Evolution Fitness

Concurrent and Distributed System Project

Pulizzi Maurizio – a.a. 2020/2021

In Supremæ Dignitatis
1343
Università di Pisa

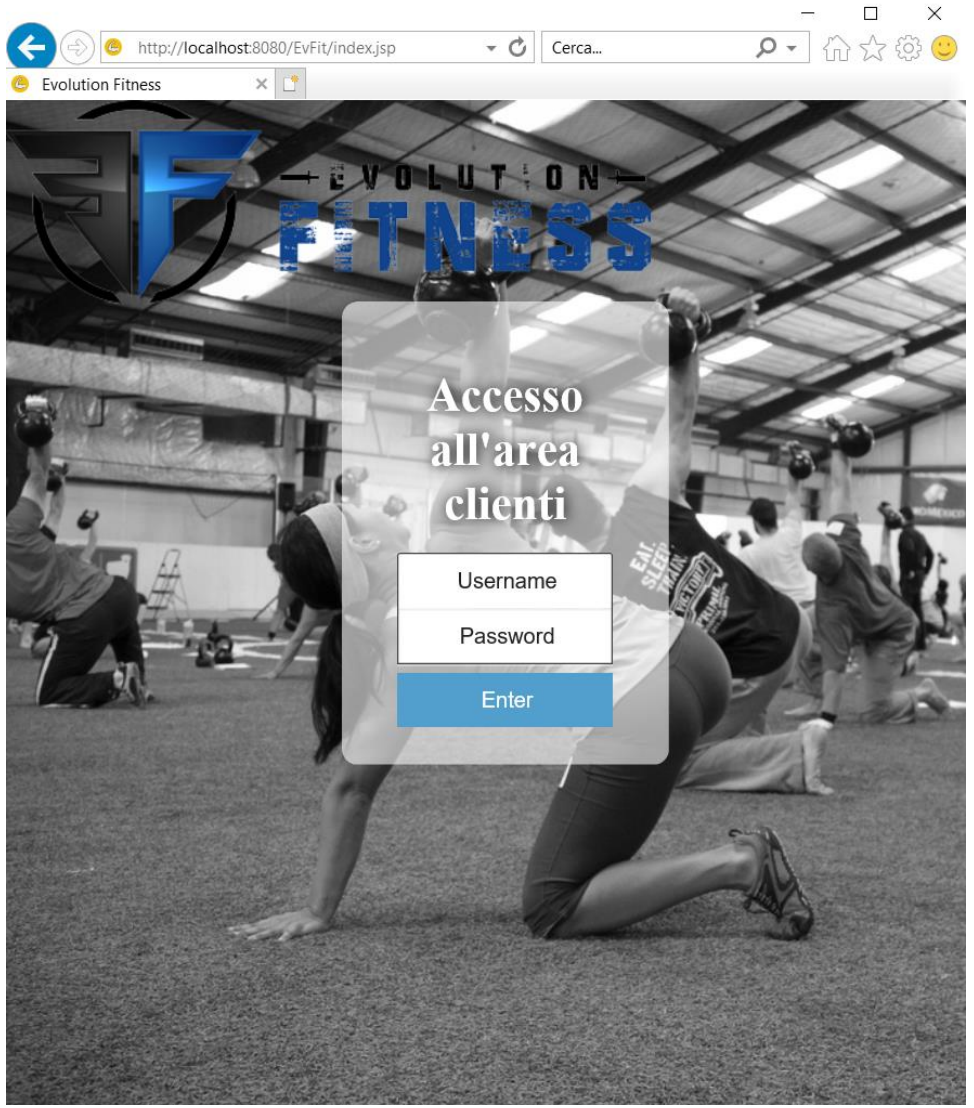
Use Cases

Use Case - Login

- The user needs to login for using the application.

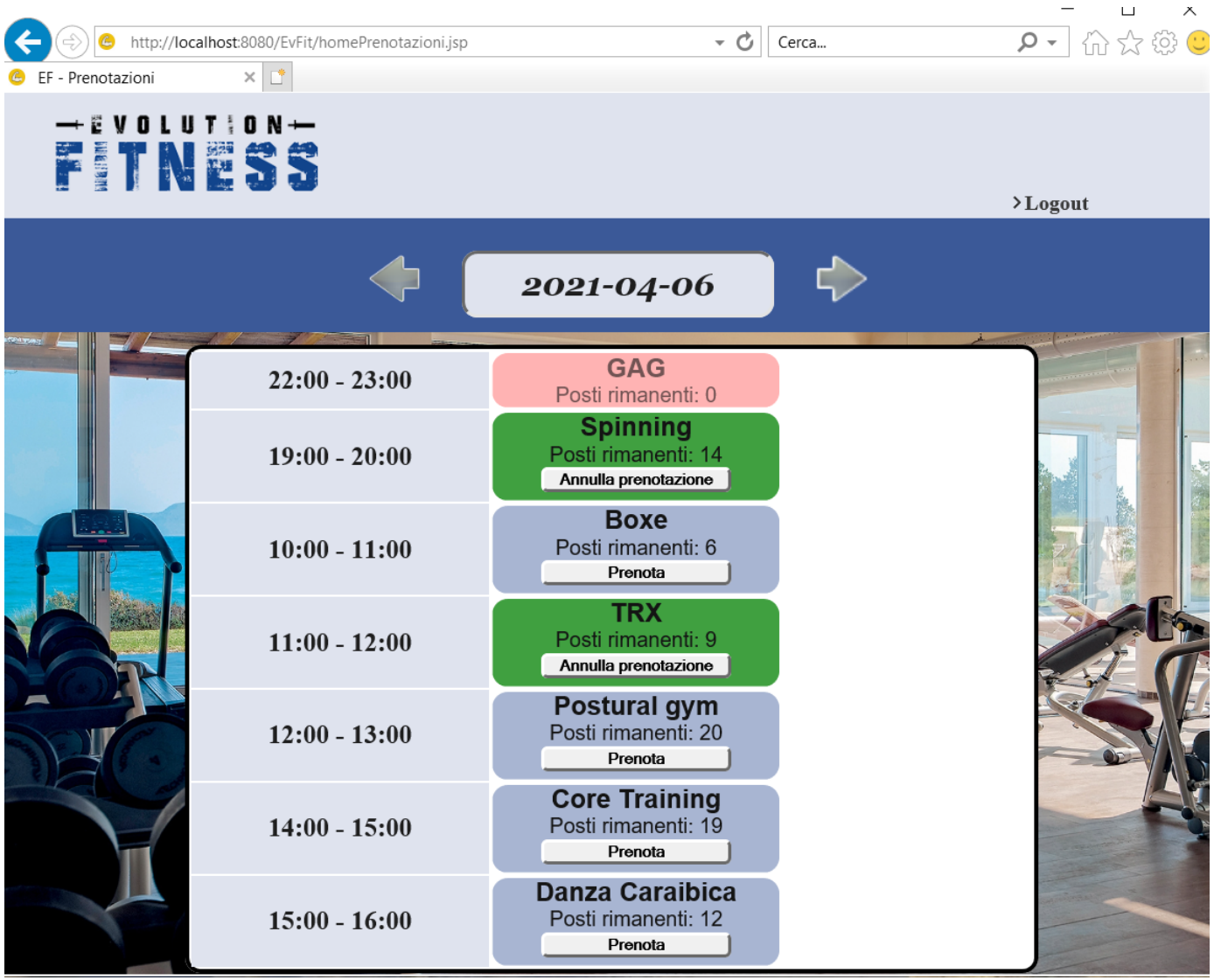
- If the user is a ***customer*** of the gym, he will be redirected to the customer panel.

- Il the user is a gym ***manager***, he will be redirected to the manager panel.



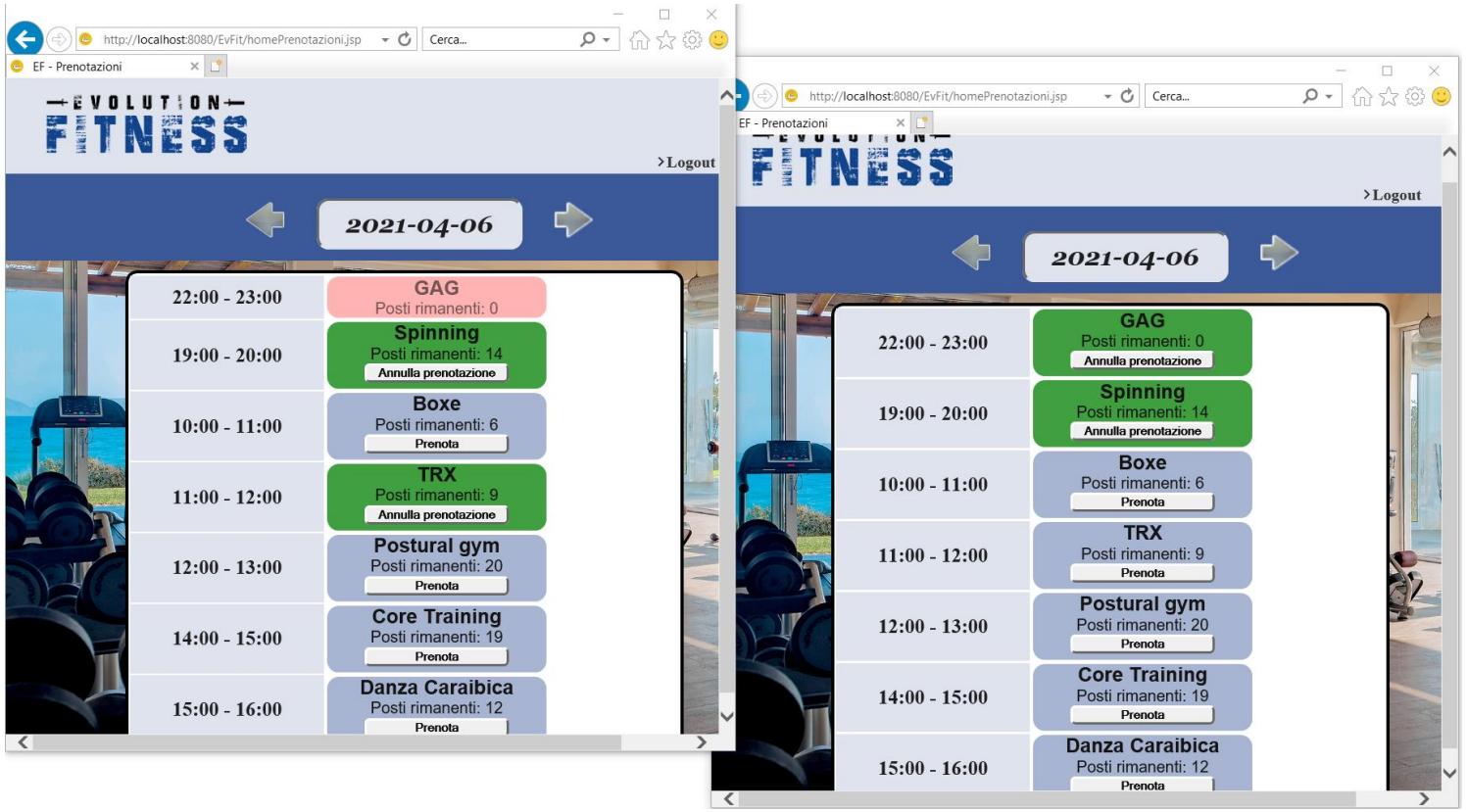
Use Case – customer panel

- A customer can book all those courses in the calendar that still have places available.
- In the panel, the courses booked by the user has a red box, the user can cancel the reservation for these courses.



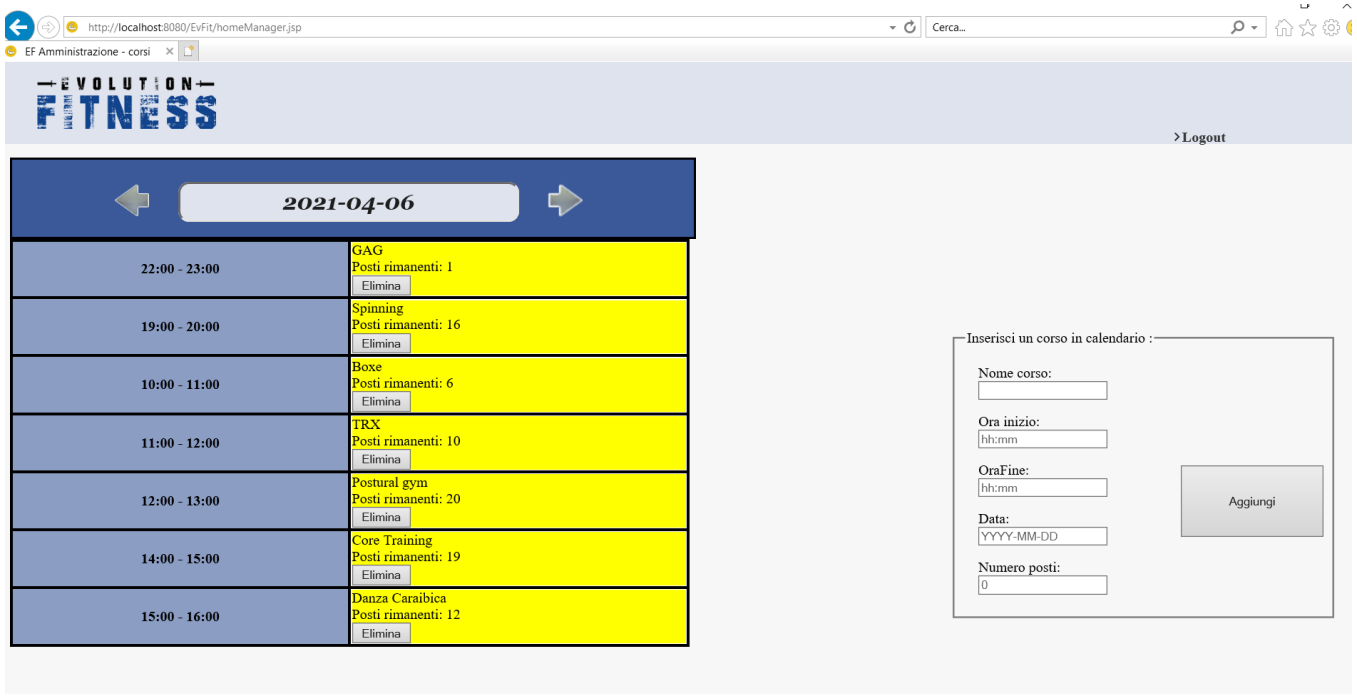
Use Case – customer panel

- Each user has a personalized view based on the courses they have booked.

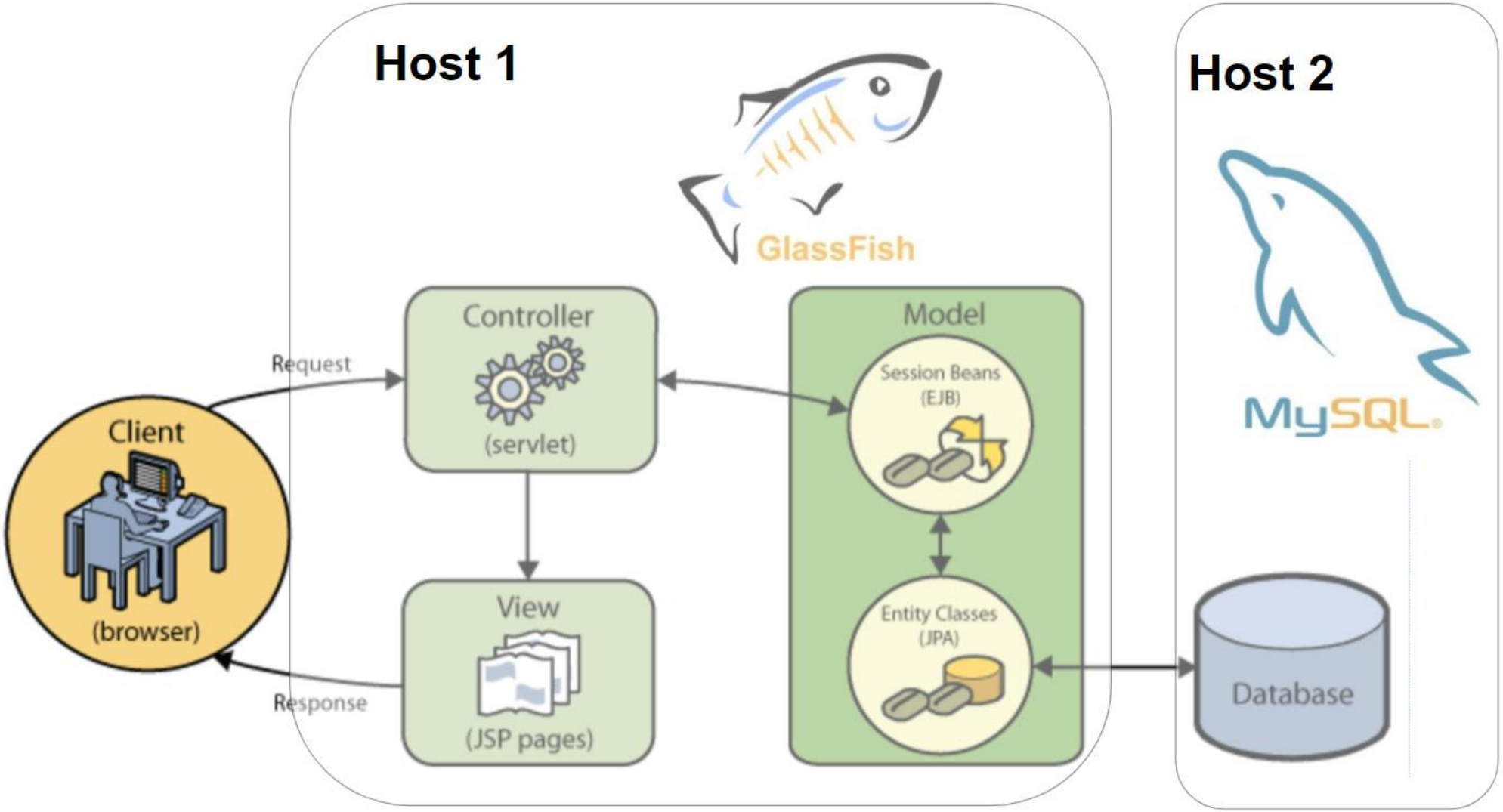


Use Case – manager panel

- The manager can add new courses to the calendar or remove courses already scheduled.



System Structure



Concurrency Management

Concurrency Management

- Singleton Session Bean used for handling the concurrency between clients
  - BEAN concurrency management type used:

    ➔ "The developer of the singleton is *responsible* for ensuring that the state of the singleton is synchronized across all clients."

```
@ConcurrencyManagement(ConcurrencyManagementType.BEAN)
@Singleton
@LocalBean
public class coursesManager implements coursesManagerLocal {

    coursesManager Implementation..

}
```

Concurrency Management – Used Data Structures

```
private ReadWriteLock locksMapLock;
private HashMap<Integer, Lock> locks;
private ConcurrentHashMap<Integer, Integer> coursesMap;
```

- locks:
  - one RentrantLock for each course
  - key = idCourse
- locksMapLock:
  - ReentrantReadWriteLock
  - protect the locks hashmap from concurrency problems when new courses are created or deleted.
- coursesMap:
  - key = idCourse
  - stores the number of remaining seats for each course

bookCourse Method

```
public boolean bookCourse(int idCourse, int idUser) {
    boolean res = false;
    locksMapLock.readLock().lock();
    Lock lock = locks.get(idCourse);
    lock.lock();
    try{
        int currentseats=coursesMap.get(idCourse);
        if(currentseats > 0){
            //DO RESERVATION
            Course course=courseFacade.find(idCourse);
            Reservation r = new Reservation(course, userFacade.find(idUser));
            reservationFacade.create(r);
            course.setRemainingPeopleNumber(currentseats-1);
            courseFacade.edit(course);
            //UPDATE COURSEMAP
            coursesMap.compute(idCourse, (k, v) -> (currentseats-1));
            res = true;
        }
    }finally{
        lock.unlock();
        locksMapLock.readLock().unlock();
    }
   return res;
}
```

deleteReservation Method

```
public boolean deleteReservation(int idCourse, int idUser){
    boolean res = false;
    locksMapLock.readLock().lock();
    Lock lock = locks.get(idCourse);
    lock.lock();
    try{
        int currentseats=coursesMap.get(idCourse);
        //DO DELETE
        Course course=courseFacade.find(idCourse);
        User user = userFacade.find(idUser);
        Reservation r = reservationFacade.findByIduserIdcourse(user, course);
        reservationFacade.remove(r);
        course.setRemainingPeopleNumber(currentseats+1);
        courseFacade.edit(course);
        //UPDATE COURSEMAP
        coursesMap.compute(idCourse, (k, v) -> (currentseats+1));
        res = true;
    }finally{
        lock.unlock();
        locksMapLock.readLock().unlock();
    }
    return res;
}
```

addCourse Method

```java
public boolean addCourse(String name, Date date,
        Date startTime, Date endTime, int maxPeopleNumber){
    boolean res=false;

    Course course = createCourse(name, date, startTime, endTime, maxPeopleNumber);

    locksMapLock.writeLock().lock();
    //NEW LOCK CREATION
    Lock lock = new ReentrantLock();
    locks.put(maxIdCourse, lock);
    coursesMap.put(maxIdCourse, maxPeopleNumber);

    lock.lock();
    locksMapLock.writeLock().unlock();
    locksMapLock.readLock().lock();

    try{
        courseFacade.create(course);
        res = true;
    }finally{
        lock.unlock();
        locksMapLock.readLock().unlock();
    }
    return res;
}
```

deleteCourse Method

```
public boolean deleteCourse(int idCourse) {
    boolean res = false;
    locksMapLock.writeLock().lock();
    Lock lock = locks.get(idCourse);
    lock.lock();
    try{
        //COURSE DELETION
        Course course=courseFacade.find(idCourse);
        courseFacade.remove(course);
        //REMOVING COURSE LOCK FROM THE LOCK LIST
        coursesMap.remove(idCourse);
        locks.remove(idCourse);

        res = true;
    }finally{
        lock.unlock();
        locksMapLock.writeLock().unlock();
    }
   return res;
}
```

Thank you for the attention. Questions?