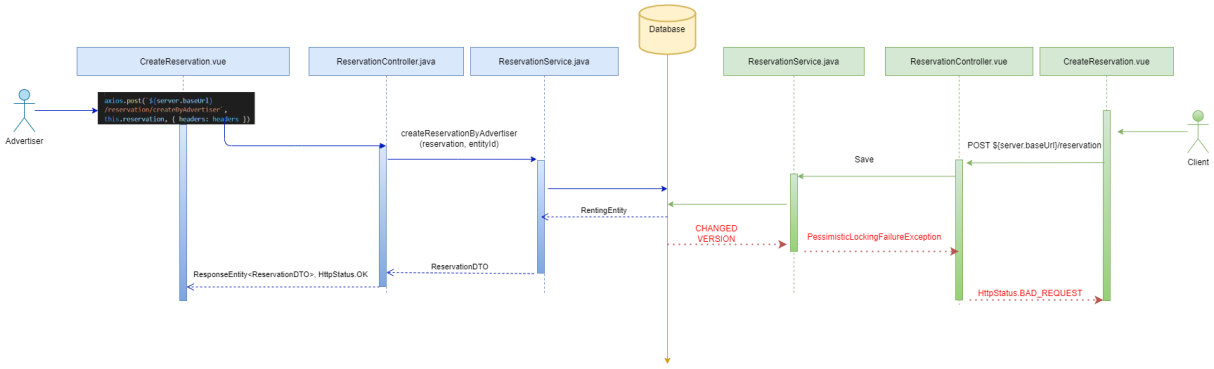
4.4 Konkurentni pristup resursima u bazi

Konfliktna situacija 1: Vlasnik vikendice/broda ili instruktor kreira rezervaciju u isto vrijeme kada i drugi klijent

Opis problema: Pored klijenta, vikendicu/brod ili avanturu može da rezerviše i vlasnik/instruktor za klijenta čija je rezervacija trenutno aktivna. Prilikom rezervacije, potebno je obezbjediti da klijent i vlasnik ne mogu u istom trenutku da izvrše rezervaciju istog entiteta, jer bi se mogla desiti situacija u kojoj se termini kreiranih rezervacija preklapaju.



Slika 1: Konfliktna situacija 1 – dijagram sekvenci

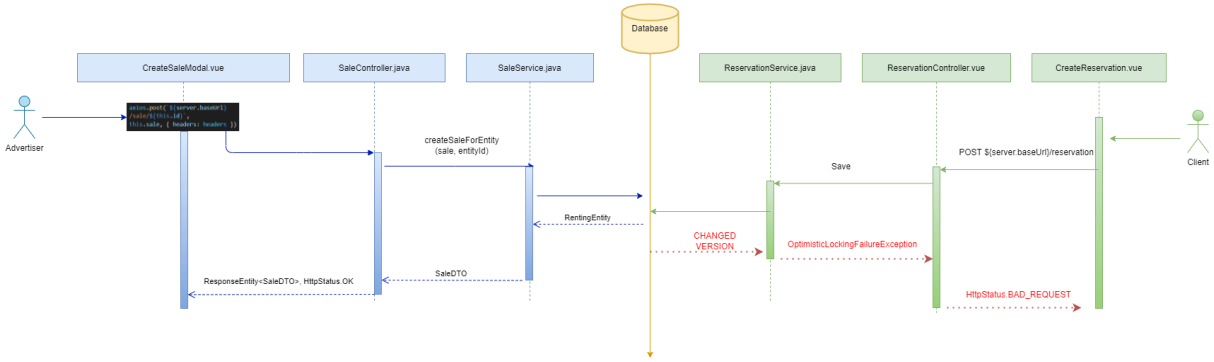
Prijedlog rješenja: Rješenje konfliktne situacije nalazi se u klasama *ReservationController, ReservationService* i Reservation*Entity*. U klasi *ReservationController*, odrađeno je rukovanje izuzetkom, kako bi klijent i vlasnik/instuktor bili obavješteni o (ne)uspješnosti kreirane rezervacije. Metode *Save(Reservation reservation)* i *saveReservationCreatedByAdvertiser (Reservation reservation, Integer entityId)* su označene kao transakcione i one pozivaju metodu *findById(Integer id)* repozitorijuma *IEntityRepository*.

Optimističkim zaključavanjem spriječili smo konfliktne scenarije - dodavanjem verzije u klasu *RentingEntity*. Na ovaj način biće poništena ona rezervacija koja pokuša da se upiše kasnije i korisnik/vlasnik koji je zakasnio će morati da ponovi postupak kako bi se podaci ispravno ažurirali.

Optimistički pristup je izabran jer ima dosta prednosti u odnosu na pesimistički. Pesimističkim zaključavanjem bismo spriječili istovremene rezervacije istog entiteta, čak i onda kada se njihovi termini ne preklapaju, što je dosta češći scenario, i time smo doprijenili boljoj upotrijebljenosti resursa.

Konfliktna situacija 2: Vlasnik vikendice/broda ili instruktor kreira akciju u isto vrijeme kada i drugi klijent vrši rezervaciju postojećeg entiteta

Opis problema: Vlasnik vikendice/broda ili instruktor ima mogućnost kreiranja brze rezervacije/akcije, koju kasnije klijent rezerviše jednim klikom. Prilikom kreiranja akcije, može se desiti u istom trenutku klijent vrši standardnu rezervaciju istog entiteta, što bi značilo da bi se mogla desiti situacija u kojoj se termini kreiranih rezervacija preklapaju.



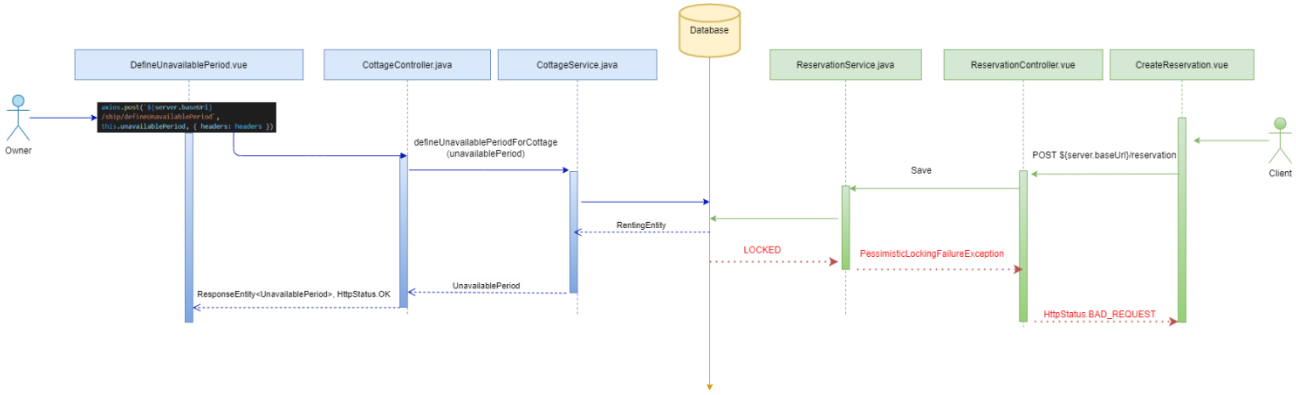
Slika 2: Konfliktna situacija 2 – dijagram sekvenci

Prijedlog rješenja: Rješenje konfliktne situacije nalazi se u klasama *ReservationController, SaleController, ReservationService, SaleService* i *EntityRepository*. U klasama *ReservationController* i *SaleController*, odrađeno je rukovanje izuzecima, kako bi klijent i vlasnik/instuktor bili obavješteni o (ne)uspješnosti kreirane rezervacije/akcije. Metode *Save(Reservation reservation)* u *ReservationService* i *createSaleForEntity(Sale sale, Integer entityId)* u *SaleService* su označene kao transakcione i one pozivaju metodu *findById(Integer id)* repozitorijuma *IEntityRepository*.

Analogno prehodnom primjeru, konfliktna situacija je riješena dodavanjem verzije u klasu *RentingEntity*. Na ovaj način biće poništena ona rezervacija/akcija koja pokuša da se upiše kasnije i korisnik/vlasnik koji je zakasnio će morati da ponovi postupak kako bi se podaci ispravno ažurirali.

Konfliktna situacija 3: Vlasnik vikendice/broda definiše period nedostupnosti vikendice/broda u isto vrijeme kada i drugi klijent vrši rezervaciju istog entiteta

Opis problema: Vlasnik vikendice/broda ima mogućnost definisanja perioda nedostupnosti tokom kojih nisu moguće rezervacije. Prilikom kreiranja ovih perioda, može se desiti da u istom trenutku klijent vrši rezervaciju istog entiteta, što bi značilo da bi se mogla desiti situacija u kojoj se termin rezervacije preklapa sa period nedostupnosti koji je upravo definisan.



Slika 3: Konfliktna situacija 3 – dijagram sekvenci

Prijedlog rješenja: Rješenje konfliktne situacije nalazi se u klasama *ReservationController, CottageController, ShipController, ReservationService, CottageService, ShipService* i *EntityRepository*. U klasama *ReservationController, CottageController* i *ShipController*, odrađeno je rukovanje izuzecima, kako bi klijent i vlasnik bili obavješteni o (ne)uspješnosti kreirane rezervacije, odnosno perioda nedostupnosti. Metode *Save(Reservation reservation)* u *ReservationService, kao i defineUnavailablePeriodForCottage(UnavailablePeriodDTO, dto)* u *CottageService,* odnosno *defineUnavailablePeriodForShip(UnavailablePeriodDTO, dto)* u *ShipService* su označene kao transakcione i one pozivaju metodu *findById(Integer id)* repozitorijuma *IEntityRepository*.

Analogno prehodnim primjerima, konfliktna situacija je riješena dodavanjem verzije u klasu *RentingEntity*. Na ovaj način biće poništena ili rezervacija ili period koji je pokušao da se upiše kasnije i korisnik/vlasnik koji je zakasnio će morati da ponovi postupak kako bi se podaci ispravno ažurirali.