Konkurentni pristup bazi (Vladimir Jovanović RA128-2018)

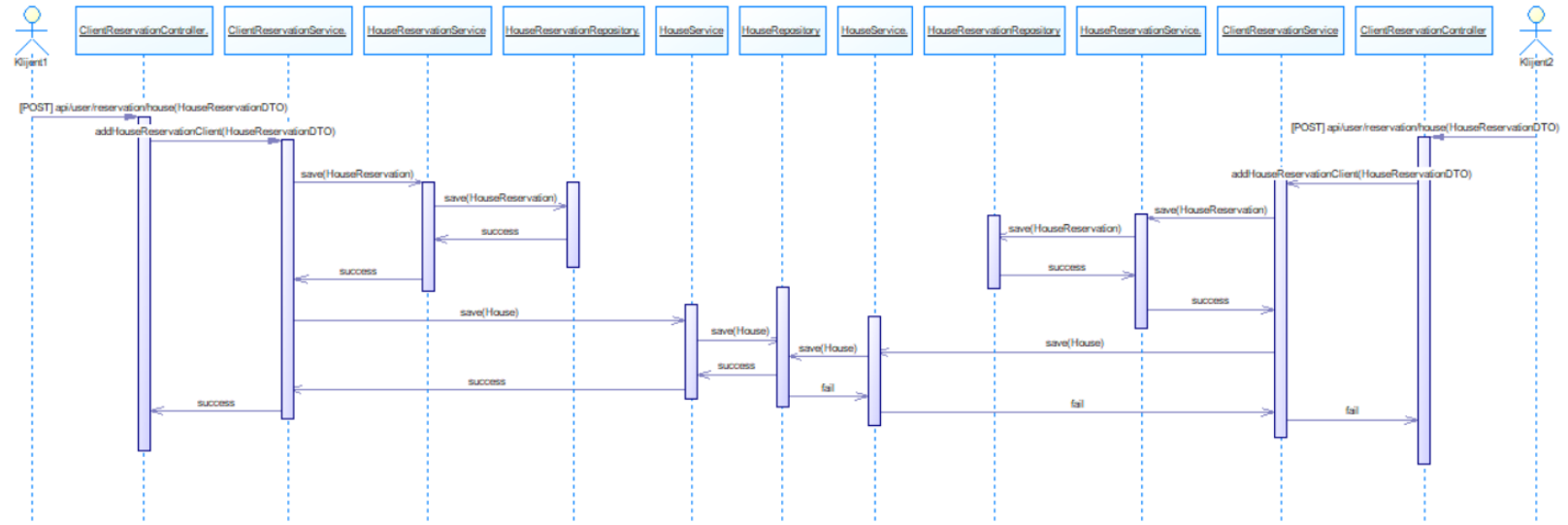
1. Postupak kreiranja rezervacije

Opis konfliktne situacije*:*

Klijenti imaju mogućnost kreiranja rezervacija. Može doći do konfliktne situacije ukoliko više klijenata pokuša da kreira rezervaciju u istom ili preklapajućem terminu. Problem se može rešiti optimističkim zaključavanjem.

Crtež toka zahteva:

Crtež toka zahteva je prikazan za primer situacije kada dva klijent pokušaju da kreiraju rezervaciju vikendice u istom ili preklapajućem terminu.



Opis načina na koji je problem rešen:

Konfliktna situacija je resena pomoću optimističkog zaključavanja. Ovaj način rada uvodi novo polje - version (verziju) u klasu House. Polje version se koristi za proveru stanja entiteta.
Klijent koji je prvi započeo transakciju će uspešno izvršiti kreiranje rezervacije. Na ilustrovanom primeru, klijent 1 će uspešno kreirati rezervaciju. Prilikom pokušaja kreiranje rezervacije u istom terminu od strane klijenta 2, vrši se poređenje početne vrednosti polja verzije sa trenutnom vrednosti polja verzije a pošto se te vrednosti razlikuju, dolazi do izuzetka. Druga transakcija se neće izvršiti.

ClientReservationService (addHouseReservationClient) – transactional

```
@Transactional
public boolean addHouseReservationClient(HouseReservationDTO dto)  {
    ReservationCheckDTO reservationCheckDTO = getReservationCheckDTO(dto);
    House house = this.houseService.getHouseById(dto.getHouseId());
    MyUser guest = this.myUserService.findUserById(dto.getGuestId());
    boolean isAvailable = this.houseReservationService.findHouseAvailability(reservationCheckDTO, dto.getHouseId());
    if(isAvailable){
        Date startDate = new Date(dto.getMilisStartDate());
        Date endDate = new Date(dto.getMilisEndDate());
        HouseReservation houseReservation = new HouseReservation(dto.getId(), startDate, endDate, dto.getMaxGuests(),
                dto.getPrice(), dto.isAvailable(), house);
        houseReservation.setAvailabilityPeriod(dto.isAvailabilityPeriod());
        houseReservation.setAction(dto.isAction());
        houseReservation.setHouse(house);
        houseReservation.setGuest(guest);
        guest.setPoints(guest.getPoints() + this.companyService.getCompanyInfo((long) 1).getPointsPerReservationClient());
        this.checkUserCategory(guest);
        houseReservation = this.houseReservationService.save(houseReservation);
        addAdditionalServices(dto, houseReservation);
        sendEmailClient(guest, house.getName(),  boatName: "",  adventureName: "");
    }

    return isAvailable;
}
```

```
@Entity
public class House {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

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

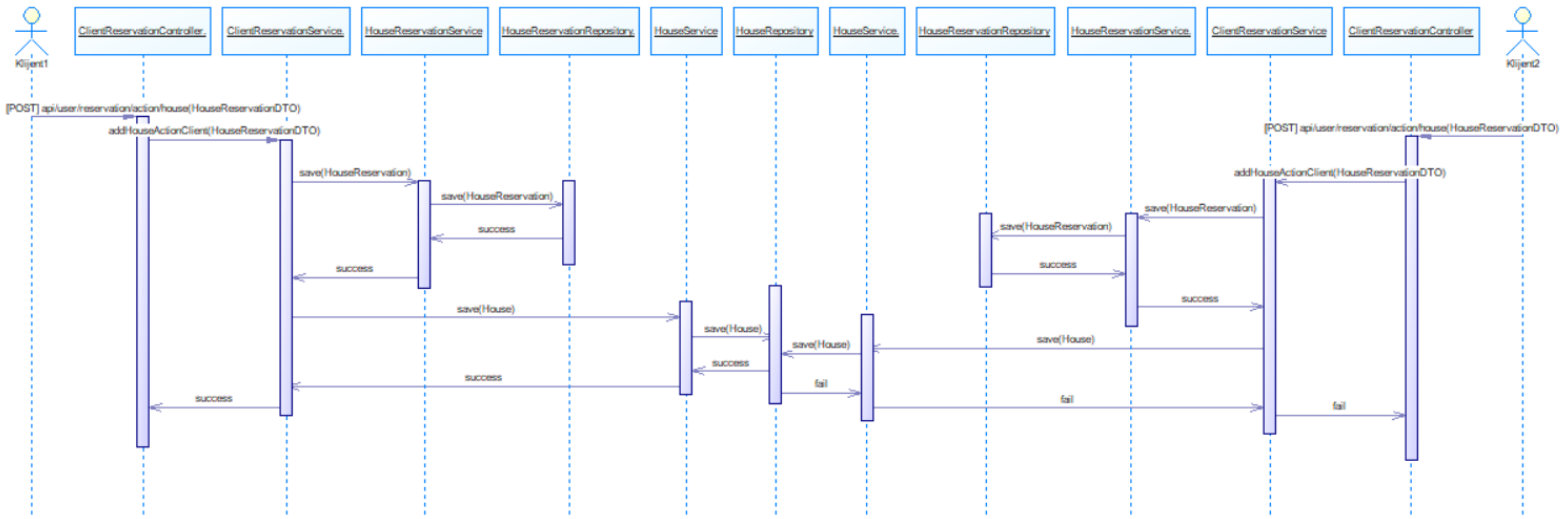
2. Postupak rezervisanja akcije

Opis konfliktne situacije*:*

Klijenti imaju mogućnost rezervisanja akcija. Do konfliktne situacije može doći ukoliko više klijenata pokuša da zakaže istu akciju. Problem se može rešiti optimističkim zaključavanjem.

Crtež toka zahteva:

Crtež toka zahteva je prikazan za primer kada dva klijent pokušaju da rezervišu istu akciju vikendice.



ClientReservationService (addHouseActionClient) – transactional

```
@Transactional
public Boolean addHouseActionClient(ActionDTO dto) {
    MyUser guest = this.myUserService.findUserById(dto.getUserId());
    HouseReservation houseReservation = this.houseReservationService.getHouseReservationById(dto.getEntityId());
    if(houseReservation == null || !houseReservation.isAction()){
        return false;
    }
    houseReservation.setGuest(guest);
    houseReservation.setAvailable(false);
    guest.setPoints(guest.getPoints() + this.companyService.getCompanyInfo((long) 1).getPointsPerReservationClient());
    this.checkUserCategory(guest);
    this.houseReservationService.save(houseReservation);
    this.sendEmailClientAction(guest,houseReservation.getHouse().getName(), boatName: "" );
    return true;
}
```

Opis načina na koji je problem rešen:

Konfliktna situacija je resena na osnovu optimističkog zaključavanja. Ovaj način rada uvodi novo polje - version (verziju) u klasu House. Polje version se koristi za proveru stanja entiteta.
Klijent koji je prvi započeo transakciju će uspešno izvršiti rezervaciju akcije. Na ilustrovanom primeru, klijent 1 će uspešno rezervisati akciju. Prilikom pokušaja rezervacije iste akcije od strane klijenta 2, vrši se poređenje početne vrednosti polja verzije sa trenutnom vrednosti polja verzije i pošto se te vrednosti razlikuju, dolazi do izuzetka. Druga transakcija se neće izvršiti.

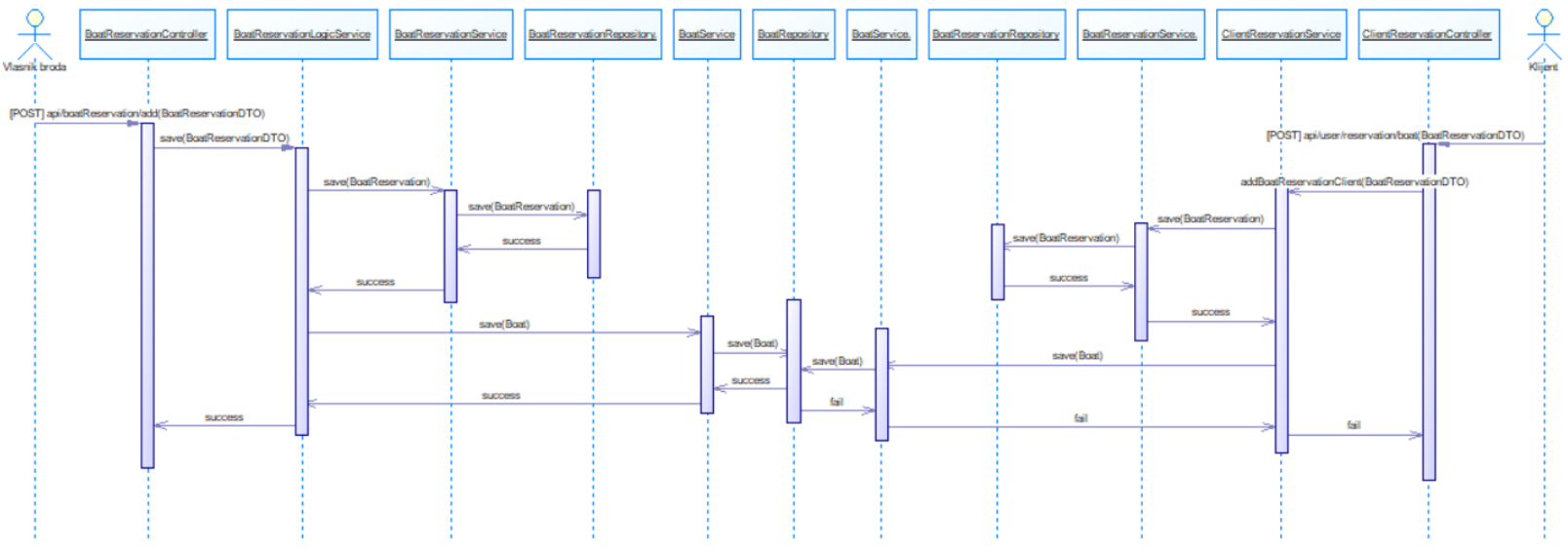
3. Postupak kreiranja rezervacije od strane klijenta dok vlasnik istovremeno definiše period nedostupnosti

Opis konfliktne situacije:

Vlasnici vikendica/brodova, instruktori mogu da definišu period nedostupnosti. Prilikom definisanja perioda nedostupnosti, moze doći do konfliktne situacije ukoliko npr klijent pokuša da napravi rezervaciju u istom terminu. Problem se može rešiti korišćenjem optimističkog zaključavanja.

Crtež toka zahteva:

Crtež toka zahteva je prikazan za primer situacije kada vlasnik broda definiše period nedostupnosti i klijent pokuša da kreira rezervaciju u istom ili preklapajućem terminu.



Opis načina na koji je problem rešen:

Ova konfliktna situacija je resena na osnovu optimističkog zaključavanja. Optimističko zaključavanje uvodi novo polje - version (verziju) u klasu Boat. Ovo polje se koristi za proveru stanja entiteta. Osoba čija transakcija je prva pokrenuta će uspešno izvršiti kreiranje rezervacije (ili definisanje perioda nedostupnosti). Na gore ilustrovanom primeru, vlasnik broda će uspešno definisati period nedostupnosti. Prilikom pokušaja kreiranje rezervacije u istom terminu od strane klijenta, vrši se poređenje početne vrednosti polja verzija sa trenutnom vrednosti polja verzije (broda) i s obzirom da se te vrednosti ne poklapaju, dolazi do izuzetka i zato se druga transakcija neće izvršiti.

BoatReservationLogicService (save) - transactional

```java
@Transactional
public boolean save(BoatReservationDTO dto) throws MessagingException {
    Boat boat = this.boatService.getBoatById(dto.getBoatId());
    MyUser owner = this.myUserService.findUserByBoatId(dto.getBoatId());

    if (isReservationFree(dto, boat)) return false;

    BoatReservation boatReservation = setFields(dto, boat);
    boatReservation =this.boatReservationService.save(boatReservation);
    setOwnerPoints(dto, owner);
    setAdditionalServices(dto, boatReservation);
    boat.addBoatReservation(boatReservation);
    this.boatService.save(boat);

    sendEmailIfClient(dto, boat, boatReservation);
    sendEmailIfAction(dto, boat);

    return true;
}
```

```java
@Entity
public class Boat {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

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

ClientReservationService (addBoatReservationClient) - transactional

```java
@Transactional
public boolean addBoatReservationClient(BoatReservationDTO dto){
    ReservationCheckDTO reservationCheckDTO = getReservationCheckDTO(dto);
    Boat boat = this.boatservice.getBoatById(dto.getBoatId());
    MyUser guest = this.myUserService.findUserById(dto.getGuestId());
    boolean isAvailable = this.boatReservationService.findBoatAvailability(reservationCheckDTO, boat.getId());
    if(isAvailable){
        Date startDate = new Date(dto.getMilisStartDate());
        Date endDate = new Date(dto.getMilisEndDate());
        BoatReservation boatReservation = new BoatReservation(dto.getId(), startDate, endDate, dto.getMaxGuests(),
                dto.getPrice(), dto.isAvailable(), boat);
        boatReservation.setAvailabilityPeriod(dto.isAvailabilityPeriod());
        boatReservation.setAction(dto.isAction());
        boatReservation.setGuest(guest);
        guest.setPoints(guest.getPoints() + this.companyService.getCompanyInfo((long) 1).getPointsPerReservationClient());
        this.checkUserCategory(guest);
        boatReservation = this.boatReservationService.save(boatReservation);
        addAdditionalServices(dto, boatReservation);
        sendEmailClient(guest, houseName: "", boat.getName(), adventureName: "");
    }
    return isAvailable;
}
```