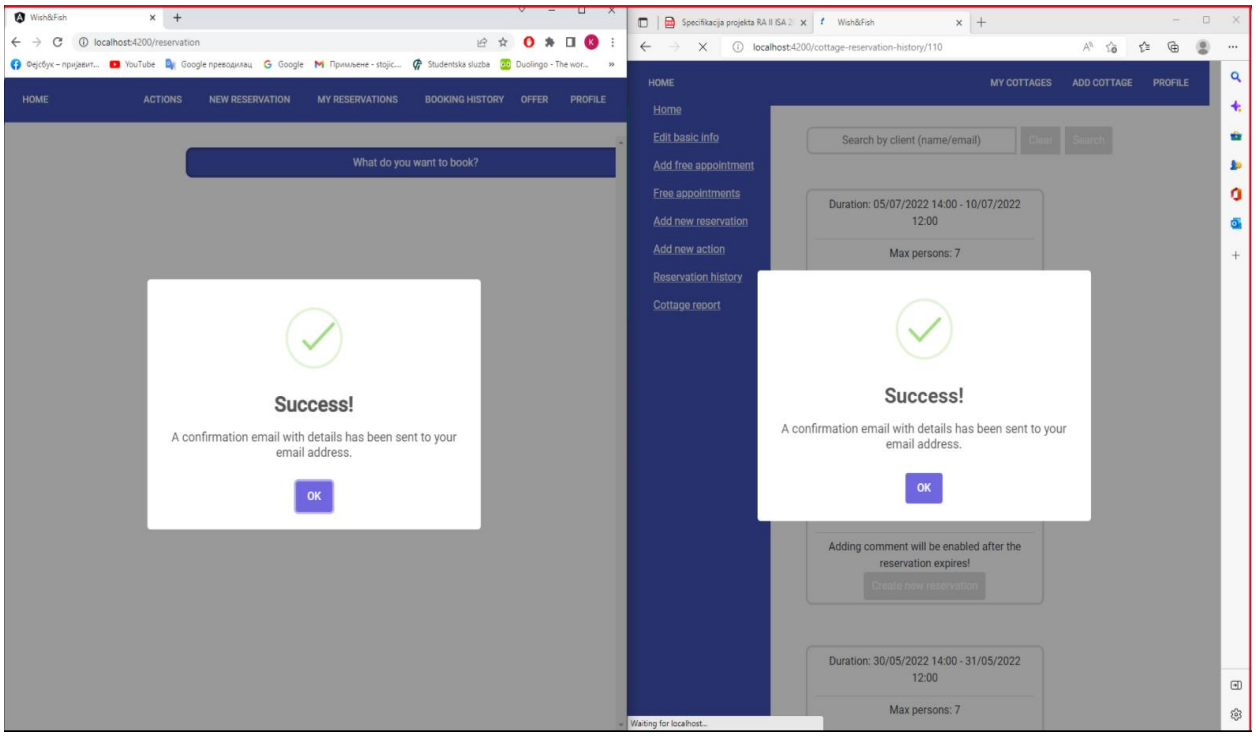
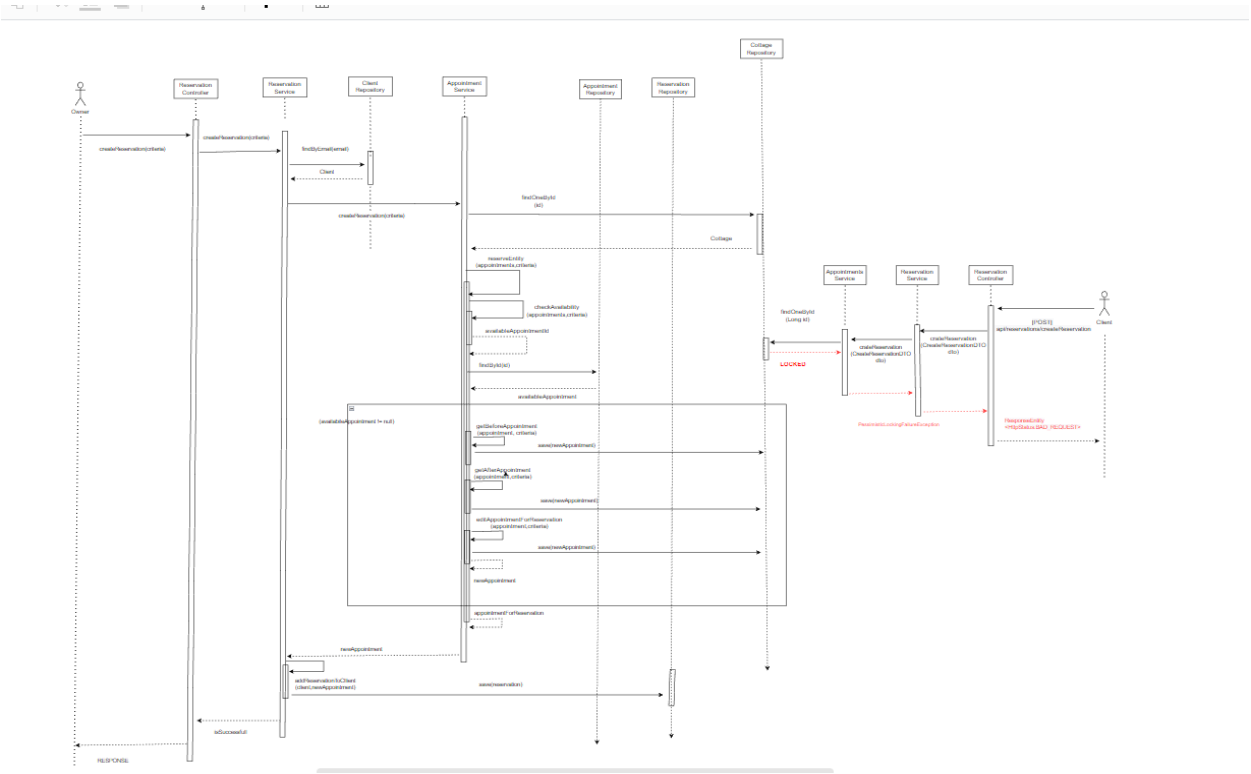
## 4.4 Конкурентни приступ ресурсима у бази – Студент 2

1. Власник викендице/брода или инструктор не може да направи резервацију у исто вријеме кад и други клијент

Опис проблема: Претпоставимо да у исто вријеме  викендицу желе да резервишу клијент за сопствене потребе али и власник исте за неког другог клијента. Обојица бирају термин који се у неком периоду поклапа са термином другог и врше резервацију. Пошто је ентите слободан у оба случаја, оба корисника ће добити потврду да су успјешно извршили резервацију. Овим ће се нарушити услов да ентите не може бити резервисан ако већ постоји резервација у изабраном термину и потребно је обезбједити да се то не деси. На слици испод приказане су повратне поруке за успјешне резервације у исто вријеме.



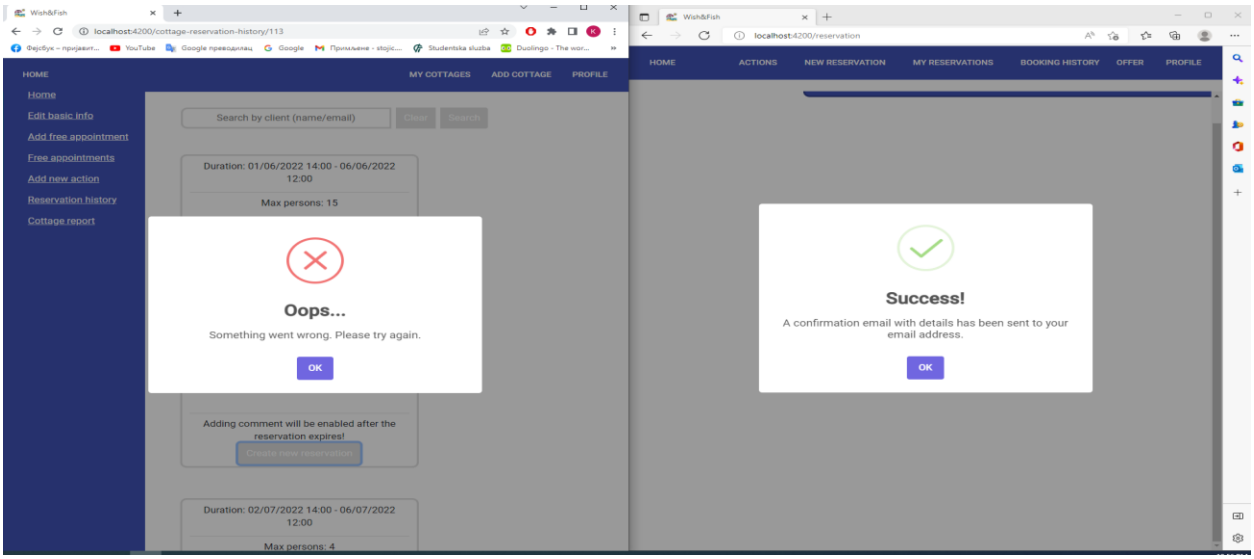



Приједлог рјешења: Рјешење је имплементирано у класама ReservationService, AppointmentService и BoatRepository/CottageRepository. У класи ReservationService  метода createReservation као и све методе које она позива(неке од њих су у AppointmentService класи) су означене као трансакционе и  у њима је извршено руковање изузецима како би клијент/власник знали да ли су успјешно или не извршили резервацију. Метода createReservation позива мотоду findOneById из класе CottageRepository/BoatRepository. Над том методом је извршено песимистичко закључавање. Као тип закључавања изабран је PESSIMISTIC_WRITE, чиме је онемогућено читање закључаног реда. На овај начин је ријешена конфликтна ситуација и онемогућемо резервисање истог ентитета у исто вријеме. На сликама испод је приказано како је то имплементирано у коду.

```java
@Transactional
public Appointment createReservation(CreateReservationDTO dto) {

    Set<Appointment> appointments = new HashSet<>();

    if (dto.getEntity() == 1) {
        try {

            Cottage cottage = cottageRepository.findOneById(dto.getEntityId());
            if (cottage == null || cottage.isDeleted()) {
                return null;
            }
            appointments = cottage.getAppointments();

        } catch(PessimisticLockingFailureException ex) {

            throw  new PessimisticLockingFailureException("Cottage already reserved!");

        }
```

```java
public interface CottageRepository extends JpaRepository<Cottage, Long> {

    @Lock(LockModeType.PESSIMISTIC_WRITE)
    @Query("select c from Cottage c where c.id = :id")
    @QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value ="0")})
    public Cottage findOneById(@Param("id")Long id);
}
```

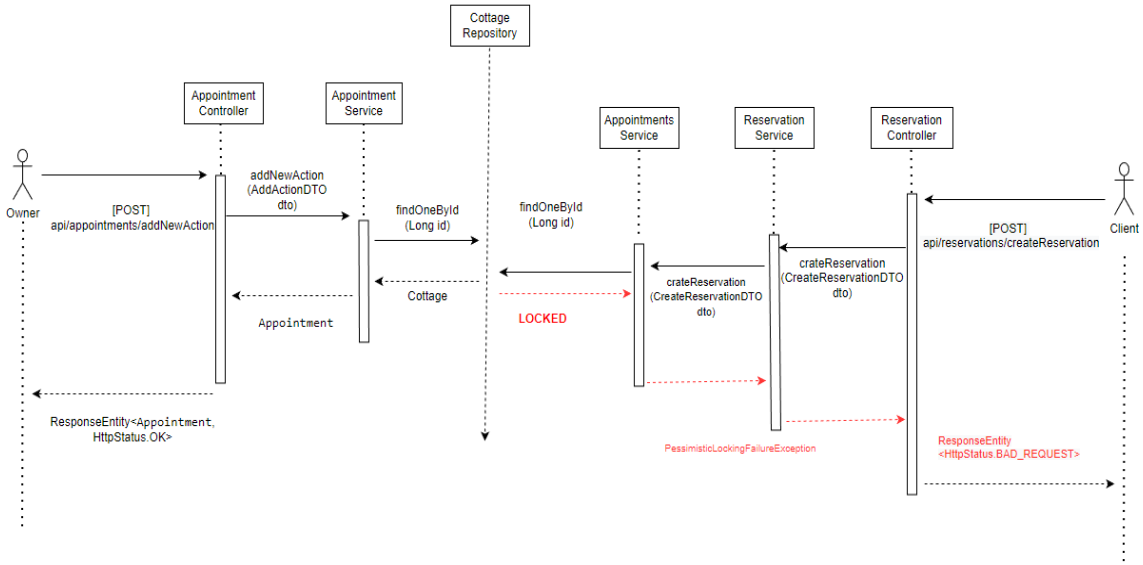
Успјешно ријешена конфликтна ситуација:



2. Власник викендице/брода или инструктор не може да направи акцију у исто вријеме кад и други клијент врши резервацију постојећег ентитета

Опис проблема: Претпоставимо да власник викендице/брода жели да дода нову акцију када и клијент жели да закаже тај ентит. Обојица бирају термин који се у неком периоду поклапа са термином другог и врше додавање акције, односно резервацију. Пошто је ентите слободан у оба случаја, оба корисника ће добити потврду да су успјешно извршили жељену акцију . Овим ће се нарушити услов да клијент не може да у исто вријеме да постоји и акција и обична резервација за ентитет и потребно је обезбједити да се то не деси.

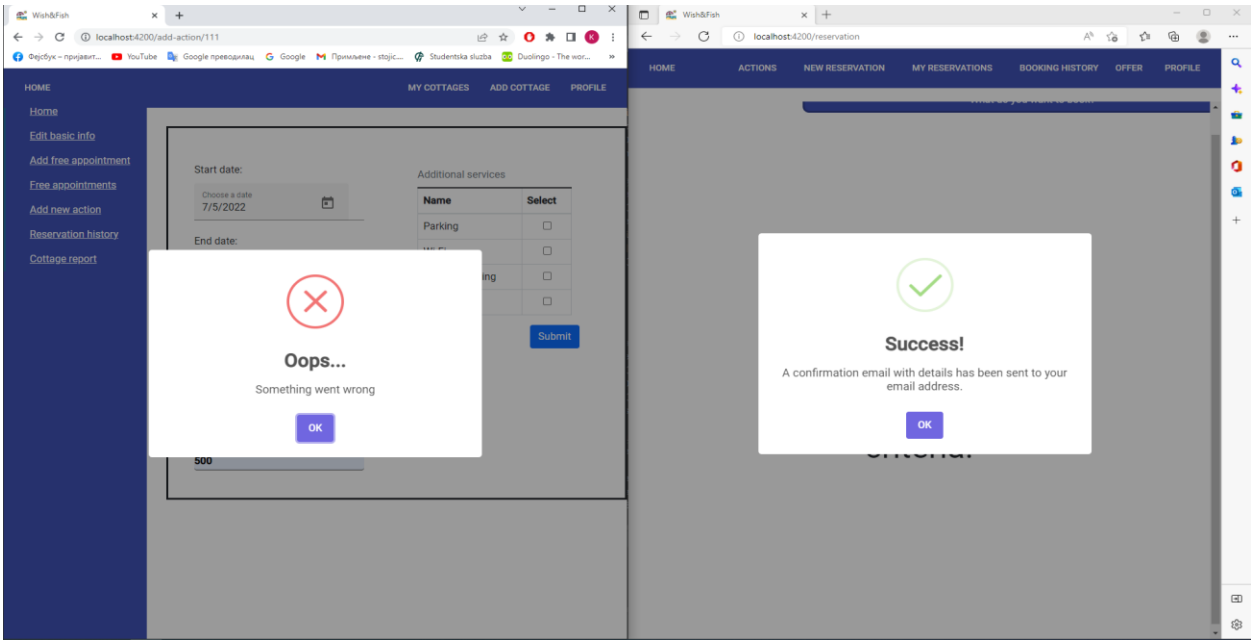
Приједлог рјешења: Рјешење је имплементирано у класама AppointmentService и BoatRepository/CottageRepository. У класи AppointmentService методе addNewAction/addNewActionBoat означене као трансакционе и  у њима је извршено руковање изузецима како би клијент/власник знали да ли су успјешно или не извршили жељену акцију. Метода addNewAction/addNewActionBoat позива мотоду findOneById из класе CottageRepository/BoatRepository. Над том методом је извршено песимистичко закључавање. Као тип закључавања изабран је PESSIMISTIC_WRITE, чиме је онемогућено читање закључаног реда. На овај начин је ријешена конфликтна ситуација и онемогућемо резервисање истог ентитета у исто вријеме. На сликама испод је приказано како је то имплементирано у коду.




```java
@Transactional
public Appointment addNewAction(AddActionDTO dto) throws MessagingException {
    LocalDateTime end = findDate(dto.getEndDate());
    LocalDateTime start = findDate(dto.getStartDate());
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");

    //ako postoji rezervacija
    for (Reservation r : reservationRepository.findAll()) {
        if (r.getAppointment().getCottage() != null) {
            if (dto.getId().equals(r.getAppointment().getCottage().getId()) && !r.getCanceled() &&
                    ((start.isAfter(r.getAppointment().getStartDate()) && start.isBefore(r.getAppointment().getEndDate()))
                            || (end.isAfter(r.getAppointment().getStartDate()) && end.isBefore(r.getAppointment().getEndDate())) ||
                            (start.isBefore(r.getAppointment().getStartDate()) && end.isAfter(r.getAppointment().getEndDate())))) {
                return null;
            }
        }
    }

    try{
        //ako postoji slobodan termin
        for (Appointment free : appointmentRepository.findAll()) {
            if (free.getCottage() != null) {
                if (!free.isDeleted() && dto.getId().equals(free.getCottage().getId())
                        && ((start.isAfter(free.getStartDate()) && end.isBefore(free.getEndDate()))
                        || (start.isEqual(free.getStartDate()) && end.isEqual(free.getEndDate())))) {
                    Cottage cottage = cottageRepository.findOneById(dto.getId());
```

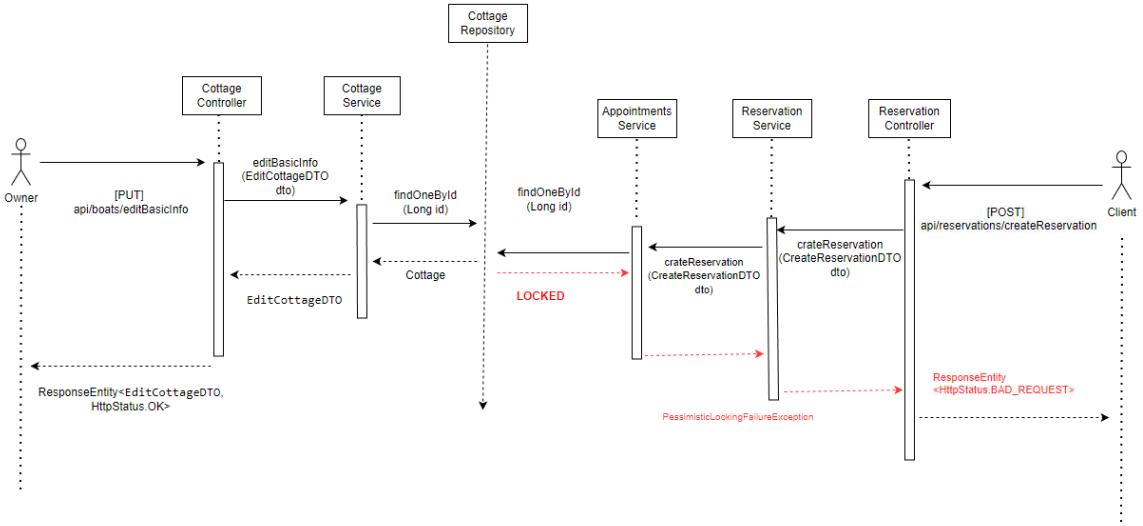
```java
@Transactional
public Appointment addNewActionBoat(AddActionDTO dto) throws MessagingException {
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");
    LocalDateTime end = findDate(dto.getEndDate());
    LocalDateTime start = findDate(dto.getStartDate());

    //ako postoji rezervacija
    for (Reservation r : reservationRepository.findAll()) {
        if (r.getAppointment().getBoat() != null) {
            if (dto.getId().equals(r.getAppointment().getBoat().getId()) && !r.getCanceled() &&
                    ((start.isAfter(r.getAppointment().getStartDate()) && start.isBefore(r.getAppointment().getEndDate()))
                            || (end.isAfter(r.getAppointment().getStartDate()) && end.isBefore(r.getAppointment().getEndDate())) ||
                            (start.isBefore(r.getAppointment().getStartDate()) && end.isAfter(r.getAppointment().getEndDate())))) {
                return null;
            }
        }
    }
    try {
        //ako postoji slobodan termin
        for (Appointment free : appointmentRepository.findAll()) {
            if (free.getBoat() != null) {
                if (!free.isDeleted() && dto.getId().equals(free.getBoat().getId())
                        && ((start.isAfter(free.getStartDate()) && end.isBefore(free.getEndDate()))
                        || (start.isEqual(free.getStartDate()) && end.isEqual(free.getEndDate())))) {
                    Boat b = boatRepository.findOneById(dto.getId());
                    //podijeli pronadjeni termin na 2 slobodna i 1 zauzeti
```

Успјешно ријешена конфликтна ситуација:



3. Власник викендице/брода или инструктор не може ажурира профил викендице/брода кад и други клијент врши резервацију постојећег ентитета

Опис проблема: Претпоставимо да власник викендице/брода жели да ажурира профил викендице/бора када и клијент жели да закаже тај ентит. Да не би дошло до тога да се информације о услугама и цијени истих промијене у моменту када клијент резервише ентит, односно да не би дошло до случаја да клијент резервише по једној цијени а власник у том моменту проијени исту, потребно је обезбједити да до тога не дође.




Приједлог рјешења: Рјешење је имплементирано у класама CottageService/BoatService и CottageRepository/BoatRepository. У класи CottageService/BoatService метода editBasicInfo је означена као трансакциона

и  у њој је извршено руковање изузецима како би клијент/власник знали да ли су успјешно или не извршили жељену акцију. Иста метода позива методу findOneById из класе CottageRepository/BoatRepository. Над том методом је извршено песимистичко закључавање. Као тип закључавања изабран је PESSIMISTIC_WRITE, чиме је онемогућено читање закључаног реда. На овај начин је ријешена конфликтна ситуација и онемогућемо да се информације о ентитетима промијене у току креирања резервације. На сликама испод је приказано како је то имплементирано у коду

```
@Transactional
public Boat editBasicInfo(EditBoatDTO editedBoat){
    try {
        Boat b = boatRepository.findOneById(editedBoat.getId());

        for (Reservation r : reservationRepository.findAll()) {
            if (r.getAppointment().getBoat() != null) {
                if ((r.getAppointment().getStartDate().isAfter(LocalDateTime.now()
                    return null;
```

```
public interface BoatRepository extends JpaRepository<Boat, Long> {

    @Lock(LockModeType.PESSIMISTIC_WRITE)
    @Query("select b from Boat b where b.id = :id")
    @QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value ="0")})
    public Boat findOneById(@Param("id")Long id);
}
```

Успјешно ријешена конфликтна ситуација: УФ