

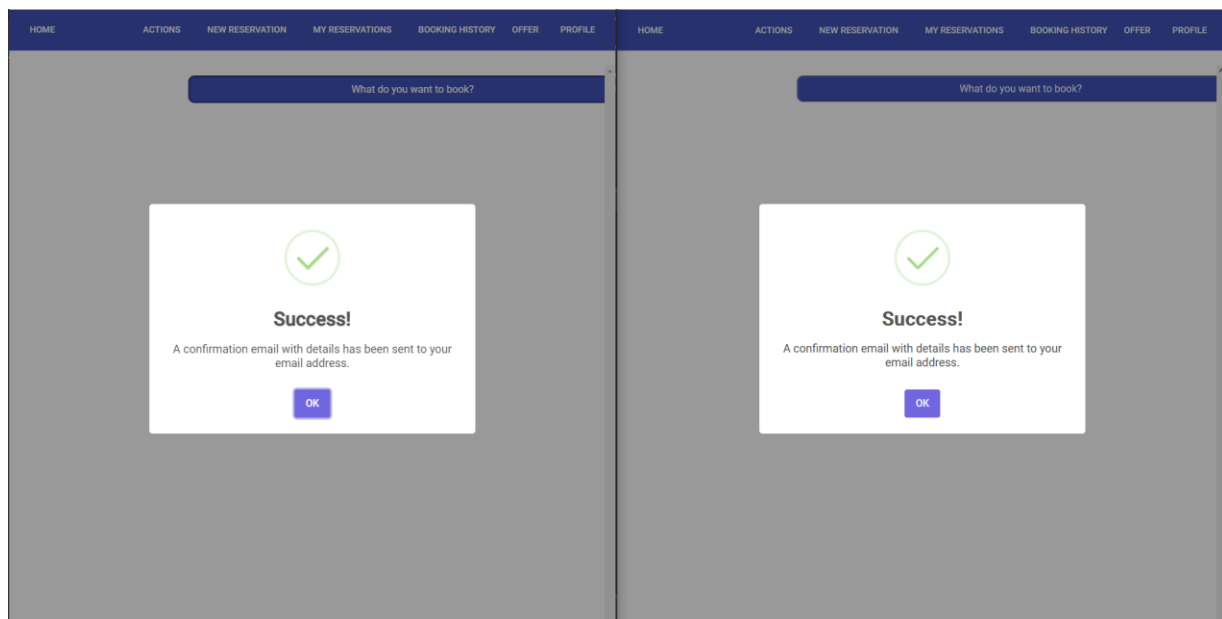
Конкурентни приступ ресурсима у бази

Студент 1 - Јелена Хрњак РА86/2018

Више клијената не могу да направе резервацију истог ентитета у исто (или преклапајуће) време

Опис конфликтне ситуације

Пре резервисања неког ентитета врши се провера доступности тог ентитета по критеријумима претраге које уносе клијенти. Свим клијентима се приказују слободни ентитети. Проблем настаје када више клијената покуша да изврши резервацију истог ентитета у исто или преклапајуће време. Свим корисницима би била омогућена резервација јер би при провери пре регистрације ентитет био слободан (ниједна резервација није извршена до краја и ентитет није резервисан). У оваквој ситуацији, исти ентитет би се резервисао више пута, што би довело до конфликта. На фотографији (слика 1.1) приказан је одговор приликом ове конфликтне ситуације.



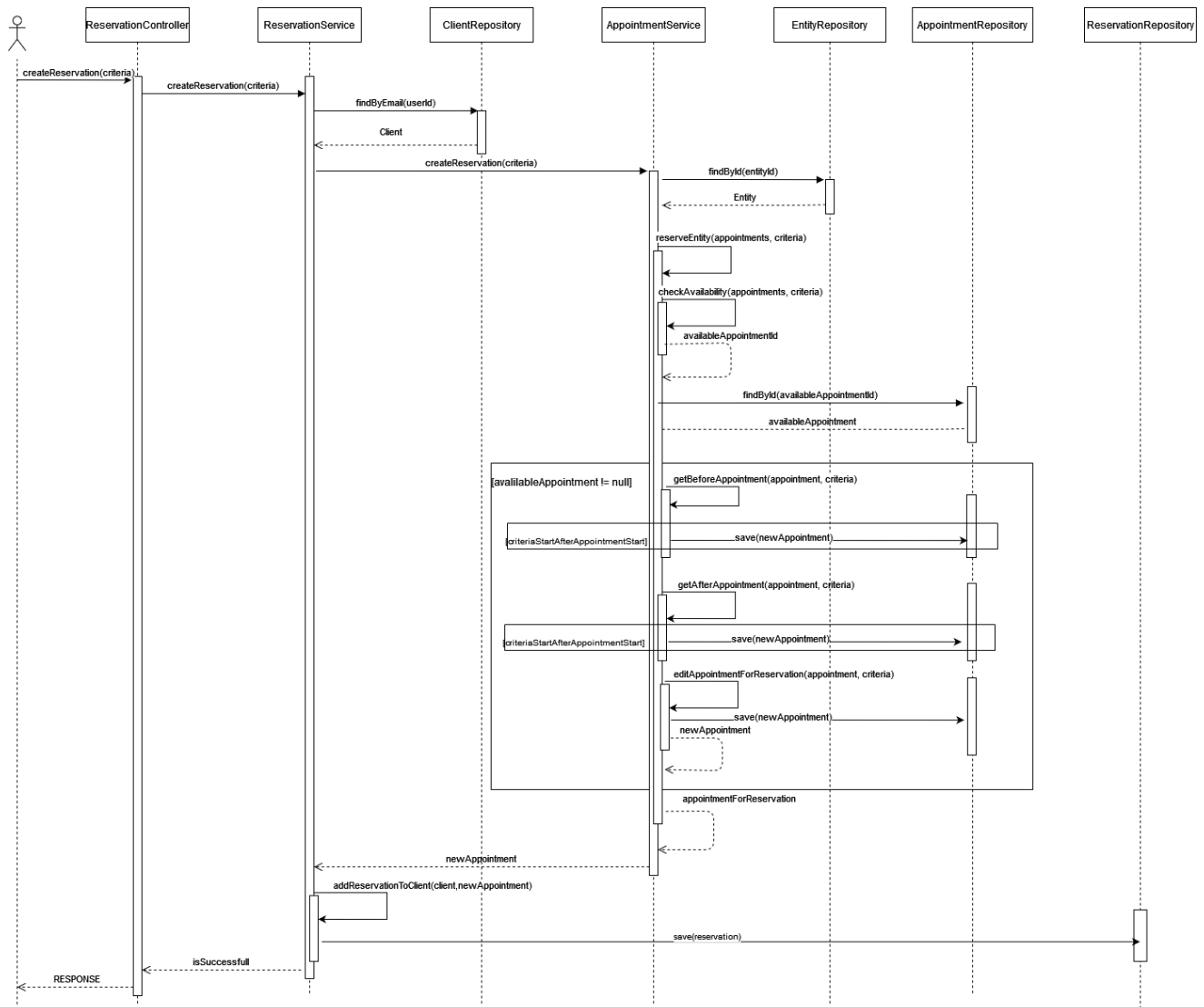
Слика 1.1 - одговор на покушај резервације више клијената у исто или преклапајуће време

Приказ конфликтне ситуације

Endpoint који се гађа приликом резервације је `/createReservation`, односно метода са истим називом `ReservationController` контролера. Та метода позива `createReservation` методу из `ReservationService` сервиса. Најпре се изврши провера да ли клијент који покушава да изврши резервацију има мање од три пенала и ако има позваће се метода `createReservation` из `AppointmentService` где ће се најпре пронаћи који ентитет је у питању (викендица, брод или авантура). Позива се метода `reserveEntity` која ће даље позивати методе за проверу доступности ентитета и дељење претходно слободног ентитета (уколико је то потребно). У методи `checkAvailability` извршиће се провера да ли је ентитет слободан у датом периоду и уколико јесте у методама `getBeforeAppointment` и `getAfterAppointment` десиће се дељење на термин пре и после заказаног термина. Након тога, у методи `editAppointmentForReservation` жељени термин заузети и додаће се додатне услуге које је клијент изабрао. На крају позваће се метода `addReservationToClient` из `ReservationService` сервиса и резервација ће се убацити у све клијентове резервације. Цео овај поступак приказан је на дијаграму тока у наставку (слика 1.2). При сваком добављању објекта по `id`-у или `email`-у извршена је провера да ли је тај објекат `null` и уколико јесте, резервација се обуставља. Ради прегледности ово није приказано на дијаграму. Такође, репозиторијум за викендице, бродове и авантуре биће приказан као један: `EntityRepository`.

Решење конфликтне ситуације

Проблем је решен песимистичким закључавањем метода за добављање ентитета који клијенти желе да резервишу. Закључане су методе `findOneById` у репозиторијумима викендица, бродова и аватнтура (слика 1.3). Приказано је закључавање у репозиторијум викендице `CottageRepository`, али је исто одрађено и у `BoatRepository` и у `FishingAdventureRepository`. Поред тога, методе које су позиване су анотиране анотацијом `@Transactional` (слика 1.4) и имплементације је уковирена `try/catch` блоковима који враћају `PessimisticLockingFailureException` са одговарајућом поруком (слика 1.5).



Слика 1.2 - дијаграм тока прве конфликтне ситуације

```

@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);
  
```

Слика 1.3 - песимистичко закључавање методе за добављање викендице помоћу id-a

```

@Transactional
public boolean createReservation(CreateReservationDTO dto) throws MessagingException, PessimisticLockingFailureException {
  
```

Слика 1.4 – анотација @Transactional изнад метода

```

try {

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

} catch(PessimisticLockingFailureException ex) {

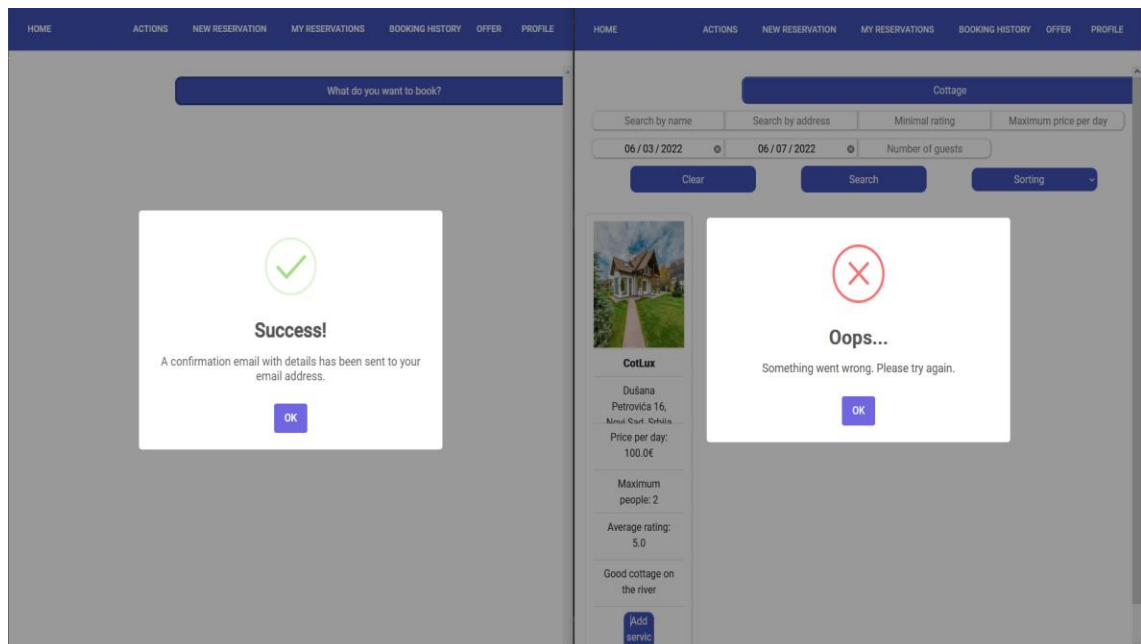
    throw new PessimisticLockingFailureException("Cottage already reserved!");

}

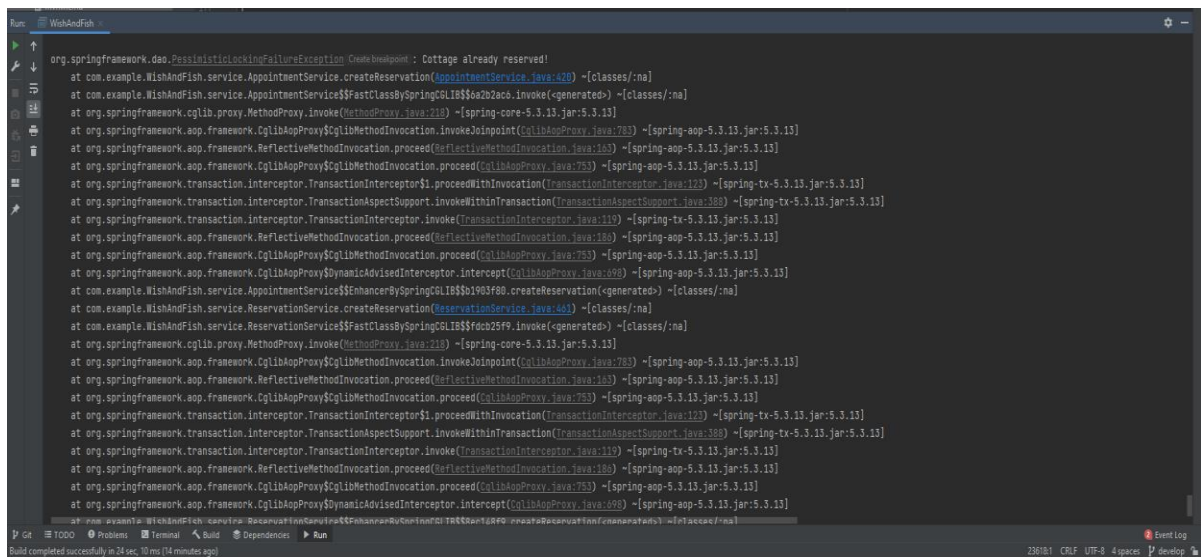
```

Слика 1.5

Након овога, више клијената неће моћи да резервишу исти ентитет у исто време, већ ће то успети само клијент који је први послао захтев (слике 1.6 и 1.7).



Слика 1.6 - одговор на покушај резервације више клијената у исто или преклапајуће време након решавања конфликта

The screenshot shows an IDE window with a console log. The log contains a series of Spring Framework logs, including transaction and proxy logs. At the bottom, there is a red error message: "org.springframework.dao.PessimisticLockingFailureException: Cottage already reserved!". The IDE interface includes a sidebar with project files, a top toolbar with icons for Git, TOOD, Problems, Terminal, Build, Dependencies, and Run, and a bottom status bar showing "236781 CR/LF UTF-8 4 spaces develop".

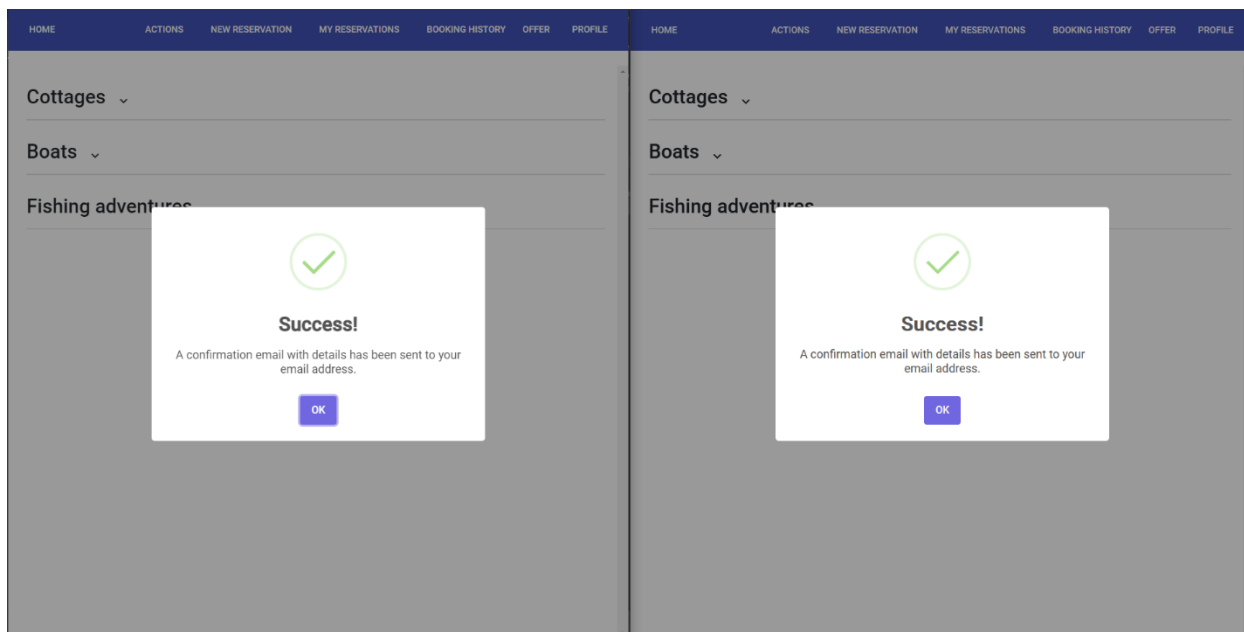
```
org.springframework.dao.PessimisticLockingFailureException: Cottage already reserved!
at com.example.WishAndFish.service.AppointmentService.createReservation(AppointmentService.java:420) ~[classes:/na]
at com.example.WishAndFish.service.AppointmentService$$FastClassBySpringGLIB$$6a2b2a0c.invoke(<generated>) ~[classes:/na]
at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:218) ~[spring-core-5.3.13.jar:5.3.13]
at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.invokeJoinpoint(CglibAopProxy.java:733) ~[spring-aop-5.3.13.jar:5.3.13]
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:183) ~[spring-aop-5.3.13.jar:5.3.13]
at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:733) ~[spring-aop-5.3.13.jar:5.3.13]
at org.springframework.transaction.interceptor.TransactionInterceptor$1.proceedWithInvocation(TransactionInterceptor.java:122) ~[spring-tx-5.3.13.jar:5.3.13]
at org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:388) ~[spring-tx-5.3.13.jar:5.3.13]
at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:119) ~[spring-tx-5.3.13.jar:5.3.13]
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:183) ~[spring-aop-5.3.13.jar:5.3.13]
at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:733) ~[spring-aop-5.3.13.jar:5.3.13]
at org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:688) ~[spring-aop-5.3.13.jar:5.3.13]
at com.example.WishAndFish.service.AppointmentService$$EnhancerBySpringGLIB$$1903f80.createReservation(<generated>) ~[classes:/na]
at com.example.WishAndFish.service.ReservationService.createReservation(ReservationService.java:461) ~[classes:/na]
at com.example.WishAndFish.service.ReservationService$$FastClassBySpringGLIB$$f0cb25f9.invoke(<generated>) ~[classes:/na]
at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:218) ~[spring-core-5.3.13.jar:5.3.13]
at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.invokeJoinpoint(CglibAopProxy.java:733) ~[spring-aop-5.3.13.jar:5.3.13]
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:183) ~[spring-aop-5.3.13.jar:5.3.13]
at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:733) ~[spring-aop-5.3.13.jar:5.3.13]
at org.springframework.transaction.interceptor.TransactionInterceptor$1.proceedWithInvocation(TransactionInterceptor.java:122) ~[spring-tx-5.3.13.jar:5.3.13]
at org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:388) ~[spring-tx-5.3.13.jar:5.3.13]
at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:119) ~[spring-tx-5.3.13.jar:5.3.13]
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:183) ~[spring-aop-5.3.13.jar:5.3.13]
at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:733) ~[spring-aop-5.3.13.jar:5.3.13]
at org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:688) ~[spring-aop-5.3.13.jar:5.3.13]
at com.example.WishAndFish.service.ReservationService.createReservation(ReservationService.java:461) ~[classes:/na]
```

Слика 1.7 - порука о грешци након истевременог резервисања истог ентитета

Више клијената не могу да направе резервацију истог ентитета на акцији у исто време

Опис конфликтне ситуације

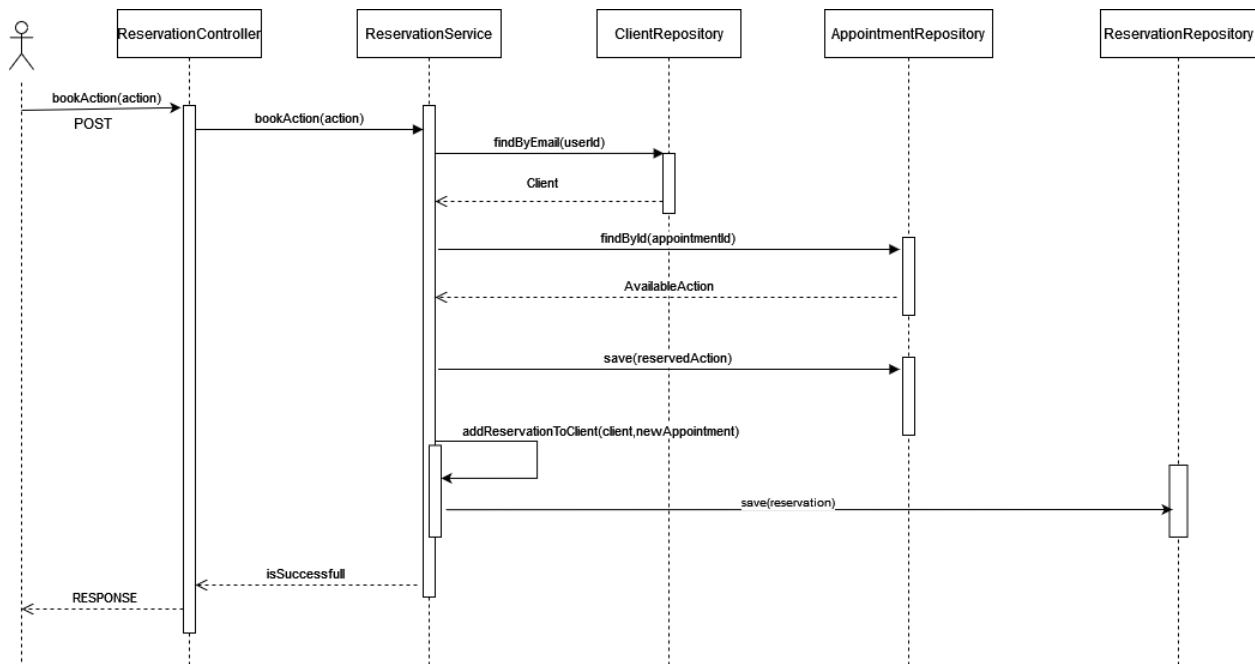
Слично као и код обичне резервације, свим клијентима се приказују слободне акције (осим ако је корисник раније није резервисао). Уколико више клијената покуша да резервише исту акцију у исто или преклапајуће време, свим корисницима би била омогућена резервација јер би при провери пре резервације акција била слободна (ниједна резервација акције није извршена до краја и акција није резервисана). У оваквој ситуацији, иста акција би се резервисала више пута, што би довело до конфликта. На фотографији (слика 2.1) приказан је одговор приликом ове конфликтне ситуације.



Слика 2.1 - одговор на покушај резервације исте акције више клијената у исто време

Приказ конфликтне ситуације

Endpoint који се гађа приликом резервације акције је [/bookAction](#), односно метода са истим називом [ReservationController](#) контролера. Та метода позива [bookAction](#) методу из [ReservationService](#) сервиса. Најпре се изврши провера да ли клијент који покушава да изврши резервацију има мање од три пенала, да ли је слободни термин који клијент жели да резервише акција која је слободна. Ако су све провере прошле термин акције ће се резервисати и сачувати помоћу [AppointmentRepository](#)-ja. На крају ће се позвати метода [addReservationToClient](#) и резервисана акција ће се убацити у све клијентове резервације. Цео овај поступак приказан је на дијаграму тока у наставку (слика 2.2). При сваком добављању објекта по id-у или email-у извршена је провера да ли је тај објекат null и уколико јесте, резервација се обуставља. Ради прегледности ово није приказано на дијаграму.



Слика 2.2 - дијаграм тока друге конфликтне ситуације

Решење конфликтне ситуације

Проблем је решен песимистичким закључавањем методе за добављање слободног термина (акције) коју клијенти желе да резервишу. Закључана је метода [findOneById](#) у [AppointmentRepository-ju](#) (слика 2.3). Поред тога, методе које су позиване су анотиране анотацијом [@Transactional](#) (слика 2.4) и имплементације је укловирена [try/catch](#) блоковима који враћају [PessimisticLockingFailureException](#) са одговарајућом поруком (слика 2.5).

```

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

Слика 2.3 - песимистичко закључавање методе за добављање акције помоћу id-а

```

@Transactional
public boolean bookAction(ActionReservationDTO dto) throws MessagingException {
  }
  
```

Слика 2.4 – анотација [@Transactional](#) изнад метода

```

try {
    Appointment appointment = appointmentRepository.findOneById(dto.getAction());

    if (appointment == null || !appointment.getIsAction() || appointment.isDeleted() || appointment.getReserved()) {
        return false;
    }

    appointment.setReserved(Boolean.TRUE);
    appointmentRepository.save(appointment);

    return addReservationToClient(client, appointment);
} catch (PessimisticLockingFailureException ex) {

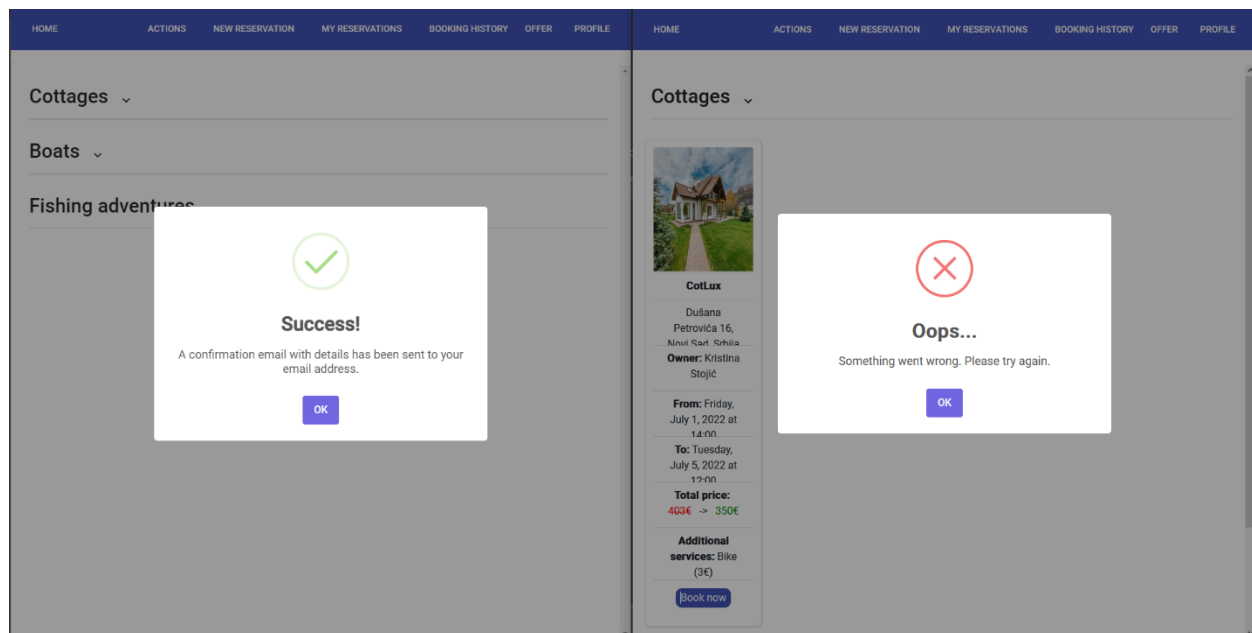
    throw new PessimisticLockingFailureException("Action already booked!");
}
}

```

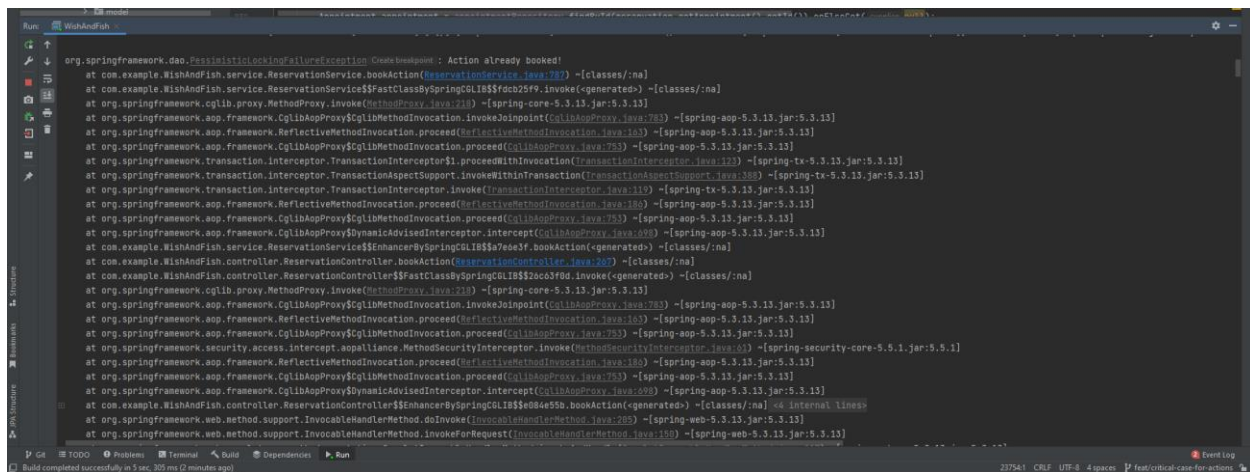
Након овога, више клијената неће моћи да резервишу исту акцију у исто време, већ

Слика 2.5

ће то успети само клијент који је први послао захтев (слике 2.6 и 2.7).



Слика 2.6 - одговор на покушај резервације исте акције од стране више клијената у исто или преклапајуће време након решавања конфликта



Слика 2.7 - порука о грешци након истовременог резервисања исте акције

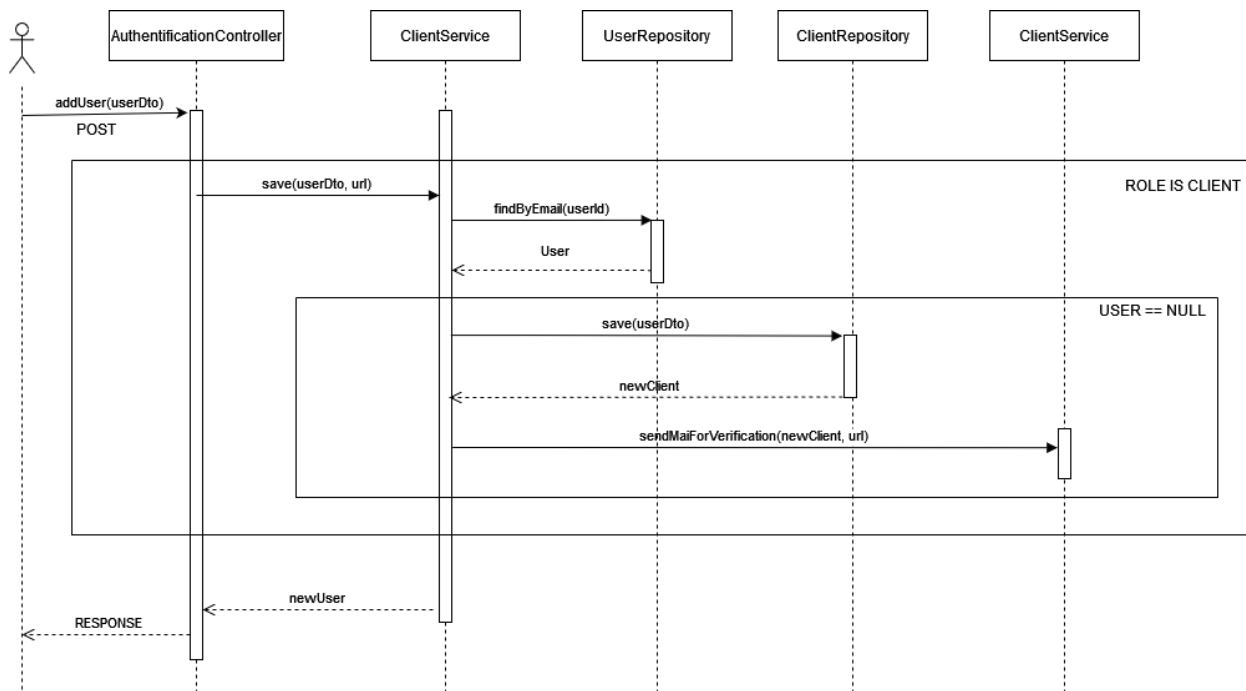
Више клијената не могу се региструју у исто време са истим email-ом

Опис конфликтне ситуације

Email корисника у систему је јединствен и самим тим при регистрацији се врши провера да ли у бази већ постоји корисник са унетим email-ом. Уколико постоји, захтев за регистрацију од новог корисника треба да буде одбијен. Међутим, уколико два или више корисника у довољно блиском временском опсегу пошаљу захтев за регистрацију са истим email-ом, постоји могућност да ће одговор при провери постојања email-а, за све захтеве бити негативан (јер регистрација није спроведена до краја ни за једног корисника). Уколико до овога дође, нарушило би се unique ограничење за email адресу и дошло би до конфликта.

Приказ конфликтне ситуације

Endpoint који се гађа приликом регистрације је `/signup`, односно метода `addUser` из `AuthenticationController` контролера. Првенствено се проверава рола за коју је послат захтев за регистрацију и у зависности од ње се даље извршава ток. У даљем тексту ће бити обрађен случај када је рола `ROLE_CLIENT`. Након провере, захтеву ће се доделити верификациони код и позваће се метода `save` из `ClientService-a`. Провериће се постојање корисника са мејлом из захтева. Уколико постоји корисник са тим email-ом захтев ће бити одбијен, а уколико не постоји наставиће се извршавање. Позивом методе `save` из `ClientRepository-ja` сачуваће се нови клијент са подацима из захтева и послаће му се верификациони email у методи `sendMailForVerification` из `EmailService-a`. Цео овај поступак приказан је на дијаграму тока у наставку (слика 3.1).



Слика 3.1 - дијаграм тока треће конфликтне ситуације

Решење конфликтне ситуације

И ова конфликтна ситуација аје решен песимистичким закључавањем. Закључана је метода за добављање свих корисника [findAllPesimistic](#) [UserRepository](#)-ја (слика 3.2). Провера уникатности email-а пребачена је у [ClientService](#) у методу [findByEmail](#) која пролази кроз све кориснике које враћа закључана метода [findAllPesimistic](#) из [UserRepository](#)-ја. Поред тога, методе које су позиване су анотиране анотацијом [@Transactional](#) и имплементације је укловирена [try/catch](#) блоковима који враћају [PessimisticLockingFailureException](#) са одговарајућом поруком (слика 3.3).

```

@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select u from User u")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
List<User> findAllPesimistic();

```

Слика 3.2 - песимистичко закључавање методе за добављање свих корисника

```
@Transactional
public User findByEmail(String email) {
    try {
        for (User u : userRepository.findAllPessimistic()) {
            if (u.getEmail().equals(email))
                return u;
        }
    } catch (PessimisticLockingFailureException ex) {
        throw new PessimisticLockingFailureException("User with this email already exists!");
    }
    return null;
}
```

Слика 3.3 – анотација @Transactional, try/catch блок са грешком и поруком, нова метода за проверу уникатности email адресе