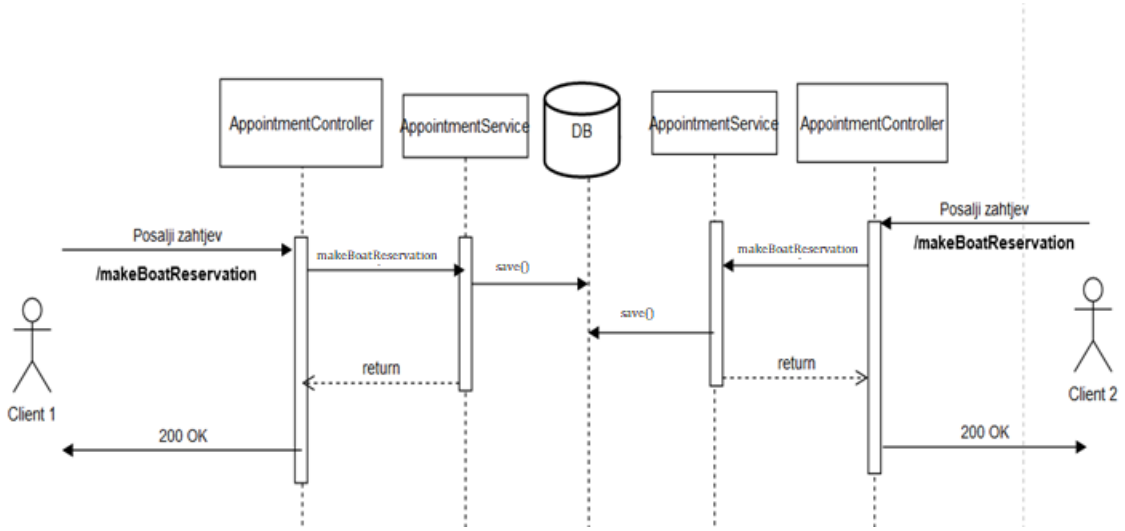
Конкурентни приступ – Бојан Продановић RA34/2018

1. Више истовремених клијената не могу да направе резервацију ентитета у исто или преклапајуће вријеме

Конфликтна ситуација подразумјева да истовремено два корисника желе да приступе резервисању истога ентитета. Будући да једна резервација може гласити само на једног корисника након што један (онај први пристигли) од два конкурентна захтјева прође приликом извршавања другог доћи ће до преписивања (overwriting) корисника који је први прошао кроз методе у сервису. Тако, иако наизглед успјешно извршена резервација, неће се налазити код првог корисника у листи резервисаних ентитета.



Слика 1: Графички приказ прве конфликтне ситуације

Са секвенционалног дијаграма се види неки главни ток одвијања конфликтне ситуације с тим да су изостављени поједини детаљи који се тичу логике ради једноставнијег приказа. У конкретном примјеру је написана метода резервације брода, али се потпуно исто односи и на викендицу и услугу инструктора.

Ситуација је ријешена помоћу песимистичког закључавања. Ред у табели се закључава одмах, нема верзирања као код оптимистичког закључавања. Save() методу у репозиторијуму резервација анотирамо са @Lock(LockModeType.PESSIMISTIC_READ) – као што се на слици испод може и видјети. Методе које врши резервације су анотиране са @Transactional (Слика 3, приказане двије од три методе).

Постоји још један начин рјешавања ове ситуације. Могуће је закључати резервацију приликом њеног добављања у сервису од стране клијента који први врши резервисање такође помоћу песимистичког закључавања и онда другопристиглом захтјеву се врати изузетак. Међутим прво рјешење се чини дјелотворније.

```
@Lock(LockModeType.PESSIMISTIC_READ)
public Appointment save(Appointment a);
```

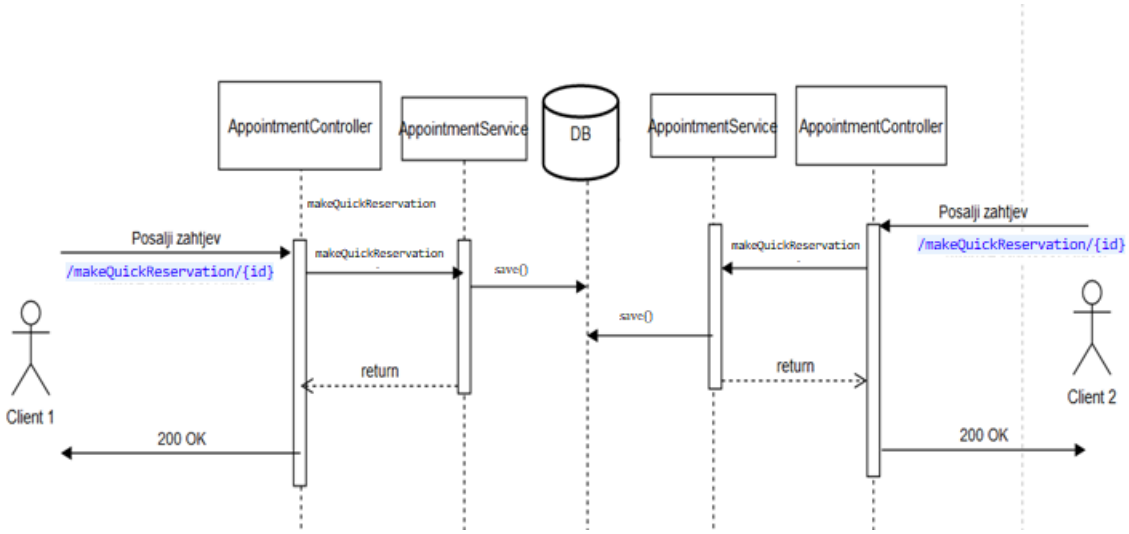
Слика 2: Дио рјешења у репозиторијуму

```
@Override
@Transactional
public String makeBoatReservation(BoatAppointment appointment) {
    // TODO Auto-generated method stub
    TokenBasedAuthentication aut = (TokenBasedAuthentication) SecurityContextHolder.getContext().get
    User usr = (User) aut.getPrincipal();
    appointment.setUser(usr);
    ((AppointmentRepository)repository).save(appointment);
    emailService.sendReservationMail(usr);
    return null;
}

@Override
@Transactional
public String makeResortReservation(ResortAppointment appointment) {
    // TODO Auto-generated method stub
    TokenBasedAuthentication aut = (TokenBasedAuthentication) SecurityContextHolder.getContext().get
    User usr = (User) aut.getPrincipal();
    appointment.setUser(usr);
    ((AppointmentRepository)repository).save(appointment);
    emailService.sendReservationMail(usr);
    return null;
```

Слика 3: Дио рјешења у сервису

2. Више клијената не могу истовремено да направе резервацију истог ентитета на акцији

Конфликтна ситуација по ономе што подразумјева је истовјетна са првом која је рјешавана. Истовремено два корисника желе да приступе резервисању истога ентитета, само што се овдје ради о тзв. „брзој резервацији“. Будући да једна резервација може гласити само на једног корисника након што један (онај први пристигли) од два конкурентна захтјева прође приликом извршавања другог доћи ће до преписивања (overwriting) корисника који је први прошао кроз методе у сервису. Тако, иако наизглед успјешно извршена резервација, неће се налазити код првог корисника у листи резервисаних ентитета.



Слика 4: Графички приказ друге конфликнте ситуације

Ситуација је ријешена као и у претходном примјеру. Метода save() у репозиторијуму која служи за модификацију постојећих (одабраних) резервација је анотирана са @Lock, а тип је постављен на PESSIMISTIC_READ. Док је метода у сервису анотирана са @Transactional, што се на слици испод може и видјети.

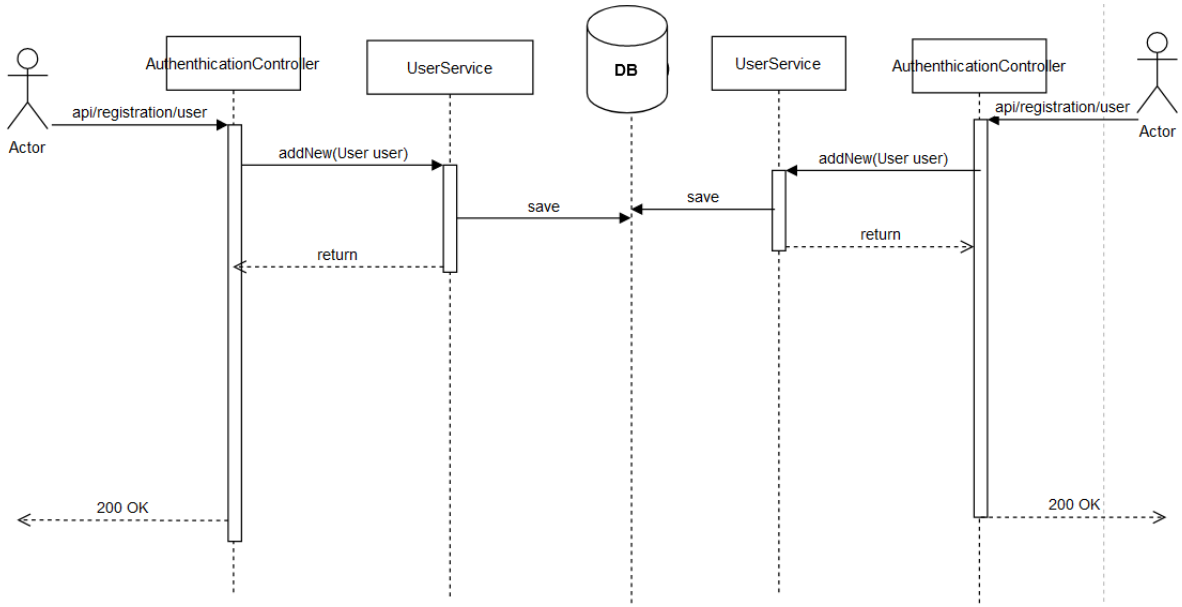
```
@Override
@Transactional
public String makeQuickReservation(Integer id) {
    // TODO Auto-generated method stub
    TokenBasedAuthentication aut = (TokenBasedAuthentication) SecurityContextHolder.getContext().getAuthentication
    User usr = (User) aut.getPrincipal();
    Appointment appointment = repository.getById(id);

    if(appointment.getType()==AppointmentType.BOAT) {
```

Слика 5: Графички приказ дијела рјешења у сервису

Из секвенцијалног дијаграма изостављен приказ класе репозиторијума јер је беспотребно. Позива се само CRUD метода save().

3. Више клијената жели да се региструје помоћу исте мејл адресе

До ове конфликтне ситуације долази када два корисника желе да се региструју на апликацију користећи исту и-мејл адресу. По спецификацији и правилу једна и-мејл адреса може припадати само једном налогу стога би овдје дошло до нарушавања јединствености.



Слика 6: Графички приказ конфликтне ситуације

Проблем је ријешен песимистичким закључавањем слично као и у претходна два случаја тако што је метода репозторијума анотирана са @Lock, а за разлику од претходних случајева тип „браве“ је постављен на PESSIMISTIC_WRITE – ради се о ексклузивном закључавању гдје је немогуће читање, писање и измјене датог ентитета док претходник не заврши са тиме. А када претходник заврши, следећи корисник који намјерава да се региструје са истим мејлом биће одбијен у редовној процедури јер та и-мејл адреса већ постоји у бази, тј. постоји корисник са том и-мејл адресом. Метода у сервису је анотирана са @Transactional, што се на слици испод може и видјети.

```
@Transactional
@PostMapping("api/registration/user")
public ResponseEntity<?> register(@RequestBody RegistrationDTO dto) {
    try {
        User user = userRegistrationMapper.convertToEntity(dto,User.class);
        usersService.addNew(user);
        emailService.sendRegisterConfirmationMail(user);
        return ResponseEntity.ok(user);
    } catch (RegistrationException ex) {
        System.out.println(ex.getMessage());
        return ResponseEntity.status(400).body(ex.getMessage());
    }
}
```

```java
@Lock(LockModeType.PESSIMISTIC_WRITE)
public User save(User user);
```