Rešavanje konfliktnih situacija

Zorana Stamenković IN10-2017

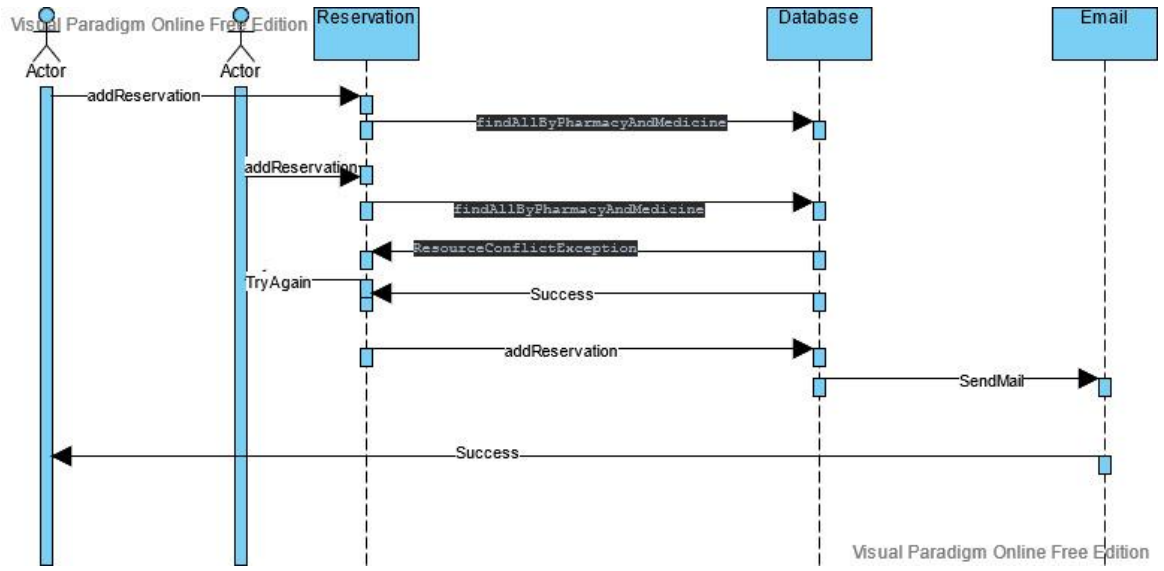
1.Rezervacija leka

Opis funkcionalnosti:

Pacijent otvara stranicu za rezervaciju leka, prikazuju mu se svi dostupni lekovi, nakon odabira leka, prikazuju mu se apoteke u kojoj je taj lek dostupan, nakon odabira apoteke, pacijent mora da unese datum i vreme do kad mora da podigne lek.

Opis konfliktne situacije:

Lek koji je korisnik izabrao, može u međuvremenu biti rezervisan od strane drugog pacijenta. Kao posledicu imali bi više rezervacija leka, nego što imamo leka u odabranoj apoteci.



Rešenje konfliktne situacije:

Korišćen je optimistički pristup, tj provera se stanje leka u odabranoj apoteci, još jednom pre upisa. Ukoliko se stanje leka smanjilo, tj svelo na nulu doći će do izuzetka “ResourceConflictException”, i vratiti korisniku da nije uspeo da rezerviše lek. Ukoliko postoji još lekova u odabranoj apoteci, omogućene su njihove rezervacije.

```java
@Override
public void addReservation(ReservationDTO reservationDto){
    Reservation reservation = reservationMapper.bean2Entity(reservationDto);
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    Optional<Patient> patientOptional = patientRepository.findById(((User) authentication.getPrincipal()).getId());
    if(patientOptional.isPresent()) {
        Optional<Pharmacy> pharmacy= pharmacyRepository.findById((long) reservationDto.getPharmacyid());
        Optional<Medicine> medicine= medicineRepository.findById((long) reservationDto.getMedicineid());
        MedicineQuantityPharmacy mqp = medicineQuantityPharmacyRepository.findAllByPharmacyAndMedicine(pharmacy.get(),medicine.get());
        if (mqp.getQuantity() == 0){
            System.out.println("Promenjen!");
            throw new ResourceConflictException(1L,"Promenjen u medjuvremenu!");

        }
        reservation.setPharmacy(pharmacy.get());
        reservation.setMedicine(medicine.get());
        reservation.setEndDate(reservationDto.getEndDate());
        reservation.setPickUpTime(reservationDto.getPickedUpTime());
        reservation.setEndTime(reservationDto.getEndTime());
        reservation.setPatient(patientOptional.get());
        Patient patient = patientOptional.get();

        reservationRepository.saveAndFlush(reservation);

        JavaMailSenderImpl mailSender = getJMS();
        SimpleMailMessage mailMessage = new SimpleMailMessage();
        mailMessage.setFrom("apoteka@gmail.com");
        mailMessage.setTo(patient.getEmail());
        mailMessage.setSubject("Rezervisanje leka");
```

```java
    mailMessage.setText("Postovani " + patient.getFirstName() + ",\n"+ "\n"+
            "Uspesno ste rezervisali lek " + reservation.getMedicine().getName() +
            " broj rezervacije" + reservation.getId() +  "\n"+ "\n"+
            "Pozdrav," + "\n"+
            "AP tim");
    mailSender.send(mailMessage);


    mqp.setQuantity(mqp.getQuantity() - 1);

    medicineQuantityPharmacyRepository.saveAndFlush(mqp);
```

2.Rezervacija unapred zadatih termina

Opis funkcionalnosti:

Pacijent otvara stranicu za rezervisanje termina kod dermatologa, pritiskom na dugme zakazuje termin.

Opis konfliktne situacije:

Prilikom rezervacije moze da se desi da dva ili više korisnika su poslali zahtev za rezervaciju istog termina. Treba sprečiti da dva korisnika rezervišu isti termin.

Rešenje konfliktne situacije:

Korišćen je optimistički pristup.Provera se da li je termin rezervisan u međuvremenu. Ukoliko jeste doći će do izuzetka “ResourceConflictException”, i vratiti korisniku da nije uspeo da rezerviše termin. Ukoliko nije, korisnik će rezevisati termin i stiće mu poruka na mail o uspešnoj rezervaciji.

```java
@Override
public void addExamination(ExaminationDto examinationDto) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    Optional<Patient> patientOptional = patientRepository.findById(((User) authentication.getPrincipal()).getId());
    if(patientOptional.isPresent()) {
        Examination examination = examinationMapper.bean2Entity(examinationDto);
        Optional<Examination> examination1 = examinationRepository.findById(examinationDto.getId());
        examination = examination1.get();
        System.out.println(examination.getFree());
        if (!examination.getFree()){
            System.out.println("Promenjen!");
            throw new ResourceConflictException(1L,"Promenjen u medjuvremenu!");


        }

        Patient patient = patientOptional.get();
        examination.setPatient(patient);
        examination.setFree(false);
        examination.setPenalty(true);

        JavaMailSenderImpl mailSender = getJMS();
        SimpleMailMessage mailMessage = new SimpleMailMessage();
        mailMessage.setFrom("apoteka@gmail.com");
        mailMessage.setTo(patient.getEmail());
        mailMessage.setSubject("Zakazivanje termina");
        mailMessage.setText("Postovani " + patient.getFirstName() + ",\n"+ "\n"+
                "Uspesno ste zakazali termin za " + examination.getDate() +
                " kod lekara" + examination.getDermatologist().getFirstName() + "  " + examination.getDermatologist().getLastName() +  "\n"+ "\n"+
```

.

```java
                "Pozdrav," + "\n"+
                "AP tim");
        mailSender.send(mailMessage);

        examinationRepository.saveAndFlush(examination);
    }
```

3.Ocenjivanje lekova, apoteka, dermatologa i farmaceuta

Opis funkcionalnosti:

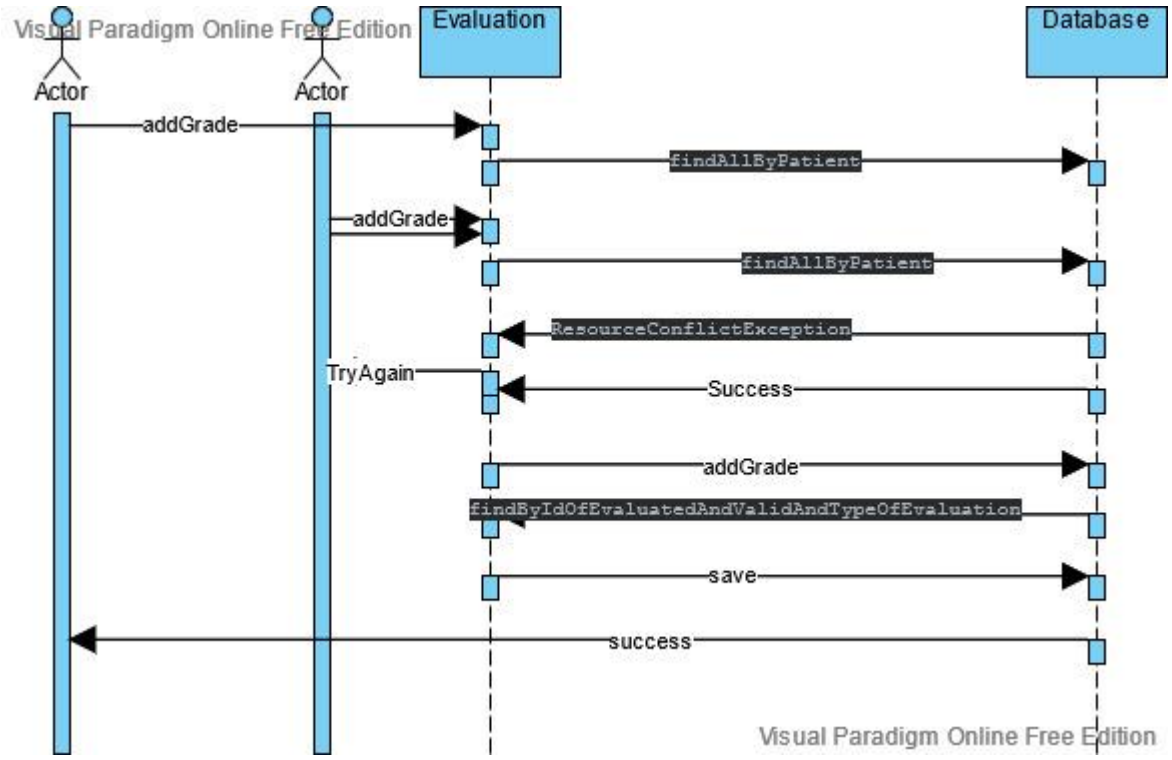
Pacijent otvara stranicu za ocenjivanje leka, apoteke, dermatologa ili farmaceuta. Korisnik odabira koga želi da oceni, unosi i potvrđuje ocenu.

Opis konfliktne situacije:

Konfliktna situacija nastaje ako pacijent otvori stranicu za ocenjivanje na dva prozora i na taj način oceni, u bazi će se zapamtiti njegova druga ocena, ali to nije relevatno stanje baze.

Rešenje konfliktne situacije:

Korišćen je optimistički pristup.Provera se da li ga je korisnik predhodno ocenio. Ukoliko jeste doći će do izuzetka “ResourceConflictException”, i vratiti korisniku da nije uspeo da ga ocenu. Ukoliko nije, zapamtiće se ocena korisnika i izračunati nova prosečna ocena.




```java
@Override
public void addGrade(EvaluationDTO evaluationDTO) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    Optional<Patient> patientOptional = patientRepository.findById(((User) authentication.getPrincipal()).getId());
    if(patientOptional.isPresent()) {
        if (evaluationRepository.findAllByPatient(patientOptional.get()).size()!=0){
            throw new ResourceConflictException(1L,"Ocenio si ga vec!");
        }
        Evaluation evaluation = evaluationMapper.bean2Entity(evaluationDTO);
        evaluation.setValid(true);
        evaluation.setPatient(patientOptional.get());
        evaluation.setGrade(evaluationDTO.getGrade());
        Patient patient = patientOptional.get();

        Optional<Dermatologist> dermatologist = dermatologistRepository.findById(evaluationDTO.getIdOfEvaluated());
        System.out.println(dermatologist.isPresent());
        Optional<Pharmacist> pharmacist = pharmacistRepository.findById(evaluationDTO.getIdOfEvaluated());
        System.out.println(pharmacist.isPresent());
        Optional<Medicine> medicine = medicineRepository.findById(evaluationDTO.getIdOfEvaluated());
        System.out.println(pharmacist.isPresent());
        Optional<Pharmacy> pharmacy = pharmacyRepository.findById(evaluationDTO.getIdOfEvaluated());
        System.out.println(pharmacy.isPresent());
        if(evaluation.getTypeOfEvaluation().equals("farmaceut")){
            Optional<Pharmacist> pharmacistChoosen = pharmacistRepository.findById(evaluationDTO.getIdOfEvaluated());
            evaluation.setName(pharmacistChoosen.get().getFirstName() + ' ' + pharmacistChoosen.get().getLastName());
        }
        if(evaluation.getTypeOfEvaluation().equals("dermatolog")){
            Optional<Dermatologist> dermatologistChoosen = dermatologistRepository.findById(evaluationDTO.getIdOfEvaluated());
            evaluation.setName(dermatologistChoosen.get().getFirstName() + ' ' + dermatologistChoosen.get().getLastName());
        }
```

```
        Optional<Medicine> medicineChoosen = medicineRepository.findById(evaluationDTO.getIdOfEvaluated());
        evaluation.setName(medicineChoosen.get().getName());
    }
    if(evaluation.getTypeOfEvaluation().equals("apoteka")){
        Optional<Pharmacy> pharmacyChoosen = pharmacyRepository.findById(evaluationDTO.getIdOfEvaluated());
        evaluation.setName(pharmacyChoosen.get().getName());
    }
    evaluationRepository.saveAndFlush(evaluation);
    //List<Evaluation> evaluations = evaluationRepository.findByIdOfEvaluatedAndValid(evaluationDTO.getIdOfEvaluated(), true);
    if ((pharmacist.isPresent()) & evaluation.getTypeOfEvaluation().equals("farmaceut")) {
        Set<Evaluation> evaluations = evaluationRepository.findByIdOfEvaluatedAndValidAndTypeOfEvaluation(evaluationDTO.getIdOfEvaluated(), b: true, tip: "farmaceut");
        float new_grade = 0;
        for (Evaluation eva : evaluations) {
            new_grade = new_grade + eva.getGrade();
        }
        new_grade = new_grade / evaluations.size();
        Pharmacist pharmacist_help = new Pharmacist();
        pharmacist_help  = pharmacist.get();
        pharmacist_help.setEvaluationGrade(new_grade);
        pharmacistRepository.saveAndFlush(pharmacist_help);
    }
    if (dermatologist.isPresent() & evaluation.getTypeOfEvaluation().equals("dermatolog")) {
        Set<Evaluation> evaluations = evaluationRepository.findByIdOfEvaluatedAndValidAndTypeOfEvaluation(evaluationDTO.getIdOfEvaluated(), b: true, tip: "dermatolog");
        float new_grade = 0;
        for (Evaluation eva : evaluations) {
            new_grade = new_grade + eva.getGrade();
        }
        new_grade = new_grade / evaluations.size();
        dermatologist.get().setEvaluationGrade(new_grade);
        Dermatologist dermatologist_help = new Dermatologist();
        dermatologist_help  = dermatologist.get();
        dermatologist_help.setEvaluationGrade(new_grade);
        dermatologistRepository.saveAndFlush(dermatologist_help);
```

```
if (medicine.isPresent() & evaluation.getTypeOfEvaluation().equals("lek")){
    Set<Evaluation> evaluations = evaluationRepository.findByIdOfEvaluatedAndValidAndTypeOfEvaluation(evaluationDTO.getIdOfEvaluated(), b: true, tip: "lek");
    float new_grade = 0;
    for (Evaluation eva : evaluations) {
        new_grade = new_grade + eva.getGrade();
    }
    new_grade = new_grade / evaluations.size();
    System.out.println(evaluations.size());
    medicine.get().setEvaluationGrade(new_grade);
    Medicine medicine_help = new Medicine();
    medicine_help  = medicine.get();
    medicine_help.setEvaluationGrade(new_grade);
    medicineRepository.saveAndFlush(medicine_help);
}
if (pharmacy.isPresent() & evaluation.getTypeOfEvaluation().equals("apoteka")){
    Set<Evaluation> evaluations = evaluationRepository.findByIdOfEvaluatedAndValidAndTypeOfEvaluation(evaluationDTO.getIdOfEvaluated(), b: true, tip: "apoteka");
    float new_grade = 0;
    for (Evaluation eva : evaluations) {
        new_grade = new_grade + eva.getGrade();
    }
    new_grade = new_grade / evaluations.size();
    pharmacy.get().setEvaluationGrade(new_grade);
    Pharmacy pharmacy_help = new Pharmacy();
    pharmacy_help  = pharmacy.get();
    pharmacy_help.setEvaluationGrade(new_grade);
    pharmacyRepository.saveAndFlush(pharmacy_help);
}
```