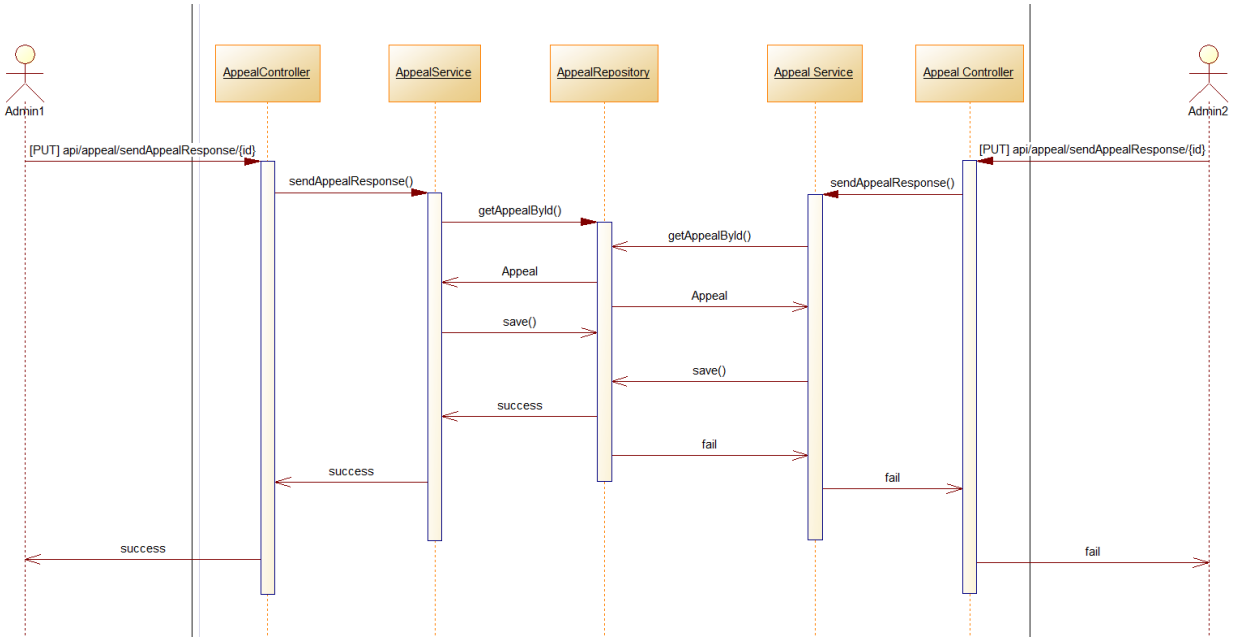
Konkurentni pristup bazi

1. Postupak odgovora na žalbu

   Administrator sistema vidi sve žalbe na koje može da odgovori. Odgovori se unose u slobodnoj formi i šalju se obema stranama zasebno na email.

   Problem:

   Prilikom odgovaranja na žalbe korisnika, dva različita administratora mogu istovremeno da pokušaju da odgovore na istu žalbu korisnika. U sistemu se vodi evidencija da li je odgovoreno na žalbu. Do konflikta dolazi kada oba korisnika pokušaju da odgovore na žalbu tj. da promene njeno stanje. Problem se rešava korišćenjem optimističkog zaključavanja.



   Ishod:

   Admin čija transakcija se prva kreirala će izmeniti stanje žalbe i uvećati vrednost verzije žalbe za jedan. Prilikom pokušaja izmene, druga transakcija će uporediti početnu sa trenutnom verzijom žalbe I doći će do izuzetka. Druga transakcija se neće izvršiti.

```
@Entity
public class Appeal {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String review;
    private boolean isAnswered;

    @Version
    @Column(nullable = false)
    private Integer version;
```

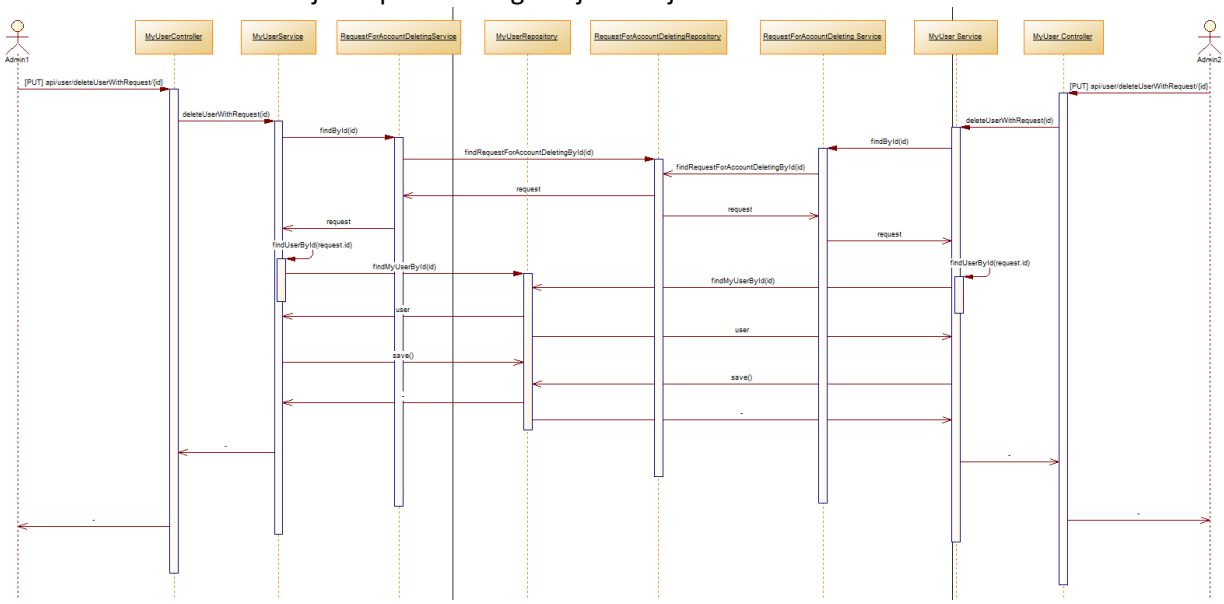
```java
@Transactional
public boolean sendAppealResponse(Long id, ReportAppealAnswerDTO answerDTO) throws MessagingException {
    Appeal appeal = this.getAppealById(id);
    appeal.setAnswered(true);
    this.save(appeal);
    this.emailService.sendAnswerEmail(new EmailDTO("Odgovor na žalbu", answerDTO.getGuestResponse(), appeal.getSenderId().getEmail()));
    this.emailService.sendAnswerEmail(new EmailDTO("Žalba klijenta", answerDTO.getOwnerResponse(), appeal.getOwnerId().getEmail()));
    return true;
}
```

2. Odgovor na zahtev za brisanje naloga

Svi korisnici mogu da pošalju zahtev za brisanje naloga. Tekst zahteva unosi se u slobodnoj formi. Administrator sistema vidi sve zahteve koje može da odobri ili odbije. Odgovori se unose u slobodnoj formi i šalju se na email.

Problem:

Prilikom odgovaranja na zahteve za brisanje naloga korisnika, dva različita administratora mogu istovremeno da pokušaju da odgovore na isti zahtev korisnika. U sistemu se vodi evidencija da li je odgovoreno na zahtev korisnika. Do konflikta dolazi kada oba korisnika pokušaju da odgovore na zahtev korisnika tj. da promene njegovo stanje. Na primer, jedan admin može odlučiti da prihvati zahtev i obriše korisnika, a drugi može odlučiti da odbije zahtev za brisanje naloga. Problem se rešava korišćenjem optimističkog zaključavanja.



Ishod:

Admin čija transakcija se prva kreirala će izmeniti stanje zahteva za brisanje naloga i uvećati vrednost verzije zahteva za brisanje za jedan. Prilikom pokušaja izmene, druga transakcija će uporediti početnu sa trenutnom verzijom zahtevaa za brisanje I doći će do izuzetka. Druga transakcija se neće izvršiti.

```java
@Entity
public class RequestForAccountDeleting {

    @Id
    @GeneratedValue
    private Long id;

    @Version
    @Column(nullable = false)
    private Integer version;
```

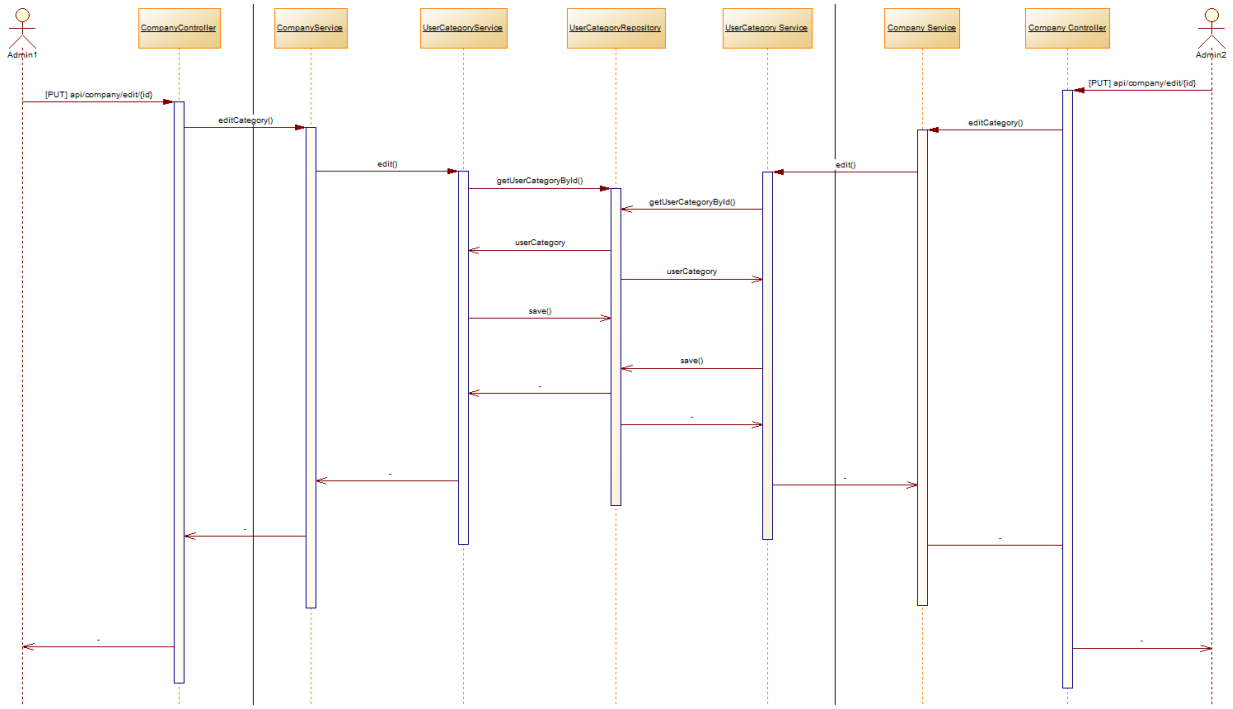
```java
@Transactional
public boolean deleteUserWithRequest(Long id, String clientMessage) throws MessagingException {
    RequestForAccountDeleting request = this.requestForAccountDeletingService.findById(id);
    MyUser user = this.findUserById(request.getUser().getId());
    user.setDeleted(true);
    this.save(user);
    request.setAnswered(true);
    this.requestForAccountDeletingService.save(request);
    this.emailService.sendAnswerEmail(new EmailDTO("Odgovor na zahtev za brisanje naloga", clientMessage, user.getEmail()));
    return true;
}

@Transactional
public boolean declineDeleteRequest(Long id, String clientMessage) throws MessagingException {
    RequestForAccountDeleting request = this.requestForAccountDeletingService.findById(id);
    request.setAnswered(true);
    this.requestForAccountDeletingService.save(request);
    this.emailService.sendAnswerEmail(new EmailDTO("Odgovor na zahtev za brisanje naloga", clientMessage, request.getUser().getEmail()));
    return true;
}
```

3. Izmena kategorije loyalty programa

Administrator sistema može da definiše loyalty program za sve klijente koji važi na nivou čitavog sistema. Administrator definiše broj poena koje ostvaruje klijent nakon svake rezervacije. Takođe, definiše broj poena koje ostvaruje svaki vlasnik/instruktor pecanja pri svakoj uspešnoj rezervaciji. Sem broja poena, administrator sistema definiše skale na osnovu koje se određuje kategorija klijenata i pružaoca usluga (npr. Regular, Silver, Gold). Na osnovu kategorije, klijent ostvaruje dodatni popust koji se primenjuje na svaku rezervaciju. Na osnovu kategorije, vlasnik/instruktor ostvaruje veći prihod koji se primenjuje na svaku rezervaciju.

Problem:

Prilikom izmene kategorije loyalty programa, dva različita administratora mogu istovremeno da pokušaju da izmene istu kategoriju. U sistemu se vodi evidencija o verziji kategorije. Do konflikta dolazi kada oba korisnika pokušaju da izmene istu kategoriju tj. da promene njeno stanje. Problem se rešava korišćenjem optimističkog zaključavanja.



Ishod:

Admin čija transakcija se prva kreirala će izmeniti kategoriju loyalty programa i uvećati vrednost njene verzije za jedan. Prilikom pokušaja izmene, druga transakcija će uporediti početnu sa trenutnom verzijom kategorije i doći će do izuzetka. Druga transakcija se neće izvršiti.

```
@Entity
public class UserCategory {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Version
    @Column(nullable = false)
    private Integer version;
```

```
  @Transactional
  public boolean edit(Long id, UserCategoryDTO dto) {
      UserCategory category = this.userCategoryRepository.getUserCategoryById(id);
      category.setName(dto.getName());
      category.setDiscountPercentage(dto.getDiscountPercentage());
      category.setPoints(dto.getPoints());
      this.userCategoryRepository.save(category);
      return true;
  }
```