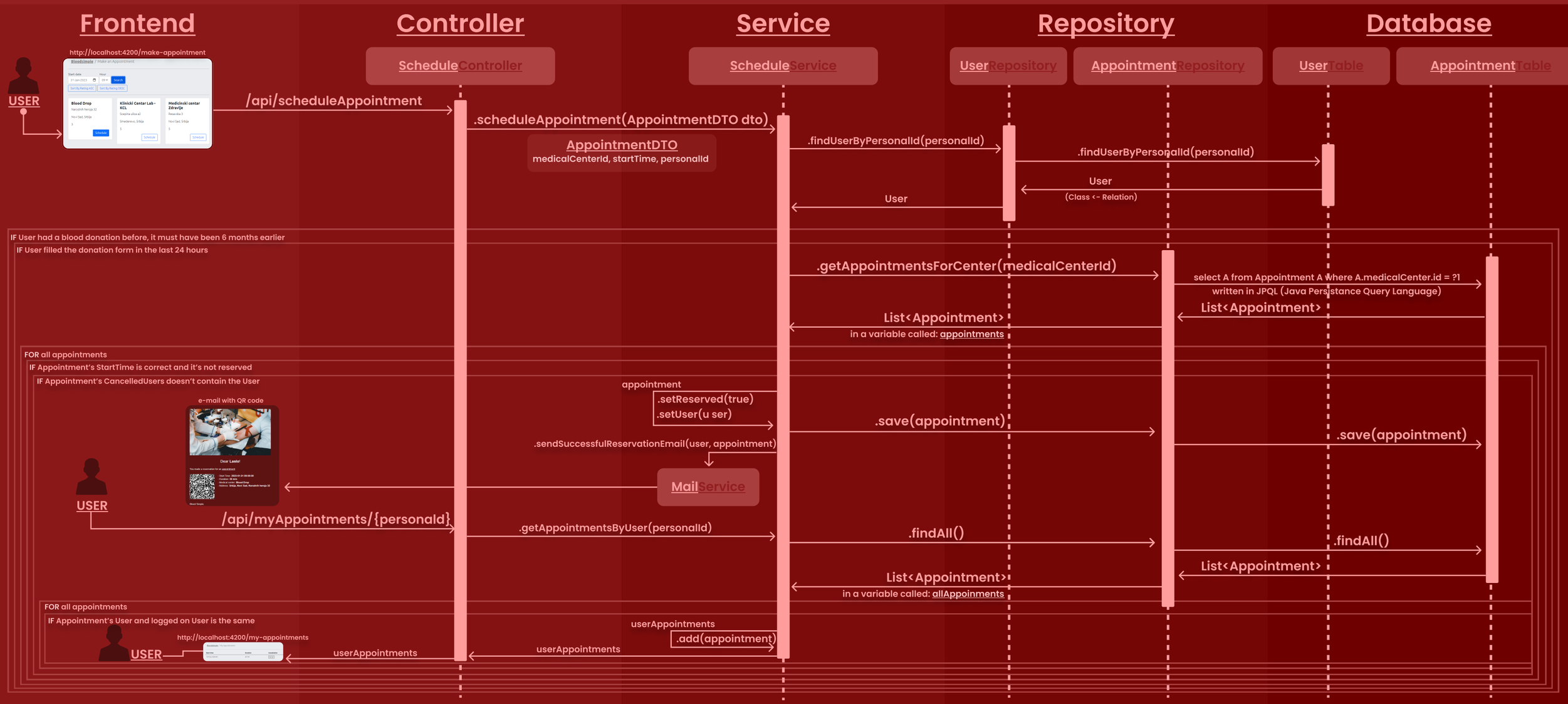
BloodSimple

7.5 Concurrent access to resources in the database

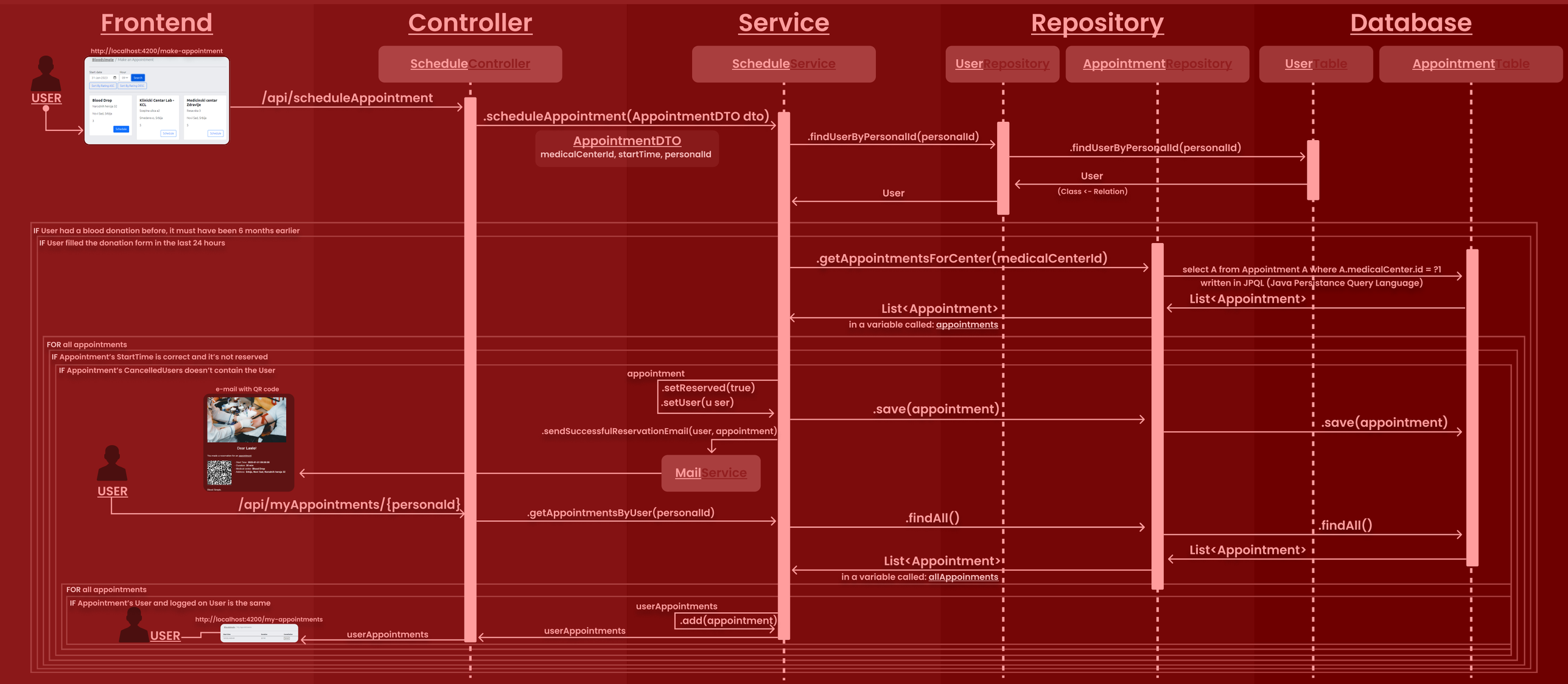
Concurrent access #1: The terms of the appointments that are predefined may not be reserved by more different users



BloodSimple

7.5 Concurrent access to resources in the database

Concurrent access #2: Multiple simultaneous users of the application cannot book an appointment that has become unavailable in the meantime



BloodSimple

7.5 Concurrent access to resources in the database

Concurrent access #1: The terms of the appointments that are predefined may not be reserved by more different users

Concurrent access #2: Multiple simultaneous users of the application cannot book an appointment that has become unavailable in the meantime

The two situations are characterized by the fact that multiple clients cannot make a reservation for the same entity at the same/overlapping time.
If something like this were allowed, there would be a mismatch of data, or more precisely, the database would not be in a consistent state.
Two appointments would be made for the same entity with the same time period for two different users.

One of the effective ways to resolve these situations is to use optimistic locking.
In order to realize this, a field was added that represents the number of changes:
the version of the row in the table Appointments (class **Appointment**) which contains the list of cancelled users and the reserved field.
Also, added the Transactional annotation above the corresponding **scheduleAppointment()** method.
The approach of adding a column as a change counter was used, as a simpler approach and in order not to compare all the values (fields/columns) of the object with the values in the database. The disadvantage of the optimistic locking approach in this case is that it will be locked at every booking, even if they are not in overlapping terms, which further slows down the process, and there will be a longer waiting time for the user.

```java
@Table(name = "APPOINTMENTS")
public class Appointment {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "appointment_id")
    private Long id;

    @Version
    private int version;
```

```java
@Transactional(readOnly = false)
public AppointmentScheduleResponseDTO scheduleAppointment(AppointmentScheduleDTO dto, User user, String siteURL) {
    AppointmentScheduleResponseDTO appointmentScheduleResponseDTO = new AppointmentScheduleResponseDTO();
    if (user == null) {
        return null;
    } else {
        try {
            List<Appointment> appointments = getAppointmentsByCenter(dto.getMedicalCenterId());
            for (Appointment appointment : appointments) {
                if (appointment.getStartTime().equals(dto.getStartTime()) && !appointment.isReserved()) {
                    Set<User> cancelledUsers = appointment.getCancelledUsers();
                    for (User cancelUser : cancelledUsers) {
                        if (!cancelUser.getId().equals(user.getId())) {
```

●●●

```java
        } catch (OptimisticLockingFailureException e) {
            System.out.println("ERROR - OptimisticLockingFailureException");
            appointmentScheduleResponseDTO.setResponse("ERROR - Optimistic Locking Failure Exception");
            return null;
        }
    }
}
```