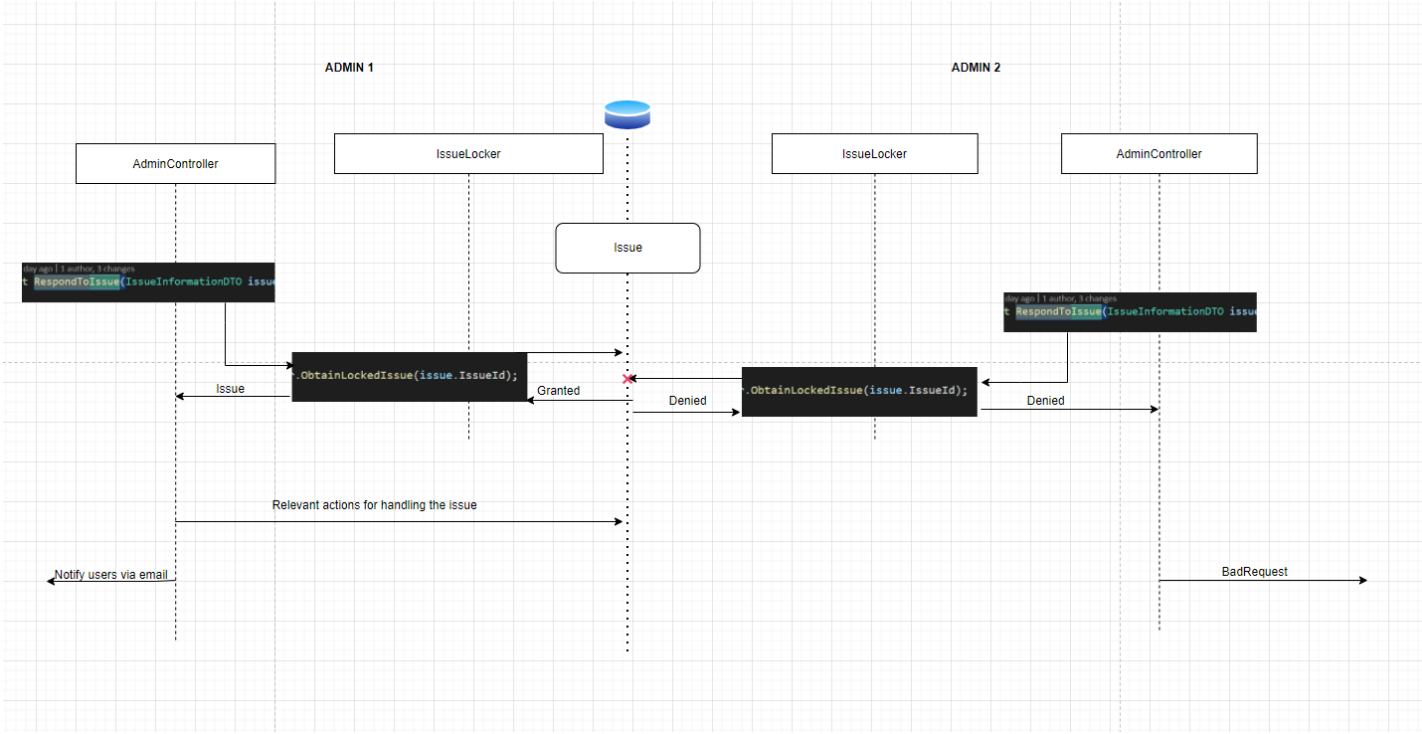
Ukoliko servis ne uspe da dobavi ključ za dati zahtev za brisanje, korisnik će dobiti poruku u kojoj biva obaveštavan da pokuša kasnije, s obzirom da je prevelika “gužva” za dati zahtev.

Na jednu žalbu može da odgovori samo jedan administrator sistema

- **Opis scenarija**

Administrator sistema ima pristup žalbama korisnika. Ima opciju da odgovara na njih. Na kraju akcije se šalje odgovor korisnicima na mejl. Ukoliko bi dva administratora sistema u isto vreme pokušala da interaguju sa istom žalbom, došlo bi do nepredviđenog ponašanja, stoga se ovakav slučaj mora regulisati.

- **Crtež tokova**



- **Opis rešenja**

Opisan problem je rešen primenom pesimističkog zaključavanja. Implementacija rešenja ovog problema nalazi se u klasama:

- **AdminController** - metoda **RespondToIssue**

– **IssueLocker** - metoda **ObtainLockedIssue**

Neophodno je zaključati Issue kada se radi sa njime, kako nijedna druga nit ne bi mogla da nastavi da radi dalje sa konkretno tim servisom. Implementirana je klasa IssueLocker.

```
2 references | srdjansukovic, 1 day ago | 1 author, 1 change
public class IssueLocker
{
    private readonly IUnitOfWork uow;
    private int timeout;
    private readonly int pollingInterval;

    1 reference | srdjansukovic, 1 day ago | 1 author, 1 change
    public IssueLocker(IUnitOfWork uow)
    {
        this.uow = uow;
        timeout = uow.GetRepository<ISystemConfigReadRepository>().GetValue<int>("LockingInterval");
        pollingInterval = uow.GetRepository<ISystemConfigReadRepository>().GetValue<int>("PollingInterval");
    }

    1 reference | srdjansukovic, 1 day ago | 1 author, 1 change
    public Issue ObtainLockedIssue(int issueId)
    {
        while (timeout != 0)
        {
            timeout -= pollingInterval;
            try
            {
                var issue = uow.GetRepository<IIssueReadRepository>()
                    .GetById(issueId, LockMode.UpgradeNoWait);
                return issue;
            }
            catch
            {
                Thread.Sleep(pollingInterval);
            }
        }

        throw new ADOException();
    }
}
```

Ideja iza ovog servisa jeste, da ukoliko neko želi da radi sa žalbom, mora prvo da dobije ključ za nju, ili da je zaključa. U bazi su upisane dve vrednosti – timeout i polling interval. Timeout predstavlja maksimalno čekanje neke niti pre pucanja. Polling interval je vreme koje nit čeka kako bi opet pokušala da otključa Issue.

Primer korišćenja ovog AccountDeletionRequestLocker-a za zaključavanje zahteva:

```
try
{
    oldIssue = issueLocker.ObtainLockedIssue(issue.IssueId);
}
catch
{
    return BadRequest(Responses.UnavailableRightNow);
}
```

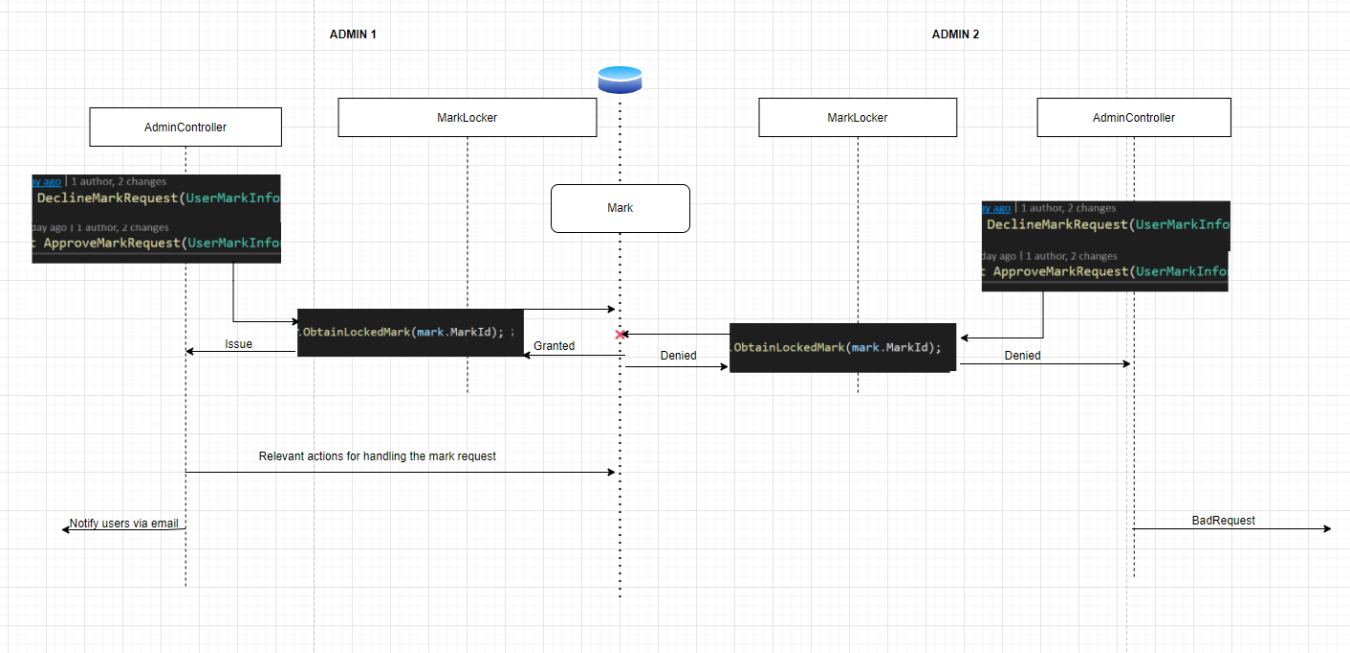
Ukoliko servis ne uspe da dobavi ključ za dati za datu žalbu, korisnik će dobiti poruku u kojoj biva obaveštavan da pokuša kasnije, s obzirom da je prevelika "gužva" za datu žalbu.

Na jednan zahtev za odobrenje ocene može da odgovori samo jedan administrator sistema

- **Opis scenarija**

Administrator sistema ima pristup zahtevima za odobrenje ocene korisnika. Ima opciju da odgovara na njih. Na kraju potvrdne akcije se šalje odgovor korisnicima na mejl. Ukoliko bi dva administratora sistema u isto vreme pokušala da interaguju sa istim zahtevom za ocenu, došlo bi do nepredviđenog ponašanja, stoga se ovakav slučaj mora regulisati.

- **Crtež tokova**



- **Opis rešenja**

Opisan problem je rešen primenom pesimističkog zaključavanja. Implementacija rešenja ovog problema nalazi se u klasama:

- **AdminController –** metode **ApproveMarkRequest** i **DeclineMarkRequest**
- **MarkLocker** - metoda **ObtainLockedMark**

Neophodno je zaključati Mark kada se radi sa njime, kako nijedna druga nit ne bi mogla da nastavi da radi dalje sa konkretno tim servisom. Implementirana je klasa MarkLocker.

```
5 references | srdjansukovic, 1 day ago | 1 author, 1 change
public class MarkLocker
{
    private readonly IUnitOfWork uow;
    private int timeout;
    private readonly int pollingInterval;

    2 references | srdjansukovic, 1 day ago | 1 author, 1 change
    public MarkLocker(IUnitOfWork uow)
    {
        this.uow = uow;
        timeout = uow.GetRepository<ISystemConfigReadRepository>().GetValue<int>("LockingInterval");
        pollingInterval = uow.GetRepository<ISystemConfigReadRepository>().GetValue<int>("PollingInterval");
    }

    2 references | srdjansukovic, 1 day ago | 1 author, 1 change
    public Mark ObtainLockedMark(int markId)
    {
        while (timeout != 0)
        {
            timeout -= pollingInterval;
            try
            {
                var mark = uow.GetRepository<IMarkReadRepository>()
                    .GetById(markId, LockMode.UpgradeNoWait);
                return mark;
            }
            catch
            {
                Thread.Sleep(pollingInterval);
            }
        }

        throw new ADOException();
    }
}
```

Ideja iza ovog servisa jeste, da ukoliko neko želi da radi sa ocenom, mora prvo da dobije ključ za nju, ili da je zaključa. U bazi su upisane dve vrednosti – timeout i polling interval. Timeout predstavlja maksimalno čekanje neke niti pre pucanja. Polling interval je vreme koje nit čeka kako bi opet pokušala da otključa Mark.

Primer korišćenja ovog MarkLocker-a za zaključavanje zahteva:

```
try
{
    oldMark = markLocker.ObtainLockedMark(mark.MarkId);
}
catch
{
    return BadRequest(Responses.UnavailableRightNow);
}
```

Ukoliko servis ne uspe da dobavi ključ za dati za datu ocenu, korisnik će dobiti poruku u kojoj biva obaveštavan da pokuša kasnije, s obzirom da je prevelika “gužva” za datu ocenu.