Concurrency Control with Spring Framework

Thursday, July 07, 2022 9:25 AM

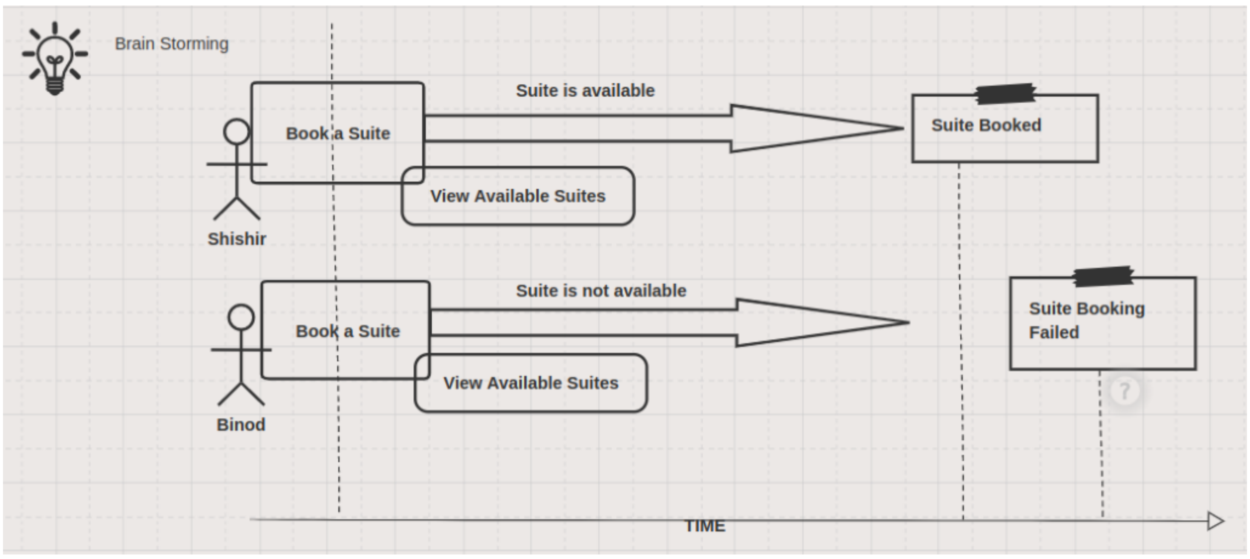
We have heard of the term concurrency many times. Currency is the fact of two or more events or circumstances happening or existing the same time. Simple real life example would be two persons, Binod and Shishir requesting for only book available in the library at the same interval of time.

In modern software system that we are building, it is not common for having thousands and even millions of users independently and concurrently interacting with our resources. Concurrency is everywhere in modern programming, whether we like it or not. We want to avoid the situations when changes made by one client is overridden by another client without even knowing. So, concurrency control comes in picture, in order to prevent these kind of situations violating our data integrity.

Now, the question comes, how do we reflect concurrency control in our API ? What error messages user's see?

To understand these, we can consider a use case; imagine a hotel named Kathmandu Marriott Hotel has only one Presidential Suite available. We have a system automating the process of booking of rooms & suites by customers. A room or suite can only be booked if it is currently available.

Brain-storming in the event



Each customers can book a room (Send request). For booking a room, customers need to see list of the available rooms or search the available rooms (Read) and make a booking request.

Understanding the scenario in simple terms

- Shishir requests Suite resource via GET endpoint
- Binod requests Suite resource via GET endpoint
- Shishir makes changes on Suite resource (I.e. Booking) and saves its changes via PUT request
- Binod makes changes on Suite resource (I.e. Booking), on the same field as Shishir, and saves its changes via PUT request.

Here, Shishir and Binod both have requested the same version of Suite resource, we now have a problem because the PUT request triggered by Binod erases the changes made my Shishir and the chances are why his booking does not show up anymore, even though he swears that he did not forget the clicking of "book the suite" button.

Locking

In a high-level perspective, there are two types of locking:
1. Pessimistic Locking
2. Optimistic Locking

Pessimistic Locking means as soon as Shishir starts making the changes on a Suit resource, he locks this resource. While the resource is locked, no one else can make the changes(edit) to this resource. Here, unless Shishir is done with his changes and unlocks the resource (Checkout/Cancel booking), Binod cannot make changes to it. Pessimistic locking ensures that 2 users cannot make changes in the same resource at the same interval of time.

Optimistic locking allows users to fetch a resource in parallel but it will not allow user to erase a previous version of a resource. In our scenario, let's say, Shishir and Binod both will retrieve the only Presential Suite resource in version 11. When Shishir books the Suite, it will send to the API the information of that the version being saved is 11.
The API will retrieve and compare the resource from database with the same version. If they differ, it will not accept the operation. Here, when Shishir saves version 11, the request will be accepted but when Binod tries to save version 11, the resource in the database will be in version 12, and the booking (edit) will not be accepted preventing Binod from overriding Shishir's booking changes.

Now, the question comes, what should the REST API implement? Pessimistic or optimistic locking?

Pessimistic locking is not impossible to perform as our REST API is stateless, but this locking mechanism would be dependent on client to unlock the resource for new clients to make edit on it. There might be the chances of resource being locked for a long time.

On the contrary, optimistic locking lets every client read and write with the restriction that just before committing the transaction we need to check whether a particular record has not been modified by someone else in the meantime (adding current version or last modification timestamp attribute).

Optimistic Lock in Data Access Layer

In Java, usually JPA is utilized for handling data access including locking capabilities. We can enable optimistic locking in JPA by declaring a version attribute in an entity and marking it with @Version annotation.

```
@Entity
@Table(name = "suite")
public class Suite{
  //...
  @Version
  private long version;
  //...
}
```

Exception that we will face is OptimisticLockingFailureException while trying to edit the same version of resource.

Bad thing about optimistic locking is that version attribute needs to be added to our domain application and API levels. This causes leak of the technical details from persistence layer. The good thing is in order to perform the update, the WHERE clause can be limited to aggregate ID and version field.

There can be some arguments in favor of using optimistic locking

- Domain is dirty, but API gets clear, concise and it's easier to use preconditions
- Version can sometimes be desired by business for auditing
- If Version is hard to be accepted, we can also use Last-Modified attribute and send in a header.