

# Rešavanje problema konkurentnog pristupa bazi podataka

## Problem odgovaranja na zahtev korisnika sistema za brisanje naloga

Prilikom odgovaranja na žalbe korisnika sistema dozvoljeno je da samo jedan administrator može odgovoriti, pošto se konflikt može desiti ukoliko više admina istovremeno pokušaju da odgovore na istu žalbu. Za rešavanje ovog problema je implementiran optimistički pristup. U bazi podataka je dodato novo polje pod nazivom ConcurrencyToken i ono je tipa timestamp. Timestamp tip podatka je ugrađen u MSSQL i služi za ovakve situacije. Vrednost polja se menja automatski ukoliko dođe do bilo kakve promene u torci. Kada čitamo sve zahteve za deaktivaciju, prosleđuje se trenutna vrednost ovog polja, a prilikom narednom pristupu torci, poredićemo da li je prethodno učitana vrednost ista kao i trenutna. Ako je ista, znači nije došlo do ažuriranja torke, u suprotnom će okinuti grešku o tome kako je neko već ažurirao torku.

	Column Name	Data Type
🔑	Id	int
	UserName	nvarchar(128)
	UserSurname	nvarchar(128)
	EmailAddress	nvarchar(256)
	Reason	nvarchar(MAX)
	Status	int
	ConcurrencyToken	timestamp

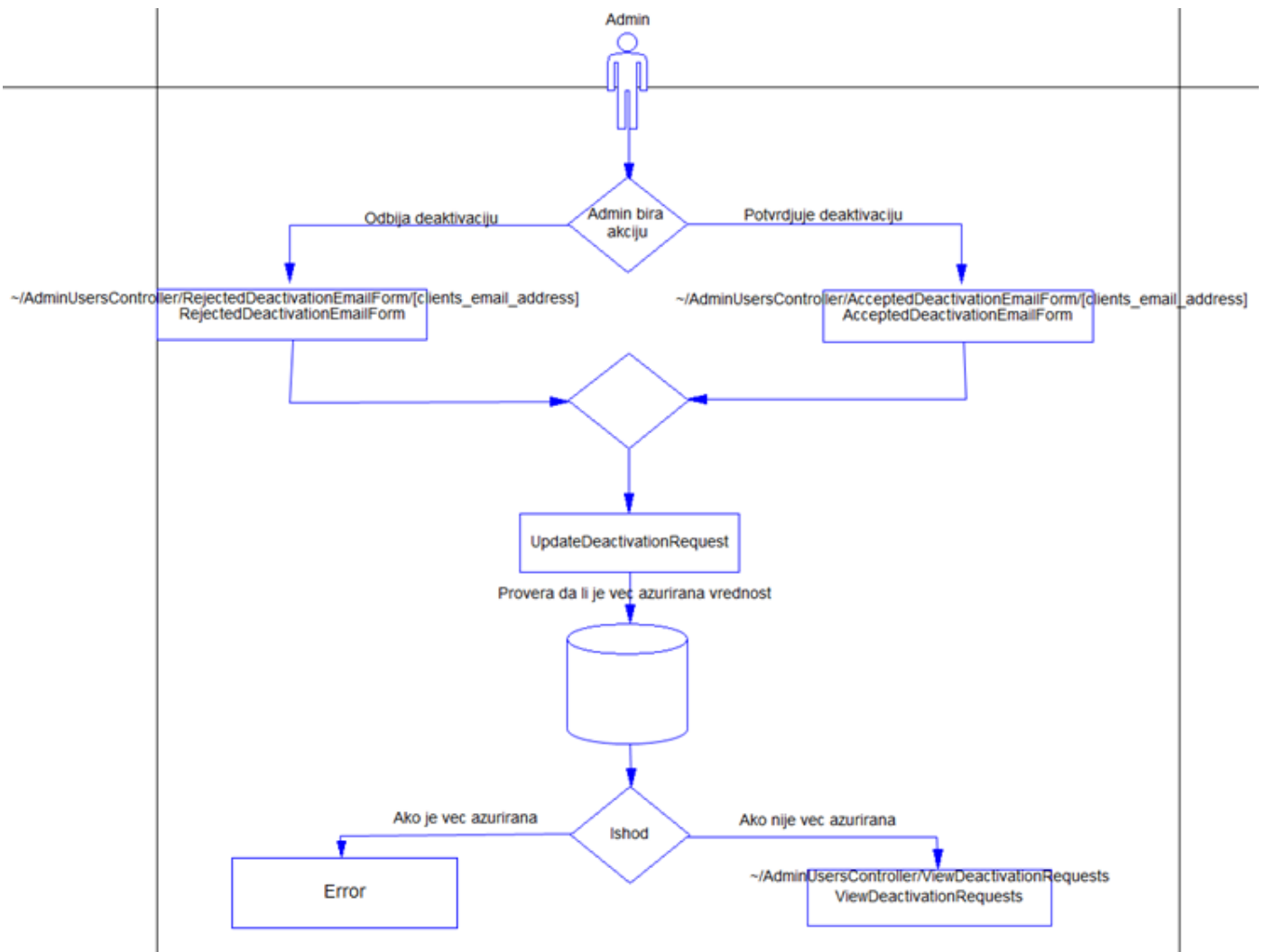
```
public class DeactivationRequest
{
    2 references
    public int Id { get; set; }
    2 references
    public string UserName { get; set; }
    2 references
    public string UserSurname { get; set; }
    8 references
    public string EmailAddress { get; set; }
    2 references
    public string Reason { get; set; }
    5 references
    public Enums.DeactivationRequestStatus Status { get; set; }
    2 references
    public byte[] ConcurrencyToken { get; set; }
}
```

Na slici ispod je prikazana metoda UpdateDeactivationRequest koja ažurira torku u bazi i ako ne uspe da nađe torku sa istom vrednošću ConcurrencyToken polja, objaviće grešku. Funkcija se poziva u akciji AcceptedDeactivationEmailForm, kontrolera AdminUsersController.

```
2 references
public static int UpdateDeactivationRequest(DeactivationRequest request)
{
    string Sql = @"UPDATE dbo.DeactivationRequests
        SET UserName = @UserName,
            UserSurname = @UserSurname,
            EmailAddress = @EmailAddress,
            Reason = @Reason, Status = @Status
        WHERE Id = @Id AND ConcurrencyToken = @ConcurrencyToken";

    var rowCount = -1;
    using (IDbConnection cnn = new SqlConnection(SSMSDataAccess.GetConnectionString()))
    {
        rowCount = cnn.Execute(Sql, request);
    }

    return rowCount;
}
```



## Problem odgovaranja na žalbe korisnika sistema

Prilikom odgovaranja na žalbe korisnika sistema dozvoljeno je da samo jedan administrator može odgovoriti, pošto se konflikt može desiti ukoliko više admina istovremeno pokušaju da odgovore na istu žalbu. Za rešavanje ovog problema je implementiran optimistički pristup. U bazi podataka je dodato novo polje pod nazivom ConcurrencyToken i ono je tipa timestamp. Timestamp tip podatka je ugrađen u MSSQL i služi za ovakve situacije. Vrednost polja se menja automatski ukoliko dođe do bilo kakve promene u torci. Kada čitamo sve zahteve za dektivaciju, prosleđuje se trenutna vrednost ovog polja, a prilikom narednom pristupu torci, poredićemo da li je prethodno učitana vrednost ista kao i trenutna. Ako je ista, znači nije došlo do ažuriranja torke, u suprotnom će okinuti grešku o tome kako je neko već ažurirao torku.

	Column Name	Data Type
🔑	Id	int
	OwnerId	nvarchar(128)
	OwnerName	nvarchar(MAX)
	OwnerSurname	nvarchar(MAX)
	OwnerEmailAddress	nvarchar(256)
	ClientsEmailAddress	nvarchar(MAX)
	ActionTitle	nvarchar(100)
	Reason	nvarchar(MAX)
	Status	int
	ConcurrencyToken	timestamp

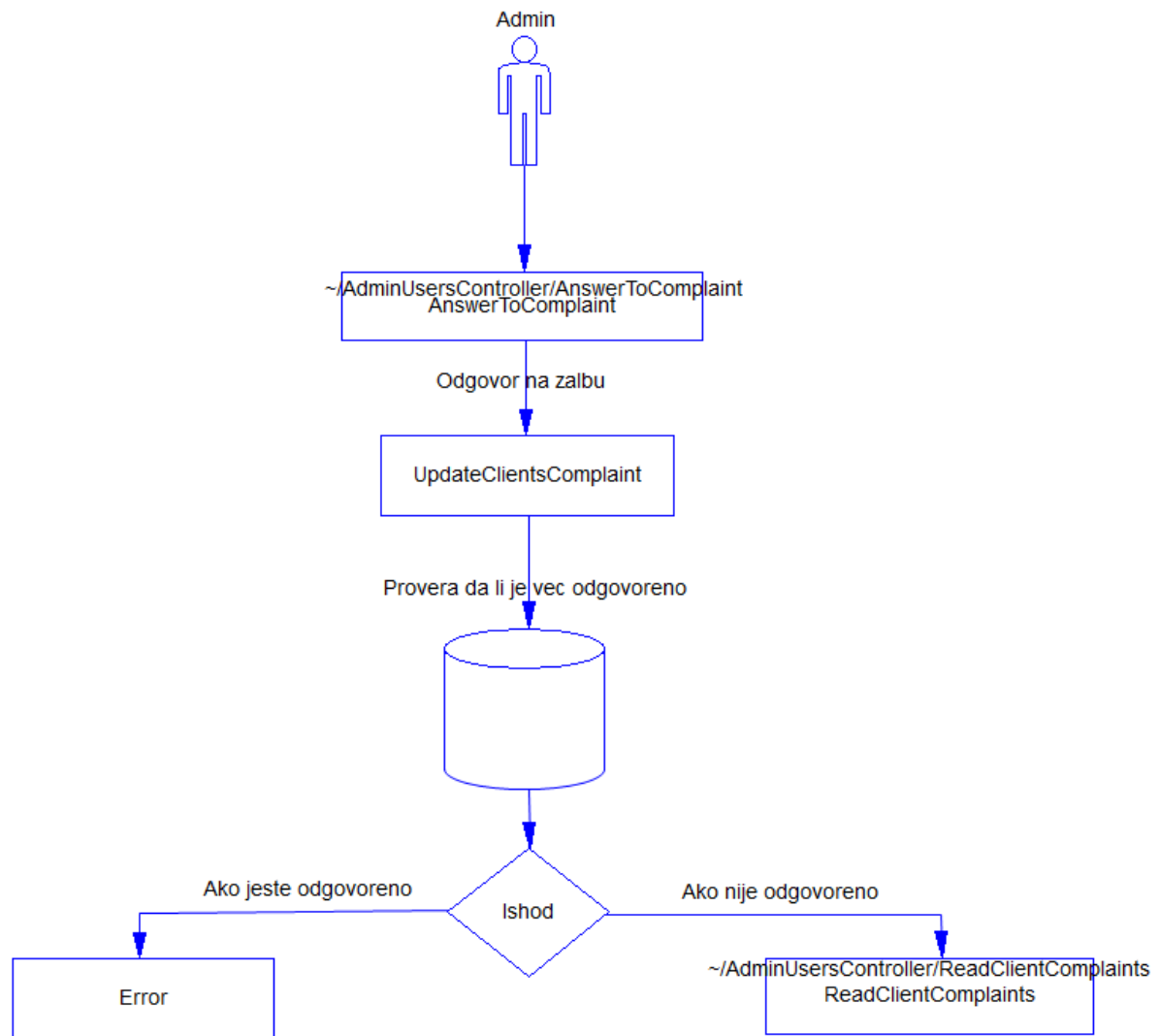
```
public class ClientComplaint
{
    public int Id { get; set; }
    4 references
    public string OwnerId { get; set; }
    4 references
    public string OwnerName { get; set; }
    4 references
    public string OwnerSurname { get; set; }
    4 references
    public string OwnerEmailAddress { get; set; }
    5 references
    public string ClientsEmailAddress { get; set; }
    5 references
    public string ActionTitle { get; set; }
    4 references
    public string Reason { get; set; }
    4 references
    public Enums.ClientComplaintStatus Status { get; set; }
    5 references
    3 references
    public byte[] ConcurrencyToken { get; set; }
}
```

Na slici ispod je prikazana metoda UpdateClientsComplaint koja ažurira torku u bazi i ako ne uspe da nađe torku sa istom vrednošću ConcurrencyToken polja, objaviće grešku. Funkcija se poziva u akciji AnswerToComplaint, kontrolera AdminUsersController.

```
public static int UpdateClientsComplaint(ClientComplaint complaint)
{
    string Sql = @"UPDATE dbo.ClientComplaints
        SET OwnerName = @OwnerName,
            OwnerSurname = @OwnerSurname,
            OwnerEmailAddress = @OwnerEmailAddress,
            ClientsEmailAddress = @ClientsEmailAddress,
            ActionTitle = @ActionTitle,
            Reason = @Reason,
            Status = @Status
        WHERE Id = @Id AND ConcurrencyToken = @ConcurrencyToken";


    var rowCount = -1;
    using (IDbConnection cnn = new SqlConnection(SSMSDataAccess.GettConnectionString()))
    {
        rowCount = cnn.Execute(Sql, complaint);
    }

    return rowCount;
}
```



## Problem prihvatanja ocene korisnika sistema

Prilikom prihvatanja ocene korisnika sistema dozvoljeno je da samo jedan administrator može potvrditi ocenu, pošto se konflikt može desiti ukoliko više admina istovremeno pokušaju da potvrde istu ocenu. Za rešavanje ovog problema je implementiran optimistički pristup. U bazi podataka je dodato novo polje pod nazivom ConcurrencyToken i ono je tipa timestamp. Timestamp tip podatka je ugrađen u MSSQL i služi za ovakve situacije. Vrednost polja se menja automatski ukoliko dođe do bilo kakve promene u torci. Kada čitamo sve zahteve za deaktivaciju, prosleđuje se trenutna vrednost ovog polja, a prilikom narednom pristupu torci, poredićemo da li je prethodno učitana vrednost ista kao i trenutna. Ako je ista, znači nije došlo do ažuriranja torke, u suprotnom će okinuti grešku o tome kako je neko već ažurirao torku.

	Id	int
	ClientsEmailAddress	nvarchar(100)
	EntityTitle	nvarchar(100)
	OwnersEmailAddress	nvarchar(100)
	Description	nvarchar(MAX)
	ActionRating	int
	OwnerInstructorRating	int
	Status	int
	ConcurrencyToken	timestamp

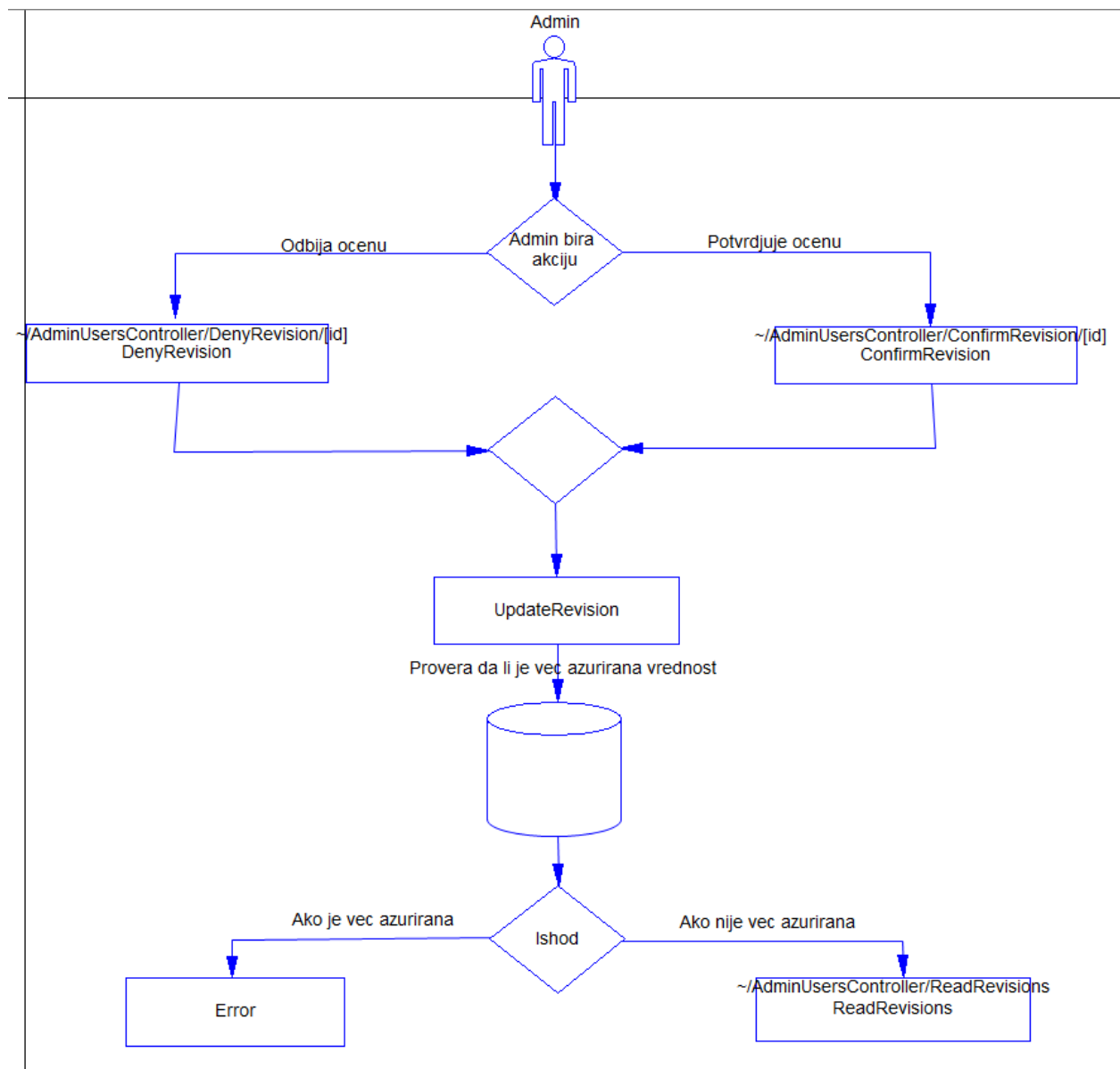
```
public class Revision
{
    10 references
    public int Id { get; set; }
    6 references
    public string ClientsEmailAddress { get; set; }
    9 references
    public string EntityTitle { get; set; }
    9 references
    public string OwnersEmailAddress { get; set; }
    6 references
    public string Description { get; set; }
    9 references
    public float ActionRating { get; set; }
    7 references
    public float OwnerInstructorRating { get; set; }
    6 references
    public Enums.RevisionStatus Status { get; set; }
    5 references
    public byte[] ConcurrencyToken { get; set; }
}
```

Na slici ispod je prikazana metoda UpdateRevision koja ažurira torku u bazi i ako ne uspe da nađe torku sa istom vrednošću ConcurrencyToken polja, objaviće grešku. Funkcija se poziva u akciji ConfirmRevision i DenyRevision, kontrolera AdminUsersController.

```
public static int UpdateRevision(Revision revision)
{
    string Sql = @"UPDATE dbo.Revisions
    SET ClientsEmailAddress = @ClientsEmailAddress,
        EntityTitle = @EntityTitle,
        OwnersEmailAddress = @OwnersEmailAddress,
        Description = @Description,
        ActionRating = @ActionRating,
        OwnerInstructorRating = @OwnerInstructorRating,
        Status = @Status
    WHERE Id = @Id AND ConcurrencyToken = @ConcurrencyToken";

    var rowCount = -1;
    using (IDbConnection cnn = new SqlConnection(SSMSDataAccess.GetConnectionString()))
    {
        rowCount = cnn.Execute(Sql, revision);
    }

    return rowCount;
}
```



## Problem istovremenog pravljenja rezervacije od strane instruktora i od strane klijenta

Obe uloge imaju mogućnost pravljenja rezervacija. Klijent regularno može zakazivati, dok instruktor nema uopšte mogućnost da uđe u opciju za pravljenje rezervacije ako trenutno nije aktivna rezervacija sa nekim od njegovih klijenata. Konkurentni pristup bazi se može desiti ukoliko je klijent odlučio da zakaže rezervaciju, a instruktoru je trenutno aktivna neka rezervacija pa i on može kreirati rezervaciju. Problem je rešen pesimističkim pristupom. Korišćen je nivo izolacije `SERIALIZABLE`. Deo koda koji obezbeđuje transakciju koja će upisati rezervaciju u tabelu je prikazan na slici ispod. Prikazana je metoda `CreateAdventureReservationsSerializable`. Funkcija se poziva u akciji `CreateAdventureReservationForCurrentlyActiveUseri`, kontrolera `InstructorUsersController`.

```
        ClientsEmailAddress = clientsemailAddress,
        ReservationType = type,
        AdventureId = adventureId,
        InstructorId = instructorId
    };

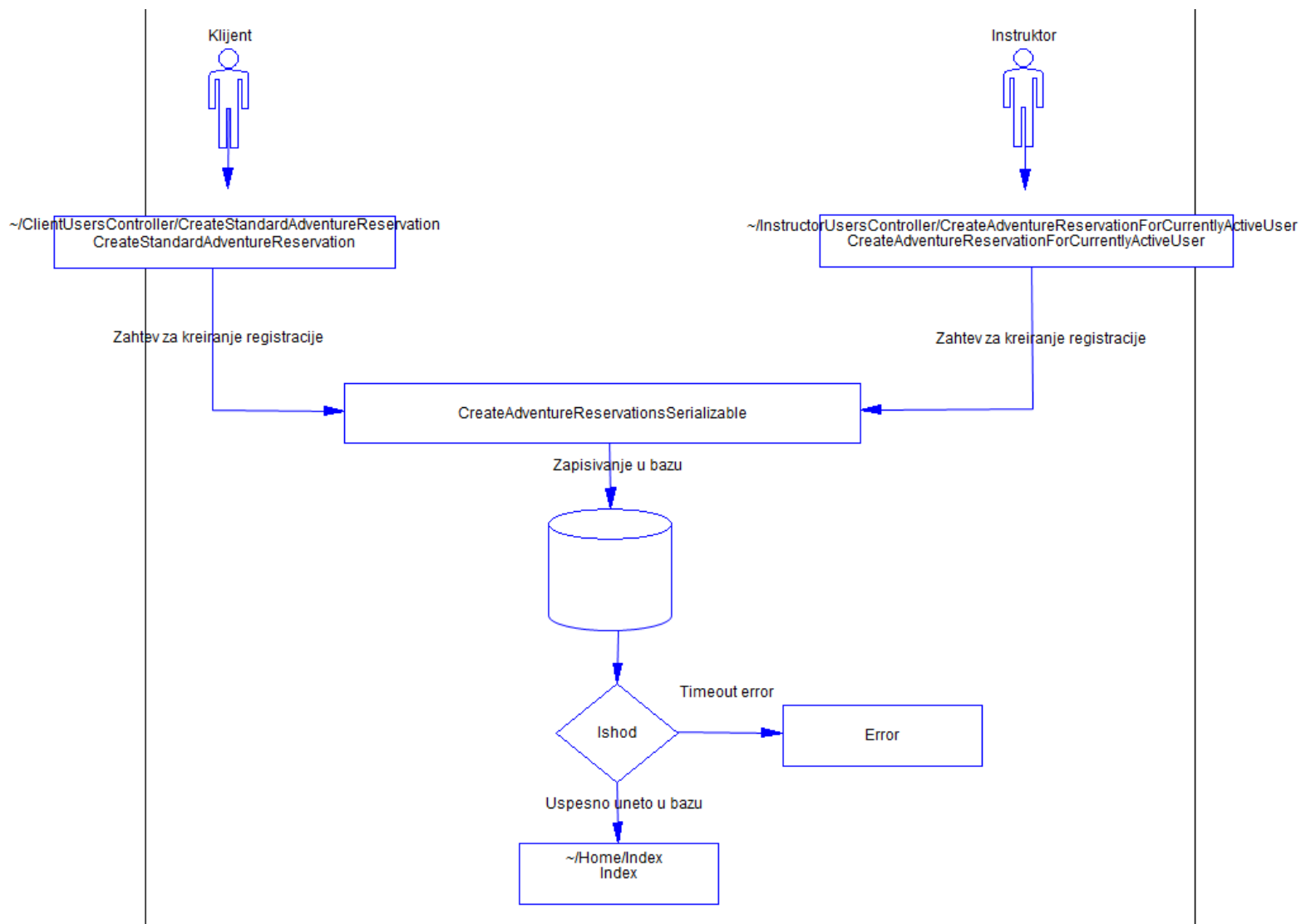
    string sql = @" SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
    BEGIN TRANSACTION
    SELECT * FROM dbo.AdventureReservations
    WAITFOR DELAY '00:00:10'
    INSERT INTO dbo.AdventureReservations WITH(TABLELOCKX)
    (Place, CreationTime, StartDate, StartTime, EndDate, EndTime, ValidityPeriodDate,
    ValidityPeriodTime, dayOfWeek, Month, Year, MaxNumberOfPeople, AdditionalServices,
    Price, Discount, IsReserved, ClientsEmailAddress, ReservationType, AdventureId, InstructorId)

    VALUES (@Place, @CreationTime, @StartDate, @StartTime, @EndDate, @EndTime, @ValidityPeriodDate,
    @ValidityPeriodTime, @dayOfWeek, @Month, @Year, @MaxNumberOfPeople, @AdditionalServices,
    @Price, @Discount, @IsReserved, @ClientsEmailAddress, @ReservationType, @AdventureId, @InstructorId);
    COMMIT TRANSACTION";

    return SSMSDataAccess.SaveData(sql, data);
}
catch (Exception)
{
    throw;
}
```

Naredba `WITH(TABLELOCKX)` obezbeđuje `EXCLUSIVE` zaključavanje nad tabelom. Trajanje transakcije je stavljeno da bude duže nego obično i onaj korisnik (instruktor ili klijent) koji je prvi stigao do ovog dela koda će moći da pokrene transakciju. Drugi korisnik će morati da čeka da se ova transakcija završi, jer ne može da pristupi istoj tabeli u istom trenutku. Pošto je trajanje transakcije duže od tolerancije za `Timeout` grešku, drugom klijentu će se posle par trenutaka okinuti greška da je zahtev za upisivanje neuspešan.

Ideja za rešenje: [How can I tell if a table is locked in an SQL server? - Quora](#)





## Problem istovremenog pravljenja akcije (instruktor) i rezervacije (klijent)

Instruktor ima mogućnost da pravi akcije (brze rezervacije) za svoje avanture, dok klijent ima mogućnost da rezervise te avanture. Konflikti pristupa može nastati kada instruktor i klijent žele ovo da urade u isto vreme. Problem je rešen pesimističkim pristupom. Korišćen je nivo izolacije `SERIALIZABLE`. Deo koda koji obezbeđuje transakciju koja će upisati rezervaciju u tabelu je prikazan na slici ispod. Prikazana je metoda `CreateAdventureReservationsSerializable`. Funkcija se poziva u akciji `CreateReservation`, kontrolera `InstructorUsersController`.

```
        ClientsEmailAddress = clientsemailAddress,
        ReservationType = type,
        AdventureId = adventureId,
        InstructorId = instructorId
    };

    string sql = @" SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
    BEGIN TRANSACTION
    SELECT * FROM dbo.AdventureReservations
    WAITFOR DELAY '00:00:10'
    INSERT INTO dbo.AdventureReservations WITH(TABLOCKX)
    (Place, CreationTime, StartDate, StartTime, EndDate, EndTime, ValidityPeriodDate,
    ValidityPeriodTime, dayOfWeek, Month, Year, MaxNumberOfPeople, AdditionalServices,
    Price, Discount, IsReserved, ClientsEmailAddress, ReservationType, AdventureId, InstructorId)

    VALUES (@Place, @CreationTime, @StartDate, @StartTime, @EndDate, @EndTime, @ValidityPeriodDate,
    @ValidityPeriodTime, @dayOfWeek, @Month, @Year, @MaxNumberOfPeople, @AdditionalServices,
    @Price, @Discount, @IsReserved, @ClientsEmailAddress, @ReservationType, @AdventureId, @InstructorId);
    COMMIT TRANSACTION";

    return SSMSDataAccess.SaveData(sql, data);
}
catch (Exception)
{
    throw;
}
```

Naredba `WITH(TABLOCKX)` obezbeđuje `EXCLUSIVE` zaključavanje nad tabelom. Trajanje transakcije je stavljeno da bude duže nego obično i onaj korisnik (instruktor ili klijent) koji je prvi stigao do ovog dela koda će moći da pokrene transakciju. Drugi korisnik će morati da čeka da se ova transakcija završi, jer ne može da pristupi istoj tabeli u istom trenutku. Pošto je trajanje transakcije duže od tolerancije za *Timeout* grešku, drugom klijentu će se posle par trenutaka okinuti greška da je zahtev za upisivanje neuspešan.

