Submission 3 Report

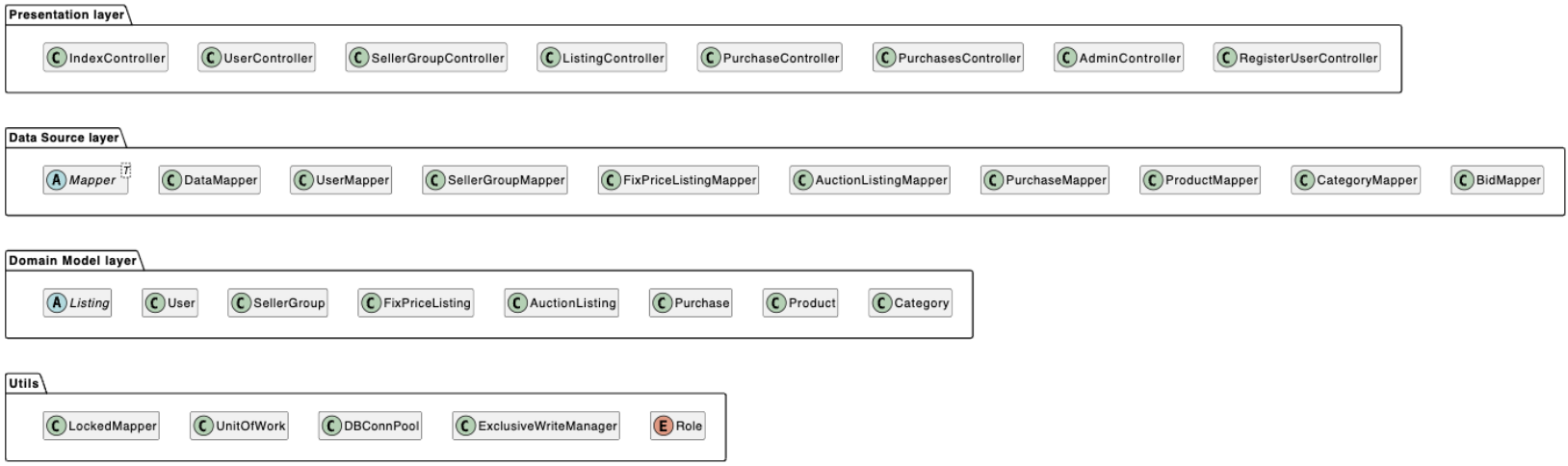
MS-Quokka

SWEN90007 SM2 2022 Project

Name	Student ID	Unimelb Username	GitHub Username	Email
Edward Bond	994614	erbond	erbond994614	erbond@student.unimelb.edu.au
Haoxin Li	911806	haoxinl1	haoxin-li	haoxinl1@student.unimelb.edu.au
Ray Chen	977720	juitengc	rayjtchen	juitengc@student.unimelb.edu.au
Yifei Yu	945753	yifeiy9	Danielyuuuuu	yifeiy9@student.unimelb.edu.au

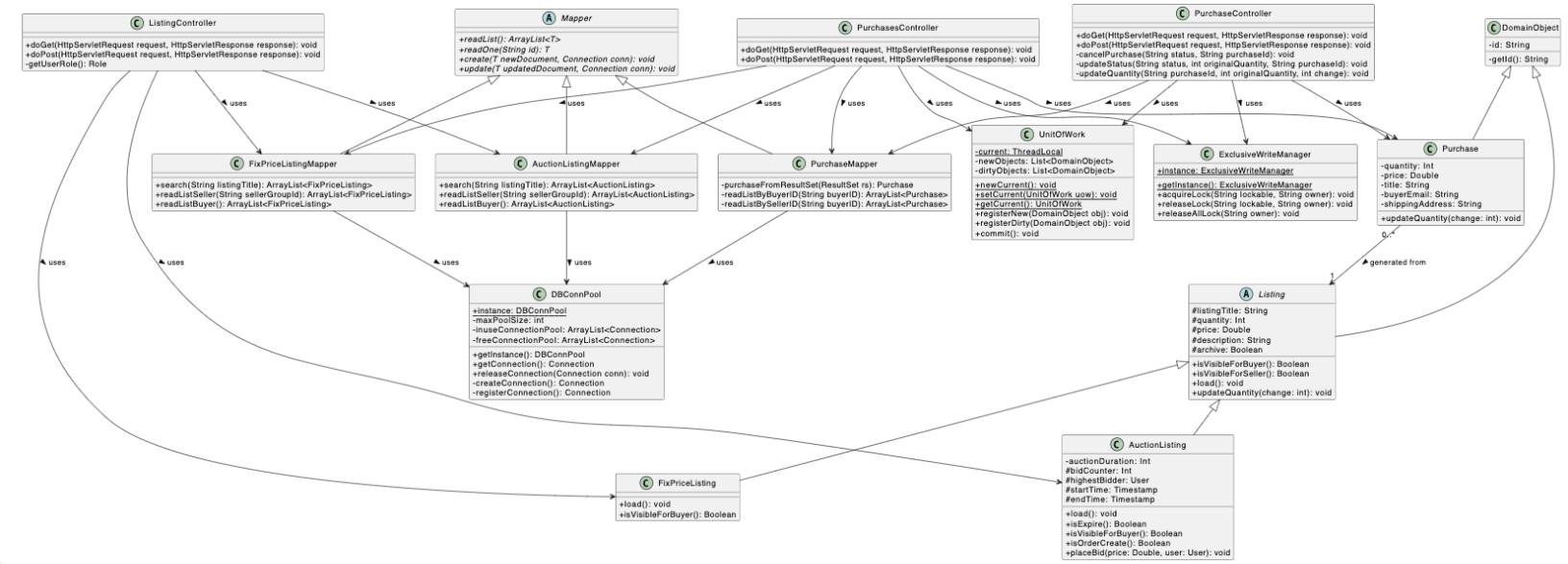
Class Diagram

Relationships between classes in different layers:



We have introduced the LockedMapper and the ExclusiveWriteManger classes.

Class diagram for the Purchase and Listing:



Concurrency Issues Overview

User case	Conflict with which use case*	Concurrency Issue ID	JMeter File Name
Seller creates a new listing	Admin removes a seller from the seller group	1	admin_remove_seller_while_seller_create_new_listing_case1.jmx
Seller views an order tied to seller group	Admin removes the seller from seller group	2	admin_remove_seller_while_seller_view_order_tied_to_seller_group_case1.jmx
Seller modifies an order (decrease quantity or cancel it)	Buyer modifies an order (increase/decrease quantity or cancel it)	3	buyer_modify_an_order_quantity_while_seller_modify_the_order_quantity_case1.jmx buyer_modify_an_order_quantity_while_seller_cancel_the_order_case2.jmx buer_cancels_the_order_while_seller_changes_the_order_status_to_processed_case3.jmx buer_cancels_the_order_while_seller_changes_the_order_status_to_fulfilled_case4.jmx
	Admin removes seller from seller group	4	admin_remove_seller_while_seller_modify_an_order_quantity_tied_to_seller_group_case1.jmx admin_remove_seller_while_seller_cancel_the_order_case1.jmx
Buyer searches for a good	None	-	-
Buyer purchases fix-price listing	Buyer purchases fix-price listing	5	two_buyer_purchasing_the_same_fixed_price_listing_case1.jmx two_buyer_purchasing_the_same_fixed_price_listing_case2.jmx
	Admin removes a listing	6	admin_remove_fixed_price_listing_while_buyer_placing_an_order_case1.jmx
Buyer places a	Buyer place bid	7	two_buyer_place_bid_on_the_same_

bid on auction listing	on auction listing		auction_listing_case1.jmx two_buyer_place_bid_on_the_same_auction_listing_case2.jmx two_buyer_place_bid_on_the_same_auction_listing_case3.jmx
	Admin removes a listing	8	admin_remove_auction_listing_while_buyer_placing_a_bid_case1.jmx
Buyer views an order	None	-	-
Buyer modifies an order (increase/decrease quantity or cancel it)	Seller modifies an order (decrease quantity or cancel it)	**The same as concurrency issue 3**	-
Seller/Buyer update their user profile	Admin removes seller from a seller group	9	admin_remove_seller_while_seller_changing_user_profile_case1.jmx
	Admin onboard sellers (i.e. add seller to a seller group)	10	admin_add_buyer_to_seller_group_while_buyer_changing_user_profile_case1.jmx
Admin onboards sellers (i.e. add seller to a seller group)	Buyer updates their user profile	**The same as concurrency issue 9**	-
Admin views all purchases	None	-	-
Admin creates a seller group	None	-	-
Admin removes seller from a seller group	Seller update their user profile	**The same as concurrency issue 10**	-
Admin removes a listing	Buyer purchases fix-price listing OR places a bid (depends on the listing type)	**The same as concurrency issue 6**	-

*** Note:** It means that when these two use cases happen at the same time, concurrency issues can occur. Description of the concurrency issue will be introduced in later sections.

Design Overview

Our design systematically applied Exclusive Write lock (one of the Pessimistic offline locks) to most of the concurrency issues that our system potentially could encounter. This decision was made after analysing our use cases. In each of the concurrency issues below, we will provide:

1. A detailed description of the issue
2. Design rationale behind the choice of Exclusive Write lock
3. Implementation details
4. How the test was conducted, and the outcome

Note that we have chosen not to explain the concurrency issues in numeric order, but in an order that makes the most logical sense. All 10 issues will be addressed.

Concurrency Issue 5 & 7

Issue 5

Issue 5 happens when two buyers try to make a purchase from the same fixed-price listing. This is an example of a potential Lost Update where two processes attempt to write data to the same location in the database simultaneously. For this issue, both buyers attempt to update the same record in the fixed_price_listing table by decreasing the value of the quantity field (i.e. available quantities left). Simultaneously creating a new record of purchase in the purchase table should be handled by the database and therefore is not of concern here.

For example, Buyer 1 and Buyer 2 each read the same listing row from the database with an available quantity of 10, and each of them makes a change to their local copy of the listing object in memory, and attempts to commit the change to the database. Buyer 1 would like to purchase 5 items, while Buyer 2 would like to purchase 2 items. The correct state of the database after these two simultaneous updates should be a quantity of 3, however if Buyer 1's request was finished after Buyer 2's request, then the listing will have an incorrect quantity of 5 as the first update will be lost (or 8 if it's the reversed case).

To handle this concurrency issue, we used Exclusive Write lock (one of the Pessimistic offline locks) after comparing each of the alternatives. We will discuss the implementation details, then explain our design rationale.

To implement the Exclusive Write lock, the first step is to implement a lock manager and the implementation detail is shown below in the sequence diagram.

The lock manager is a singleton Java class that uses a concurrentHashMap to keep track of which threads are locking which object item(s). The locking mechanism simply maps locks to owners where the locks are the objects ID in the database, while the owners are represented by the thread name. Please note that we used globally-unique primary keys over the entire database; this eliminates the need to keep track of the table on which the lock was acquired for. In the above sequence diagram, a thread will pass in its name and the object ID for acquiring the lock. If the object ID already exists in the lockMap, the check will fail and put the thread into "wait" until the lock can be acquired. Once the check has succeeded, a new record will be inserted into the lockMap with the object ID as the key and the Thread name as the value. After the thread has completed the request, it will call the releaseLock method to remove the record from the lockMap and call the notifyAll function to wake up all the currently waiting threads that may be waiting for the lock it has just released.

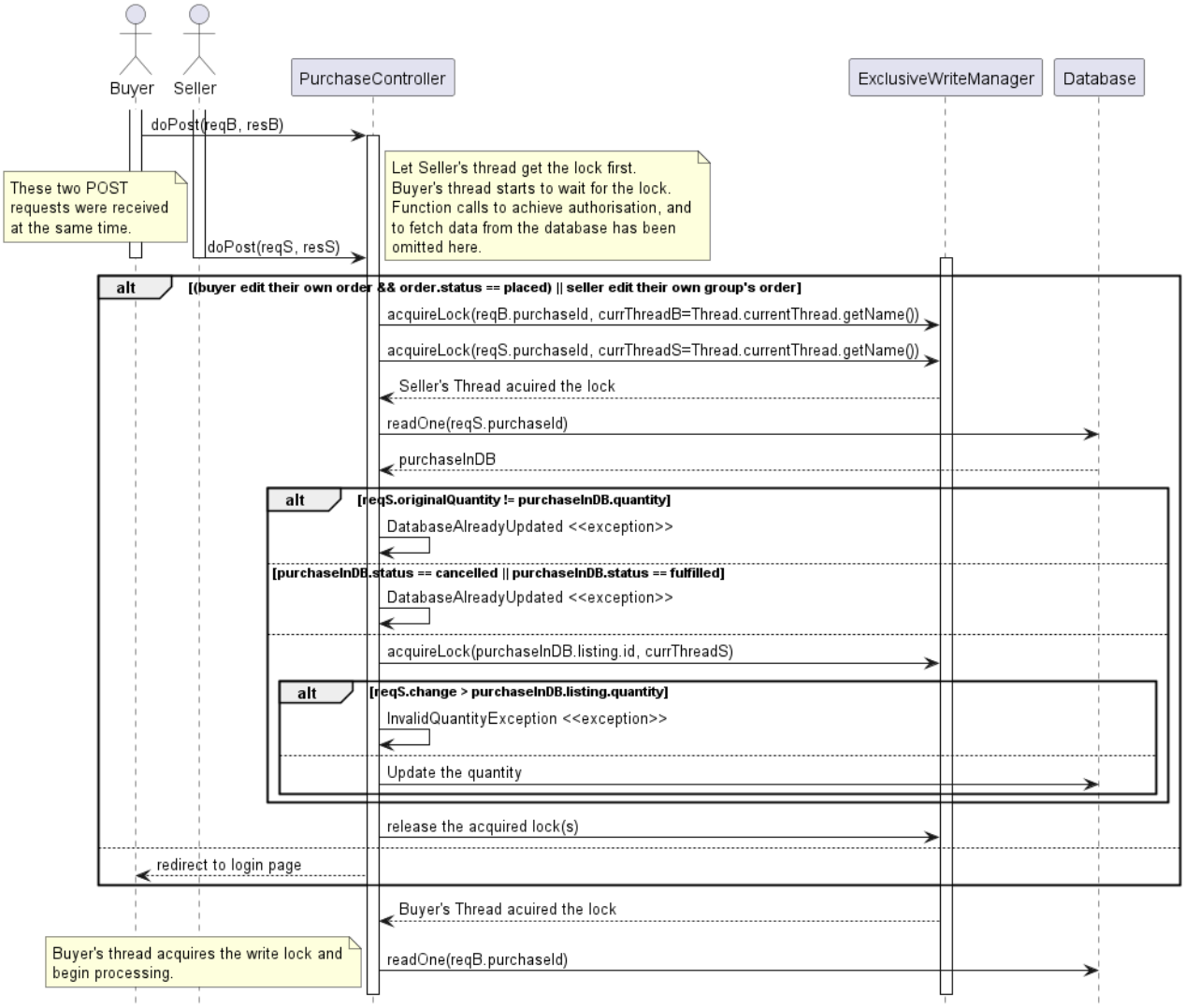
After implementing the lock manager, we also need to define the lock protocol. With an Exclusive Write lock, we have decided to:

1. Acquire the lock at the start of a business transaction for update requests only. Note that the request needs to pass authorisation checks to make sure the user is authorised to

modify the resource first, before acquiring the lock. The write lock will be acquired first, before the object is read from the database. This way we can (1) make sure we have the most up-to-date version of the record, and no one else can edit it while we have the lock, and (2) if others have updated the record before we acquired the lock, checks that the request is still legal to be made.

2. The lock is based on the primary key of the object's record in the database. In part 2, we have implemented primary keys that are globally unique, ensuring that all the keys in the lockmap are unique. This is crucial as we want to have precise control over which lock we choose to acquire and release. The values of each key-value pair in the lockmap will be threads' ID.
3. After the business transaction has been terminated (either successfully or by throwing exceptions) the lock will be released.
4. If the lock cannot be acquired, the thread requesting that lock will simply wait until it can be acquired (as shown in the sequence diagram above).

The following sequence diagram illustrates what happens when two buyers want to make a purchase of the same fixed-price listing at the same time. BuyerA and BuyerB both send a POST request to the PuchasesController, and let us assume that both POST requests were received at the same time. BuyerA's request is carried in threadA, and BuyerB's request is carried in threadB.



Both threads will try to acquire the write lock for the same listing (i.e. reqA.id and reqB.id have the same value). Let us assume that threadA got the lock first. ThreadB will have to wait until that lock is released by threadA.

After acquiring the lock, threadA will load the listing object from the database. Note that this function call to the database is already implemented in our part 2 solution. The reason behind this is that we only ask the frontend to send the necessary fields of an object to the backend for an update, rather than the entire object. In this example, the frontend only sends the type of the listing, the listing ID and the quantity requested by the user. Therefore, in order to construct the listing object in memory, the controller needs to fetch the record in the database.

Our solution for part 3 utilises this readOne function call to our advantage. After acquiring the write lock, the readOne function will be executed and we can be certain that the listing instance it returns is the most up-to-date record, as no other threads are able to modify this record in the database after threadA has acquired the write lock.

The next step is to check that from the time when BuyerA loads the listing object to read at the frontend, to the time when threadA successfully acquires the write lock, the database record has not been modified by other threads in a way such that the POST request is no longer legal. In this example, it means that we need to check if this listing has not been archived, or it still has enough stock left to fulfil threadA's request. As an example, firstly, UserA will load the listing object from the database to read on the webpage, and at this point it has a quantity of 10 remaining. Then, after 5 minutes, UserA decides to buy 5 items in this listing. However, during that 5 minutes, UserC has already bought 8 of them. This means that the second check (i.e. listing.getQuantity() < reqA.quantity) will fail, and UserA's request won't be processed.

In the case where one of the checks fails, a DatabaseAlreadyUpdated exception will be thrown, and UserA will be redirected to an error page with an appropriate error message. If both checks are passed, then threadA can safely proceed to create the purchase (by using Unit Of Work which was implemented in part 2). The write lock of this particular listing object will always be released at the end (in a "finally" block), regardless if an exception is thrown or not.

After threadA has released the write lock on this listing object, threadB can obtain the lock and continue to process.

After explaining the implementation details, let us discuss the design rationale behind the choice of Exclusive Write lock. In the following sections we will compare the other alternatives:

Alternative 1: Optimistic offline lock

This was a rather attractive choice due to its simplicity. If we implement optimistic lock, all we need to do is add an extra "version" field to each database table, check and update the versions for each POST request. We do not need to implement a lock manager or define a lock protocol.

However, we realised that the frequency of conflict will be high for this use case. For an enterprise application, it is very common for multiple users to purchase the same item at the same time. Imagine the scenario where 11 users try to purchase the same fixed-price item, with the assumption that there is enough stock for all of their requests. Using optimistic lock would mean that:

1. 10 users each read the same record, and the version is stored locally (e.g. version == 1)
2. The 11th user updated that record in the database, now version == 2
3. The 10 users, before updating that record in the database, read the version number in the database again. They would discover that the version number is no longer 1.
4. An error will be thrown to let the 10 users know that someone else has updated the database, therefore please try again later.

The worst case scenario is that one of the users will have to send the purchase request 11 times to successfully purchase. On average, there will be many retries necessary for a user to successfully purchase an item. We believe that this poor user experience is unrealistic as it will deter users from using the application to make purchases.

On the other hand, the lock manager implemented with the Pessimistic offline lock pattern will ask the incoming threads to wait. When the controller receives a post request, we lock the listing object, and read the object from the database and we can ensure this is the most up-to-date version. No one else can edit this record. Then, we check if there's enough stock left. If there are, a purchase will be made and the lock to the listing record is released. Then, the next thread can acquire the lock to perform the updates. Provided that there is enough stock left, all 11 users only have to click the 'Purchase' button once, resulting in a much improved user experience.

Alternative 2: Pessimistic offline lock - Exclusive Read lock.

This approach provides the advantage where a user will always be able to see the accurate value of quantity remaining for a listing, as a lock is required whenever data is loaded in memory. Using the same scenario where 11 buyers try to purchase the same fixed-price item, if one user has loaded the listing record in memory, all other users cannot read or update that record in the database. This would ensure that the user's local version is up-to-date with the database.

However, this comes at a cost - all other users are not allowed to browse the listing any more. An enterprise system with thousands of users will very often have multiple users browsing the same listing. It will result in a very poor user experience if a customer has to wait for a long time to even just browse an item on sale.

On the other hand, Exclusive Write lock only acquires a lock when an update to the database is requested. As a result, while one user is making the purchase (therefore updating the listing record in the database), all other users can still view this listing.

For this use case, liveness is more important than ensuring users have the most up-to-date information. Unlike a bank system where it is crucial for the user to accurately read how much money they have left in their account, it is not crucial for the buyer to know exactly how much stock is left available to purchase.

Alternative 3: Pessimistic offline lock - Read/write lock

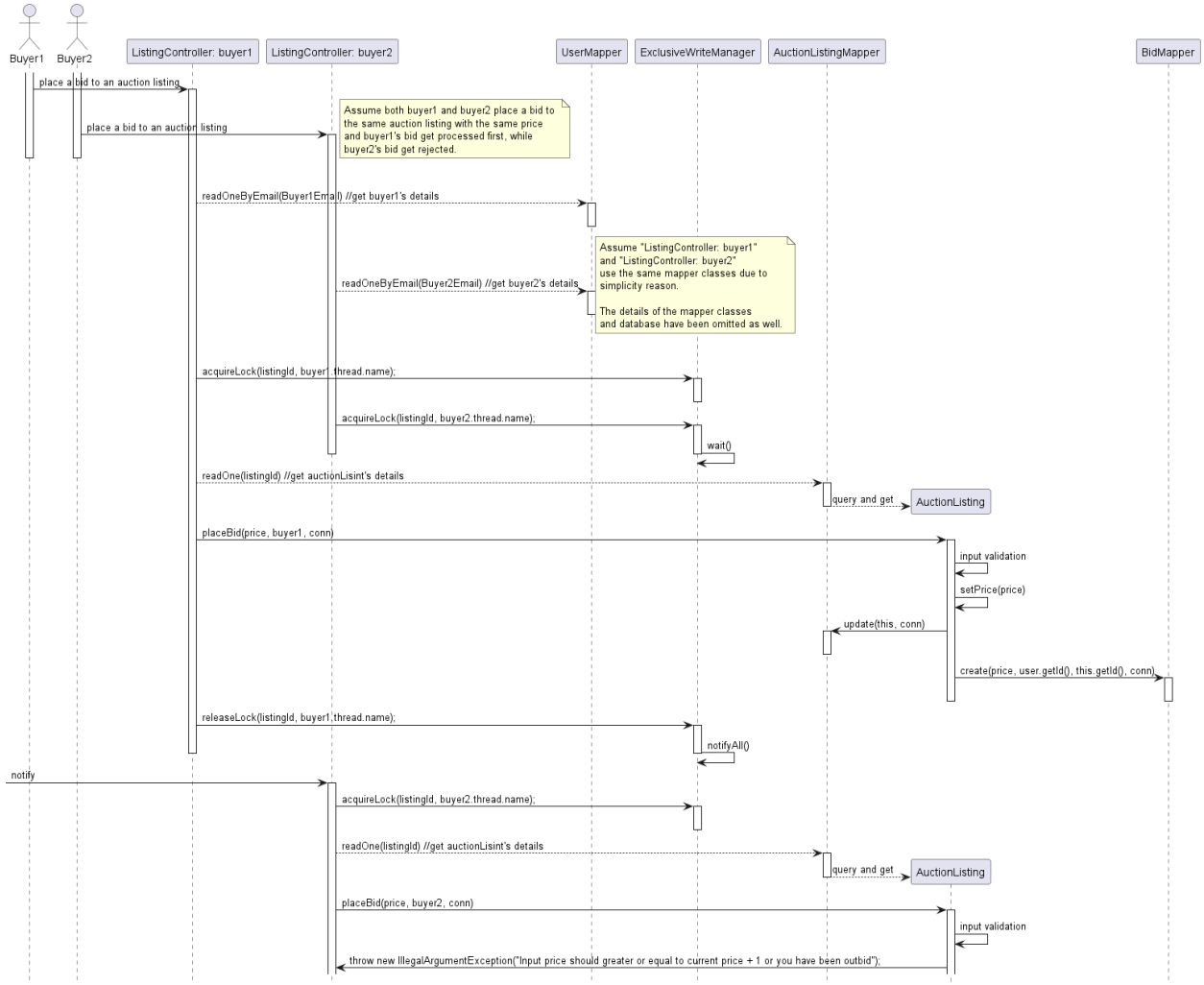
This approach requires that the read and write locks are mutually exclusive - in other words, if a user is making a purchase (and therefore updating the listing record in the database), no other users can read that particular listing object. This leads to a very similar experience for users as discussed in the Exclusive Read lock. Therefore we decided not to use this approach either.

Issue 7

Issue 7 describes the scenario where multiple users attempt to place a bid for an auction listing. This is similar to Issue 5, where each request attempts to update the same record, in this case

in the auction_listing table by increasing the price attribute (i.e. the current highest bid price of the listing). Similar to Issue 5, simultaneously creating a new record in the bid table should be handled by the database therefore it will not be discussed here.

Issue 7 is handled by implementing the Exclusive Write type of Pessimistic offline lock. The design rationale behind this choice is very similar to Issue 5 therefore we will omit it here.



Concurrency Issue 3

Issue 3 refers to the scenario where two users attempt to update a purchase, either by updating its status or updating the quantity. Since there are many different combinations of updates that could happen, each case will be discussed separately.

There are three considerations that apply to all cases that we would like to discuss first.

Firstly, where possible, we would like to re-use any existing pattern that we have implemented in our system to reduce the complexity of our design, provided that the existing solution will still work. Adding a new pattern while the existing solution works is unnecessary, making the system harder to maintain.

Secondly, we would like the user to always be able to view an order as we believe it would enhance user experience. This decision of ours suggests that we should steer away from using Exclusive Read lock or Read/Write lock for this concurrency issue.

Thirdly, since an order is an agreement established between buyer and a seller, we must make sure that each update request is legal to be made, and the resulting data is consistent after concurrent requests.

We have found a solution that satisfies all of the considerations above: it uses the Exclusive Write lock (i.e. maximising re-use), allows the user to view an order at any time (i.e. better user experience) and finally, it ensures that the data is consistent after concurrent requests. We will explain our solution in each subsection below.

Issue 3a: Both users try to update the status

Issue 3a happens when a buyer tries to cancel an order, while a seller tries to cancel/process/fulfil the same order. It can also happen when two sellers (from the same seller group) try to update the status. Both POST requests attempt to update the status field of the same record in the purchase table.

As an example, let us assume that a buyer would like to cancel a purchase that currently has the status ‘placed’ (‘placed’ means that order has been placed by a buyer but the seller has not started processing it yet). Meanwhile, the seller wants to update the same order’s status to “processed”, indicating that the seller accepts the order and will make preparations.

Let us assume that the Buyer’s thread was able to acquire the write lock on the purchase that both threads are attempting to update. The seller’s thread will wait for the lock to be released.

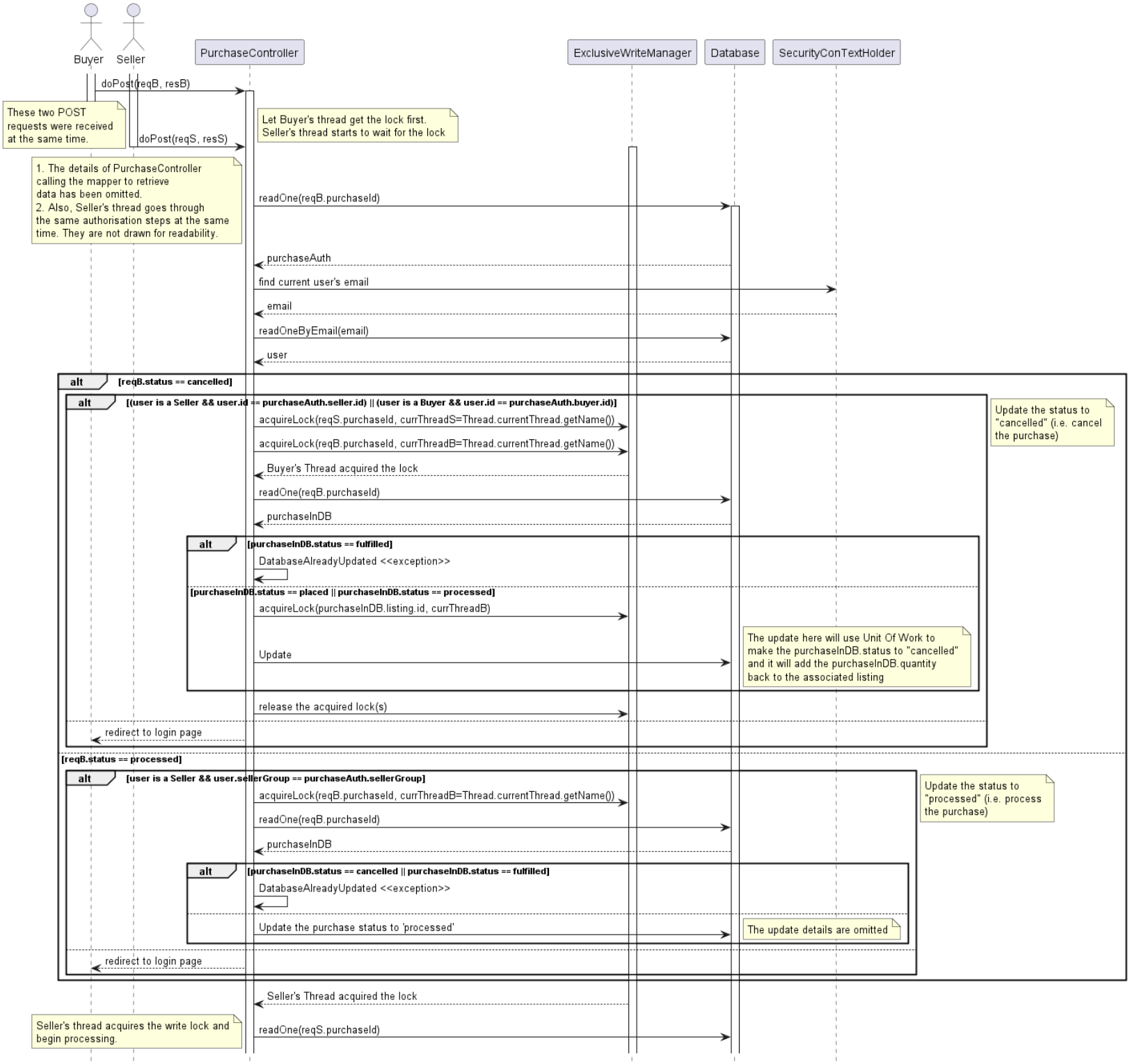
As shown in the sequence diagram below, after receiving a POST request, the purchase object was read from the database (i.e. purchaseAuth). Note that this is an additional step created solely for the purpose of authorisation check. We need to make sure that the seller or the buyer

makes requests for a purchase that they have access to (i.e. the buyer updates their own purchases, and the seller updates their own seller group's purchases).

Once the request has been authorised (i.e. [(user is a Seller && user.id == purchaseAuth.seller.id) || (user is a Buyer && user.id == purchaseAuth.buyer.id)] evaluates to true), the business transaction begins.The first step of the transaction is to acquire an Exclusive Write lock for the purchase object. This ensures that from this point onwards, no other threads can modify this purchase object.

The second step of the business transaction is to read the purchase object from the database (i.e. purchaseDB). This is a step already implemented in part 2. Note that we cannot reuse the purchase object (i.e. purchaseAuth) that we have obtained during the authorisation check, in case any updates have happened to this record in the database between authorisation check and obtaining the lock. We must read from the database again *after* the write lock has been acquired to ensure what we have loaded in memory is the most updated version.

After acquiring the write lock and reading again from the database, Buyer's thread can safely update the status from 'placed' to 'cancelled'. Note that for cancelling an order, the associated listing object needs to be updated therefore a second lock will be acquired for the listing object.



The write lock is then released and acquired by the Seller's thread which has been waiting for this lock. It reads again from the database. We have implemented guards at this point to ensure that the status has not been updated to 'cancelled' or 'fulfilled'. In this case, since the status has

already been updated to ‘cancelled’ (by the Buyer), a DatabaseAlreadyUpdated error will be thrown, letting the Seller know that it cannot update the status to ‘processed’ because the Buyer has already cancelled the order.

In this example, Exclusive Write lock alone has ensured that the status update is always consistent at the end, and sends users an alert if the data has been updated by others. It also provides a good user experience as reading the order is always allowed.

Issue 3b: Both users try to update the quantity of the order

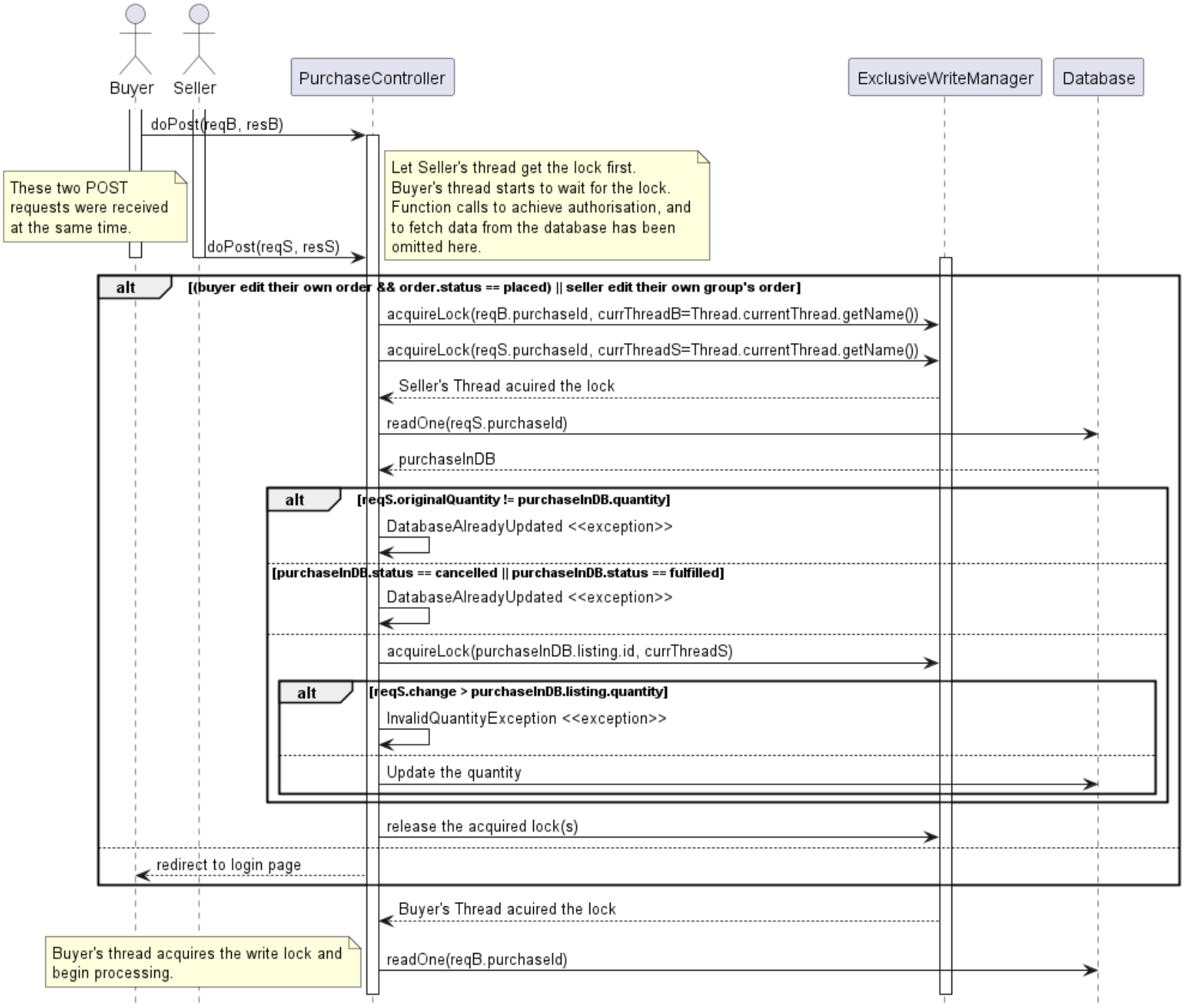
Updating the quantity is a more challenging scenario than status update. The key is to have a guard to check whether someone else has made an update to the quantity in the database. To do this, we adjusted the frontend so that it sends the original quantity that the user reads on the user interface before making any request to update it. We will label this as the *original quantity*. The changes that the user would like to make to the order is the *change*.

As an example:

1. The buyer previously made a purchase of 5 items. The original quantity for buyer == 5
2. The seller realises that they only have 3 left (due to logistic reasons unrelated to our website), and typed “-2” in our system to reduce the order quantity by 2. The original quantity == 5 and change == -2 for the seller.
3. Meanwhile, the buyer wants to add 3 more items to the order, therefore change == 3 for the buyer.

After the authorisation check, the Seller’s request is processed first so that the quantity in the database is updated from 5 to 3. When the Seller’s thread releases all the acquired locks, Buyer’s thread finishes waiting and gets the write lock for the purchase record. Then, it reads from the database to get the most up-to-date purchase record. Next, as shown in the sequence diagram below, there is a guard that checks whether the purchase’s quantity stored in the database (i.e. purchaseInDB.quantity) is still the same as the original quantity sent from the request (i.e. reqS.originalQuantity). This is a really important step. The original quantity belongs to the Buyer’s local purchase object. It means that when the Buyer loads the page to read their purchase, they have a local copy of the purchase object, and this object’s quantity is the original quantity. If this quantity is different to the one stored in the database, then it means that someone else has updated the quantity since the Buyer read it from the database and an error will be thrown to ask them to retry. If they are the same, then it means that no one else has updated the quantity therefore they can safely update the quantity (in this example, this happened for the Seller).

Using Exclusive Write lock with the guards mentioned above, we ensure that the order quantity is accurate when the users read the data on the webpage, and if other users have updated it before the request was sent an alert will be shown.



Issue 3c: One user tries to update the quantity, while the other user tries to update the status

We will not discuss Issue 3c in detail as it is similar to Issue 3a and 3b. The main differences are the guards that are in place before the second request can perform the update:

- Before updating the quantity, a guard will check if the order has already been fulfilled or cancelled. When the status is equal to either of these two values, the quantity cannot be updated again.
- Before updating the status to ‘processed’ or ‘fulfilled’, a guard will check if the order’s quantity has been updated. If that is the case, the seller needs to refresh the page and load the new quantity again.
- Before updating the status to ‘processed’, a guard will check if the order has been cancelled or fulfilled.
- Before updating the status to ‘cancelled’, we will check whether the order has already been fulfilled (and vice versa)

At the end of our discussion for Issue 3, we would like to point out the main disadvantage of our approach - there are a large number of checks that we need to perform to make sure that the second update is still legal to perform. Implementing the checks is error-prone as the business logic can be rather complicated. If we adopted the Pessimistic offline locks that include a read lock, or the Optimistic offline lock, we won’t need to implement these checks ourselves, however we will sacrifice user experience to a certain extent and we will not be able to maximise the code reuse. In this instance we have chosen user experience and reuse over the complexity of implementing multiple guards.

Deadlocks

There are a couple scenarios in our system where multiple locks are required. For example, when a buyer/seller updates an order quantity or cancels an order, the order object as well as the associated listing object needs to be updated. As shown above in the sequence diagrams, a lock is acquired for the order object as well as the listing object.

Deadlock can happen when one user acquires the lock for the order object, waiting for the lock of the listing object to be released, while the other user acquires the lock for the listing object and desires the lock of the order object. This would result in both threads being starved waiting to acquire the other locked data. We have avoided the situation by making sure that each thread will acquire the lock for different domain classes in the same order. In other words, a lock for the order object will always be acquired first, then the listing object’s lock will be acquired. It means that the second request must wait for the first request to finish before it can obtain its first lock.

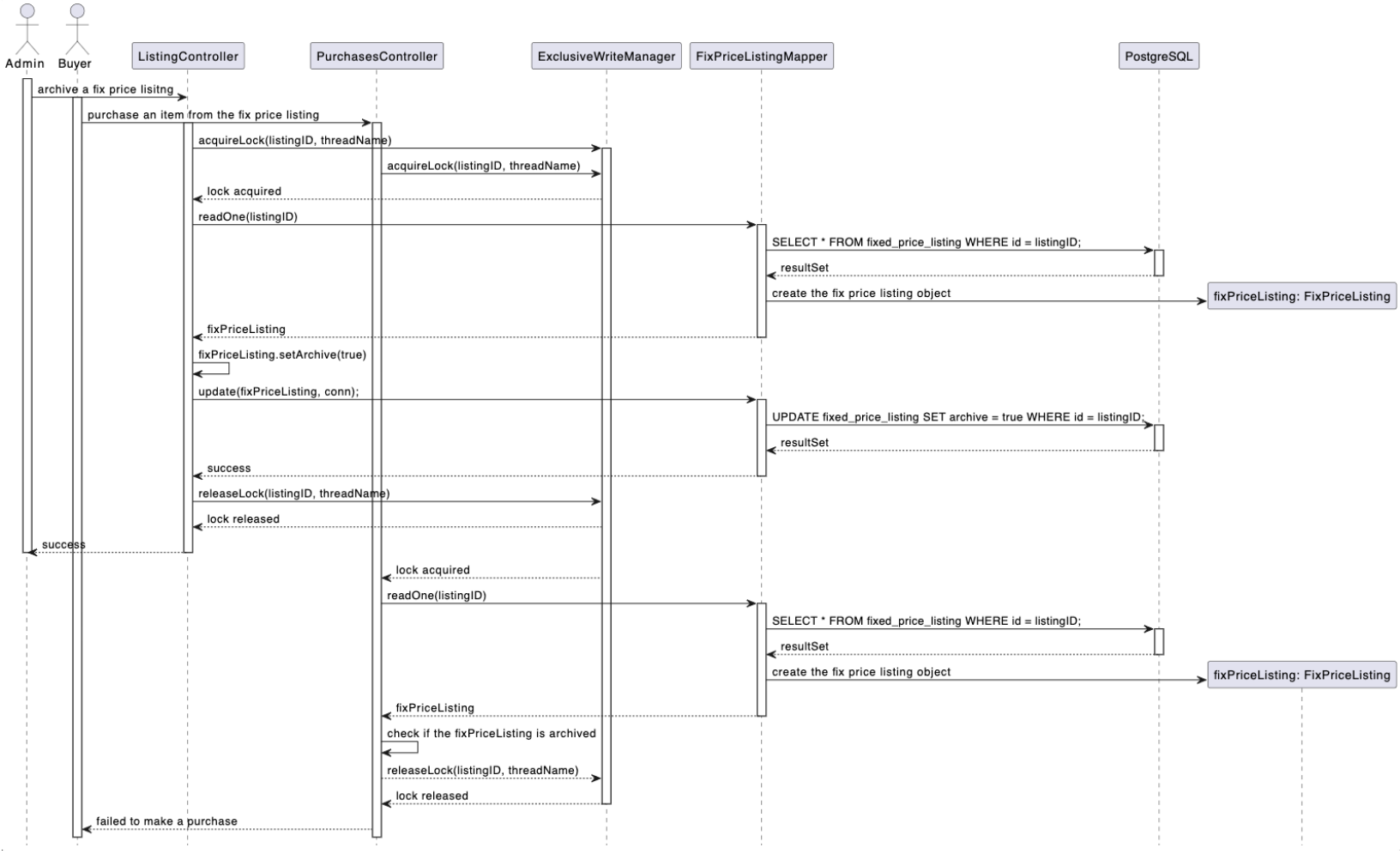
Concurrency Issue 6 & 8

Issues 6 and 8 are rather similar therefore they will be discussed together.

Issue 6 and 8 refer to the scenario where the admin has removed a fixed-price/auction listing while a buyer tries to make a purchase/place a bid on it. The outcome of the two updates depends on which request is processed first by the server.

We will use fixed-price listing as an example. If the buyer’s request is processed first, followed by the request of the admin, then the buyer's order will remain in the system. This result satisfies the business logic such that before the admin archives a listing, a buyer is allowed to make a purchase out of it.

If the admin’s request is processed first, followed by the request of the buyer to make a purchase, a guard will check just before the user makes a purchase to ensure that the listing is not archived yet. As shown in the sequence diagram below, at the start of the business transaction, the Buyer's thread will acquire the lock after the Admin’s thread has released it (note that no authorisation check is needed here for making a purchase). Then the second step is to read the listing object from the database (which we already implemented in part 2), and if the listing object’s ‘archived’ field is set to true, it means that the Admin has already cancelled the listing since the Buyer first loads the listing object into the frontend. An error will be thrown, letting the Buyer know that the Admin has already archived the listing.



Similar to Issue 5 & 7, regarding listing, we want to maximise the user experience by not requiring any lock to read data as well as not making the buyer retry too many times. Therefore Exclusive Read lock, Read/Write lock as well as Optimistic offline locks are not chosen (see Issue 5 & 7 for detailed rationale).

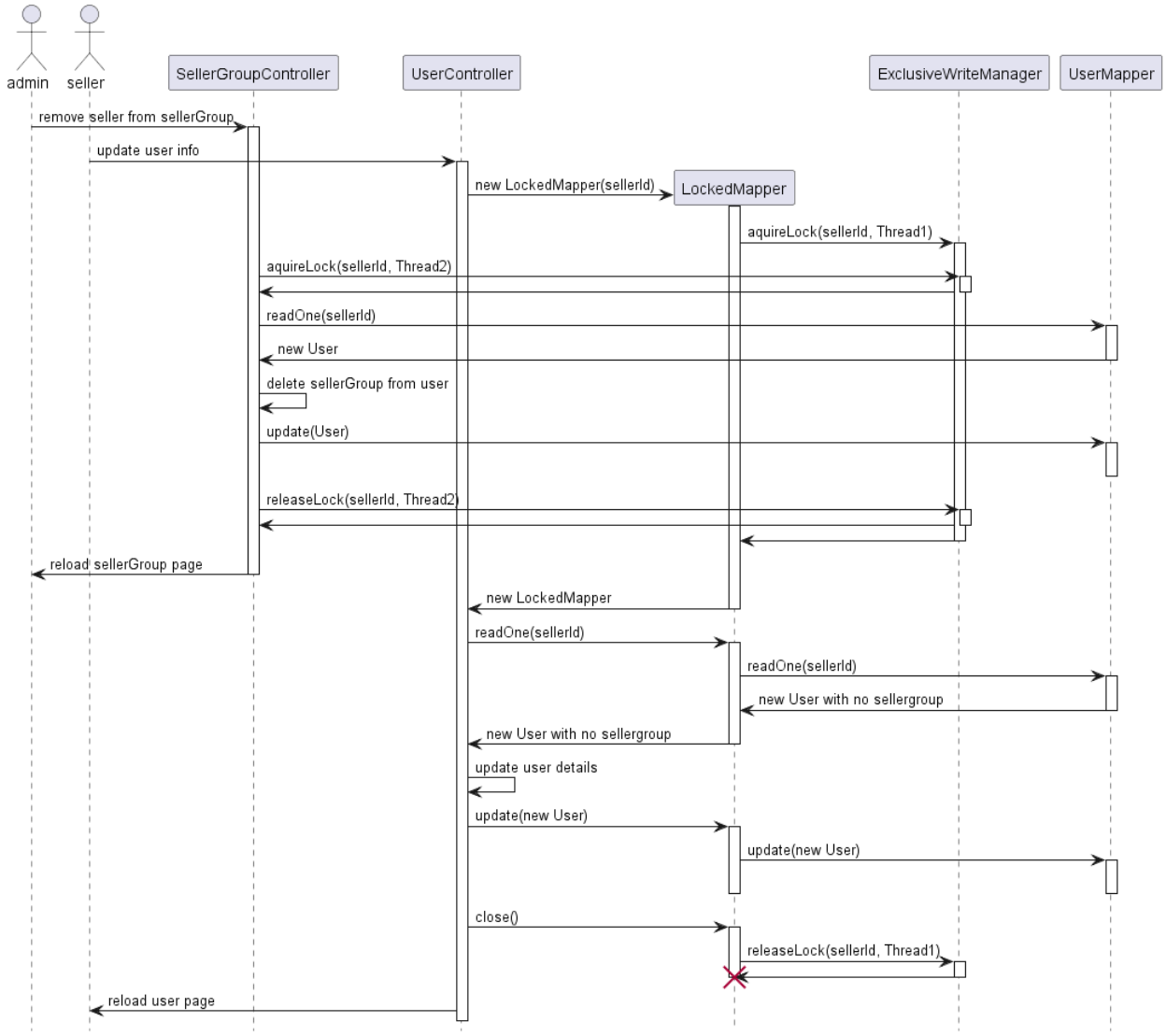
Concurrency Issue 9 & 10

Issue 9 & 10 are similar therefore they will be discussed together.

Issue 9 refers to the scenario when a seller updates their user profile while the admin removes them from the seller group. Issue 10 is the equivalent case for a buyer - when a buyer attempts to update their user profile, the admin onboard them to a seller group.

This is once again a classic example of the Lost Update issue which could happen when both threads attempt to modify the same record in the database. As we have consistently applied Exclusive Write lock across many use cases, we believe it is the best option to re-use the same pattern to avoid unnecessary complexity, provided that it works.

The sequence diagram below illustrates how the Exclusive Write lock works for Issue 9:



The Exclusive Write lock allows us to ensure that no updates are lost by guaranteeing that no updates will be made between the read call and the update call. Because of this, in interlocking requests as shown above, the value read from the database will have all prior updates applied even if the request was made later but it acquired the lock earlier. This is possible because the update being made is not reliant on the information that was displayed to the user being 100% up to date. The user will still want their details updated whether they are a seller or not.

We also implemented a prototype of an implicit lock in the userController in the form of the LockedMapper. The implicit lock is able to take the locking logic and timing out of the controller and ensures that the lock will always be released once it is no longer needed. Because locking logic is removed from the controller layer and done at one place, it reduces the risk for a developer to forget to acquire a lock in the controller, leading to possible overwrites, or forget to release a lock in the controller, leading to liveness issues. It also means that it is easier to change locking strategies should the need arise.

Concurrency Issue 1, 2, 4

We will discuss the three issues together as they are very similar. All three issues involve a seller performing a seller-only action while the admin removes the seller from the seller group simultaneously.

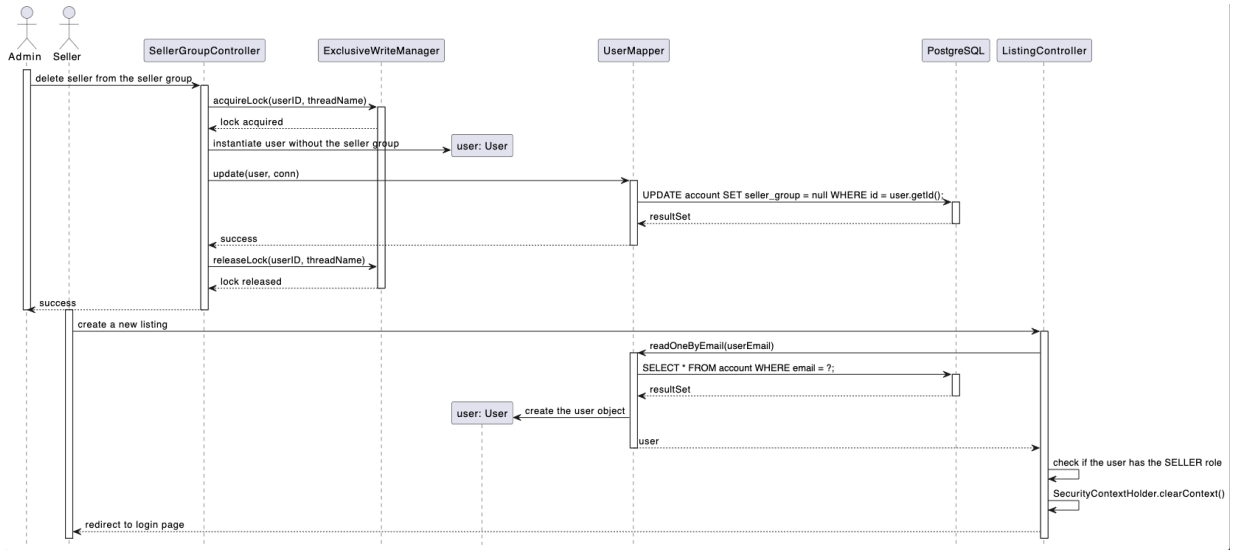
Issue 1

Issue 1 refers to the scenario where a seller tries to create a new listing, while at the same time the admin removes the seller from the seller group. This issue is interesting as it does not fall under Lost Update or Inconsistent Read. A Lost Update, by definition, happens only when the same location in the database is updated. In this case, the updates are made to different locations in the database - the user table and the listing table. Inconsistent Read involves one process reading multiple locations in the database while another process updating them, which does not apply to this scenario either.

What we need to manage in this scenario is an authorisation issue that arises from simultaneous updates. This was not an issue In part 2, as we are allowed to assume that only one user will use the system at a time. If the admin removes a seller from a seller group first, when the removed seller logs in later (i.e. serial usage of the system), they will only be able to see the user interface of a buyer. In other words, after the admin has removed a seller from a seller group, there is no possibility for the removed seller to send authorised requests as a seller from the frontend.

In part 3 however, performing a seller-only action and removal of the seller from the seller group could happen simultaneously. It means that we need to have authorisation checks in the controller to make sure that incoming requests are indeed authorised.

We have decided to add the authorisation checks in the Controller layer, as it is the Controllers' responsibility to handle incoming requests. Using other layers to handle this (e.g. the domain/mapper layer) does not make sense as an unauthorised error should be thrown as early as possible in this case. For issue 1, we added an authorisation check in the ListingController. When it receives a request to create a listing, it will perform a check in the database to make sure that the user still has the SELLER role, i.e. the admin has not removed them from the Seller Group. If the user is still a seller, then the ListingController will proceed to handle the request, otherwise, the user will be logged out and be redirected to the login page. This is evidenced in the sequence diagram below, where the admin has removed the seller from the seller group, but at the time that the seller makes a 'create a new listing' request, that seller no longer has the SELLER role, and it would fail to execute the request and will be logged out immediately.



The additional authorisation checks will introduce some performance issues as one additional database request is made for every request made by a seller. However we believe that it is more important to ensure the application performs correctly and securely, before we attempt to improve the performance of the application.

Issue 2 & 4

Issues 2 and 4 are very similar to Issue 1. Issue 2 refers to the scenario where a seller attempts to view an order tied to the seller group while the admin removes the seller from the group. Issue 2 refers to the case where a seller attempts to modify an order (i.e. decrease quantity or cancel it) while they are removed from the group by the admin. For these two cases, we have implemented the authorisation checks described above in the PurchaseController (to view/update/cancel a single order tied to a seller group) and PurchasesController (to view orders tied to a seller group), so that only the user that has the SELLER role will be allowed to perform these request, otherwise the user will be logged out immediately.

Testing

Testing strategy Overview

Our general testing strategy is to let two or more users, simulated by JMeter, simultaneously perform a task that will test one of the identified concurrency issues. Under each concurrency issue, how the test was carried out and the test result will be discussed.

Concurrency Issue 1

The concurrency issue is referring to a scenario where a seller tries to create a new listing, while admin tries to remove the seller from a seller group

Testing Approach

1	Admin removes a seller from the seller group while the seller creates a new listing.

Expected Result

Sub test cases	Expected Result
1	Two possible results may occur: ● Admin removed a seller from the seller group and the seller failed to create a new listing. ● Seller created a new listing and the admin removed the seller from the seller group.

Testing configuration

Sub test cases	Configuration
1	Seller: ● id = 5dcd2440-877f-47f9-b9a1-b708a5bd9bc2 ● Seller Group id = 593aab5d-96f6-459a-82ee-d5c83bc55309 ● Create a new listing Admin: ● Remove seller ‘5dcd2440-877f-47f9-b9a1-b708a5bd9bc2’ from seller group ‘593aab5d-96f6-459a-82ee-d5c83bc55309’.

Actual Result

Sub-test cases	Test Result	Status
1	Both requests got accepted. The seller has successfully created a new listing and the seller was removed from the seller group.	Pass

Fixed Price Listing table before the test:

	id [PK] character varying (36)	listing_title character varying (50)	quantity integer	price real	description character varying (200)	seller_group character varying (36)	product character varying (36)	archive boolean
1	0e923f33-13cd-4189-afa8...	iPhone 14 Pro Max 25...	6	2099	Brand new iPhone 14 P...	593aab5d-96f6-459a-...	7fbd1bdb-5573-4975-...	false
2	16f01056-7382-4c79-9e02...	Gloss Bomb Cream	4	29	Brand new Gloss Bomb...	593aab5d-96f6-459a-...	91f8b3e9-547a-480d-...	false
3	cc5a009e-9e05-49f3-af6e-...	Batman Batmobile Tu...	21	159	Brand new Batman Bat...	593aab5d-96f6-459a-...	036917a3-716a-4ada-...	false

Fixed Price Listing table after the test:

	id [PK] character varying (36)	listing_title character varying (50)	quantity integer	price real	description character varying (200)	seller_group character varying (36)	product character varying (36)	archive boolean
1	0e923f33-13cd-4189-afa8...	iPhone 14 Pro Max 25...	6	2099	Brand new iPhone 14 P...	593aab5d-96f6-459a-...	7fbd1bdb-5573-4975-...	false
2	16f01056-7382-4c79-9e02...	Gloss Bomb Cream	4	29	Brand new Gloss Bomb...	593aab5d-96f6-459a-...	91f8b3e9-547a-480d-...	false
3	5ae76727-2f00-4a46-952f...	iphone 14 pro max	10	1000	iphone	593aab5d-96f6-459a-...	7fbd1bdb-5573-4975-...	false
4	cc5a009e-9e05-49f3-af6e-...	Batman Batmobile Tu...	21	159	Brand new Batman Bat...	593aab5d-96f6-459a-...	036917a3-716a-4ada-...	false

Concurrency Issue 2

The concurrency issue is referring to a scenario where a seller tries to view an order tied to a seller group, while the admin tries to remove the seller from the seller group.

Testing Approach

1	Seller views an order tied to a seller group while the admin removes the seller from the seller group.

Expected Result

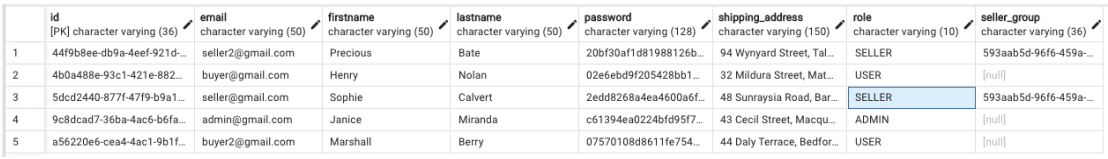
Sub test cases	Expected Result
1	Two possible results may occur: ● The seller is removed from the seller group and the seller is logged out by the system when trying to view an order tied to the seller group. ● The seller is able to view an order tied to the seller group and the seller is removed from the seller group.

Testing configuration

Sub test cases	Configuration
1	Purchase: ● id = 965b0df7-6a53-437e-bb2a-fae045f23f88

	Seller: ● id = 5dcd2440-877f-47f9-b9a1-b708a5bd9bc2 ● View the order '965b0df7-6a53-437e-bb2a-fae045f23f88'. Admin: ● Remove seller '5dcd2440-877f-47f9-b9a1-b708a5bd9bc2'

Actual Result

Sub-test cases	Test Result	Status
1	The seller was able to view the listing that is tied to the seller group, and the admin has removed the seller from the seller group. Account table before the test:  Account table after the test: 	Pass

	id [PK] character varying (36)	email character varying (50)	firstname character varying (50)	lastname character varying (50)	password character varying (128)	shipping_address character varying (150)	role character varying (10)	seller_group character varying (36)
1	44f9b8ee-db9a-4eef-921d-...	seller2@gmail.com	Precious	Bate	20bf30af1d81988126b...	94 Wynyard Street, Tal...	SELLER	593aab5d-96f6-459a-...
2	4b0a488e-93c1-421e-882...	buyer@gmail.com	Henry	Nolan	02e6ebd9f205428bb1...	32 Mildura Street, Mat...	USER	[null]
3	5dcd2440-877f-47f9-b9a1...	seller@gmail.com	Sophie	Calvert	2edd8268a4ea4600a6f...	48 Sunraysia Road, Bar...	SELLER	593aab5d-96f6-459a-...
4	9c8dcad7-36ba-4ac6-b6fa...	admin@gmail.com	Janice	Miranda	c61394ea0224bfd95f7...	43 Cecil Street, Macqu...	ADMIN	[null]
5	a56220e6-cea4-4ac1-9b1f...	buyer2@gmail.com	Marshall	Berry	07570108d8611fe754...	44 Daly Terrace, Bedfor...	USER	[null]

	id [PK] character varying (36)	email character varying (50)	firstname character varying (50)	lastname character varying (50)	password character varying (128)	shipping_address character varying (150)	role character varying (10)	seller_group character varying (36)
1	44f9b8ee-db9a-4eef-921d-...	seller2@gmail.com	Precious	Bate	20bf30af1d81988126b...	94 Wynyard Street, Tal...	SELLER	593aab5d-96f6-459a-...
2	4b0a488e-93c1-421e-882...	buyer@gmail.com	Henry	Nolan	02e6ebd9f205428bb1...	32 Mildura Street, Mat...	USER	[null]
3	5dcd2440-877f-47f9-b9a1...	seller@gmail.com	Sophie	Calvert	2edd8268a4ea4600a6f...	48 Sunraysia Road, Bar...	USER	[null]
4	9c8dcad7-36ba-4ac6-b6fa...	admin@gmail.com	Janice	Miranda	c61394ea0224bfd95f7...	43 Cecil Street, Macqu...	ADMIN	[null]
5	a56220e6-cea4-4ac1-9b1f...	buyer2@gmail.com	Marshall	Berry	07570108d8611fe754...	44 Daly Terrace, Bedfor...	USER	[null]

Concurrency Issue 3

The concurrency issue is referring to a scenario where a seller tries to modify an order that is tied to the seller group, while at the same time the buyer tries to modify the same order.

Testing Approach

Let the seller modify the order that is tied to the seller group, while at the same time let the buyer modify the same order. There will be 4 possible sub test cases for this concurrency issue.	
1	Seller decreases the order quantity while buyer increases the order quantity.
2	Buyer increases the order quantity while the seller cancels the order.
3	Buyer cancels the order while the seller changes the order status from 'placed' to

	‘processed’.
4	Buyer cancels the order while the seller changes the order status from ‘processed’ to ‘fulfilled’.

Expected Result

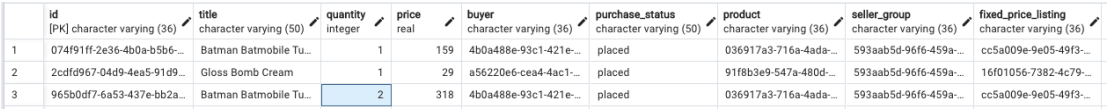
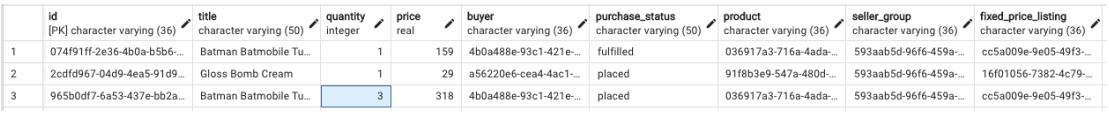
Sub test cases	Expected Result
1	Two possible results may occur: ● The buyer successfully updated the order quantity while the seller failed to update the order quantity. ● The seller successfully updated the order quantity while the buyer failed to update the order quantity.
2	The order will be cancelled no matter if the order quantity has changed or not.
3	The order will be cancelled.
4	Two possible results may occur: ● The order will be cancelled. ● The order will be fulfilled.

Testing configuration

Sub test cases	Configuration
1	Purchase: ● id = 965b0df7-6a53-437e-bb2a-fae045f23f88 ● Quantity = 2 Buyer: ● Increase the quantity of the order ‘965b0df7-6a53-437e-bb2a-fae045f23f88’ by 1. Seller: ● Decrease the quantity of the order ‘965b0df7-6a53-437e-bb2a-fae045f23f88’ by 1.
2	Purchase: ● id = 965b0df7-6a53-437e-bb2a-fae045f23f88 ● Quantity = 2 Buyer: ● Increase the quantity of the order ‘965b0df7-6a53-437e-bb2a-fae045f23f88’ by 1.

	Seller: ● Cancels the order ‘965b0df7-6a53-437e-bb2a-fae045f23f88’.
3	Purchase: ● id = 965b0df7-6a53-437e-bb2a-fae045f23f88 ● Quantity = 2 Buyer: ● Cancels the order ‘965b0df7-6a53-437e-bb2a-fae045f23f88’. Seller: ● Changes the order status of the order ‘965b0df7-6a53-437e-bb2a-fae045f23f88’ to ‘processed’.
4	Purchase: ● id = 965b0df7-6a53-437e-bb2a-fae045f23f88 ● Quantity = 2 Buyer: ● Cancels the order ‘965b0df7-6a53-437e-bb2a-fae045f23f88’. Seller: ● Changes the order status of the order ‘965b0df7-6a53-437e-bb2a-fae045f23f88’ to fulfilled’.

Actual Result

Sub-test cases	Test Result	Status
1	Only the buyer’s request got accepted. The buyer has successfully updated the order quantity while the seller failed to update the order quantity. Purchase table before the test:  Purchase table after the test: 	Pass
2	Both requests got accepted. The order quantity has increased by 1 and the order got cancelled. The listing quantity has increased by the amount that was in the cancelled order. Purchase table before the test:	Pass

Purchase table before the test:

	id [PK] character varying (36)	title character varying (50)	quantity integer	price real	buyer character varying (36)	purchase_status character varying (50)	product character varying (36)	seller_group character varying (36)	fixed_price_listing character varying (36)
1	074f91ff-2e36-4b0a-b5b6-...	Batman Batmobile Tu...	1	159	4b0a488e-93c1-421e-...	placed	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...
2	2cdfd967-04d9-4ea5-91d9...	Gloss Bomb Cream	1	29	a56220e6-cea4-4ac1-...	placed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
3	965b0df7-6a53-437e-bb2a...	Batman Batmobile Tu...	2	318	4b0a488e-93c1-421e-...	placed	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...

Purchase table after the test:

	id [PK] character varying (36)	title character varying (50)	quantity integer	price real	buyer character varying (36)	purchase_status character varying (50)	product character varying (36)	seller_group character varying (36)	fixed_price_listing character varying (36)
1	074f91ff-2e36-4b0a-b5b6-...	Batman Batmobile Tu...	1	159	4b0a488e-93c1-421e-...	fulfilled	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...
2	2cdfd967-04d9-4ea5-91d9...	Gloss Bomb Cream	1	29	a56220e6-cea4-4ac1-...	placed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
3	965b0df7-6a53-437e-bb2a...	Batman Batmobile Tu...	3	318	4b0a488e-93c1-421e-...	placed	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...

	id [PK] character varying (36)	title character varying (50)	quantity integer	price real	buyer character varying (36)	purchase_status character varying (50)	product character varying (36)	seller_group character varying (36)	fixed_price_listing character varying (36)
1	074f91ff-2e36-4b0a-b5b6-...	Batman Batmobile Tu...	1	159	4b0a488e-93c1-421e-...	placed	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...
2	2cdfd967-04d9-4ea5-91d9...	Gloss Bomb Cream	1	29	a56220e6-cea4-4ac1-...	placed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
3	965b0df7-6a53-437e-bb2a...	Batman Batmobile Tu...	2	318	4b0a488e-93c1-421e-...	placed	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...

Purchase table after the test:

	id [PK] character varying (36)	title character varying (50)	quantity integer	price real	buyer character varying (36)	purchase_status character varying (50)	product character varying (36)	seller_group character varying (36)	fixed_price_listing character varying (36)
1	074f91ff-2e36-4b0a-b5b6-...	Batman Batmobile Tu...	1	159	4b0a488e-93c1-421e-...	placed	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...
2	2cdfd967-04d9-4ea5-91d9...	Gloss Bomb Cream	1	29	a56220e6-cea4-4ac1-...	placed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
3	965b0df7-6a53-437e-bb2a...	Batman Batmobile Tu...	3	318	4b0a488e-93c1-421e-...	cancelled	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...

Fixed Price Listing before the table:

	id [PK] character varying (36)	listing_title character varying (50)	quantity integer	price real	description character varying (200)	seller_group character varying (36)	product character varying (36)	archive boolean
1	0e923f33-13cd-4189-afa8...	iPhone 14 Pro Max 25...	6	2099	Brand new iPhone 14 P...	593aab5d-96f6-459a-...	7fbd1bdb-5573-4975-...	false
2	16f01056-7382-4c79-9e02...	Gloss Bomb Cream	4	29	Brand new Gloss Bomb...	593aab5d-96f6-459a-...	91f8b3e9-547a-480d-...	false
3	cc5a009e-9e05-49f3-af6e-...	Batman Batmobile Tu...	13	159	Brand new Batman Bat...	593aab5d-96f6-459a-...	036917a3-716a-4ada-...	false

Fixed Price Listing after the table:

	id [PK] character varying (36)	listing_title character varying (50)	quantity integer	price real	description character varying (200)	seller_group character varying (36)	product character varying (36)	archive boolean
1	0e923f33-13cd-4189-afa8...	iPhone 14 Pro Max 25...	6	2099	Brand new iPhone 14 P...	593aab5d-96f6-459a-...	7fbd1bdb-5573-4975-...	false
2	16f01056-7382-4c79-9e02...	Gloss Bomb Cream	4	29	Brand new Gloss Bomb...	593aab5d-96f6-459a-...	91f8b3e9-547a-480d-...	false
3	cc5a009e-9e05-49f3-af6e-...	Batman Batmobile Tu...	15	159	Brand new Batman Bat...	593aab5d-96f6-459a-...	036917a3-716a-4ada-...	false

3	Both requests got accepted, with the seller's request getting processed first. The order status has become 'cancelled'.	Pass

Purchase table before the test:

	id [PK] character varying (36)	title character varying (50)	quantity integer	price real	buyer character varying (36)	purchase_status character varying (50)	product character varying (36)	seller_group character varying (36)	fixed_price_listing character varying (36)
1	074f91ff-2e36-4b0a-b5b6-...	Batman Batmobile Tu...	1	159	4b0a488e-93c1-421e-...	processed	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...
2	2cdfd967-04d9-4ea5-91d9...	Gloss Bomb Cream	1	29	a56220e6-cea4-4ac1-...	placed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
3	965b0df7-6a53-437e-bb2a...	Batman Batmobile Tu...	3	318	4b0a488e-93c1-421e-...	placed	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...

Purchase table after the test:

	id [PK] character varying (36)	title character varying (50)	quantity integer	price real	buyer character varying (36)	purchase_status character varying (50)	product character varying (36)	seller_group character varying (36)	fixed_price_listing character varying (36)
1	074f91ff-2e36-4b0a-b5b6-...	Batman Batmobile Tu...	1	159	4b0a488e-93c1-421e-...	processed	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...
2	2cdfd967-04d9-4ea5-91d9...	Gloss Bomb Cream	1	29	a56220e6-cea4-4ac1-...	placed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
3	965b0df7-6a53-437e-bb2a...	Batman Batmobile Tu...	3	318	4b0a488e-93c1-421e-...	cancelled	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...

4	Only the seller's request got accepted, the order status has become 'fulfilled'. The buyer's request gets denied due to the order status being 'fulfilled'.	Pass

Purchase table before the test:

	id [PK] character varying (36)	title character varying (50)	quantity integer	price real	buyer character varying (36)	purchase_status character varying (50)	product character varying (36)	seller_group character varying (36)	fixed_price_listing character varying (36)
1	074f91ff-2e36-4b0a-b5b6-...	Batman Batmobile Tu...	1	159	4b0a488e-93c1-421e-...	processed	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...
2	2cdfd967-04d9-4ea5-91d9...	Gloss Bomb Cream	1	29	a56220e6-cea4-4ac1-...	placed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
3	965b0df7-6a53-437e-bb2a...	Batman Batmobile Tu...	3	318	4b0a488e-93c1-421e-...	processed	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...

Purchase table after the test:

	id [PK] character varying (36)	title character varying (50)	quantity integer	price real	buyer character varying (36)	purchase_status character varying (50)	product character varying (36)	seller_group character varying (36)	fixed_price_listing character varying (36)
1	074f91ff-2e36-4b0a-b5b6-...	Batman Batmobile Tu...	1	159	4b0a488e-93c1-421e-...	fulfilled	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...
2	2cdfd967-04d9-4ea5-91d9...	Gloss Bomb Cream	1	29	a56220e6-cea4-4ac1-...	placed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
3	965b0df7-6a53-437e-bb2a...	Batman Batmobile Tu...	3	318	4b0a488e-93c1-421e-...	fulfilled	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...

Concurrency Issue 4

The concurrency issue is referring to a scenario where a seller tries to modify an order, while at the same time the admin tries to remove the seller from the seller group.

Testing Approach

Let the seller modify an order, while at the same time let the admin remove the seller from the seller group. There will be 2 possible sub test cases for this concurrency issue.	
1	Seller decreases the quantity of an order, while admin removes the seller from the seller group.
2	Seller cancels an order, while admin removes the seller from the seller group.

Expected Result

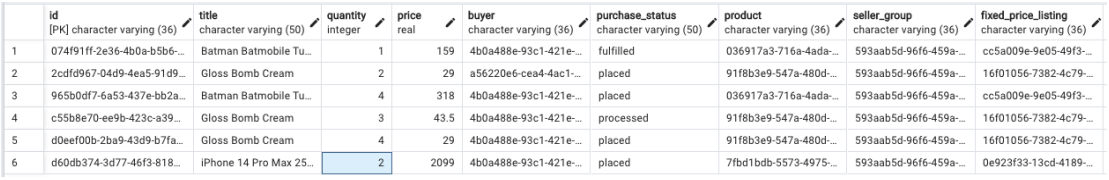
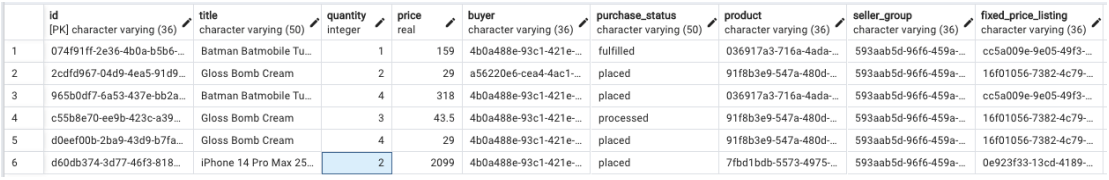
Sub test cases	Expected Result
1	Two possible results may occur: ● The seller successfully changed the quantity of an order, and the admin removed the seller from the seller group. ● The admin removed the seller from the seller group, and the seller failed to change the quantity of an order.
	Two possible results may occur: ● The seller successfully cancelled the order, and the admin removed the seller from the seller group. ● The admin removed the seller from the seller group, and the seller failed to cancel the order.

Testing configuration

Sub test cases	Configuration
1	Purchase: ● id = 965b0df7-6a53-437e-bb2a-fae045f23f88 ● Quantity = 2 Admin: ● Remove seller ‘5dcd2440-877f-47f9-b9a1-b708a5bd9bc2’ from the seller group Seller: ● id = 5dcd2440-877f-47f9-b9a1-b708a5bd9bc2 ● Decrease the quantity of the order

	‘965b0df7-6a53-437e-bb2a-fae045f23f88’ by 1.
	Purchase: ● id = 965b0df7-6a53-437e-bb2a-fae045f23f88 ● Quantity = 2 Admin: ● Remove seller ‘5dcd2440-877f-47f9-b9a1-b708a5bd9bc2’ from the seller group Seller: ● id = 5dcd2440-877f-47f9-b9a1-b708a5bd9bc2 ● Cancel the order ‘965b0df7-6a53-437e-bb2a-fae045f23f88’.

Actual Result

Sub-test cases	Test Result	Status
1	Only the admin’s request got accepted. The admin removed the seller from the seller group, and the seller failed to change the quantity of an order. Purchase table before the test:  Purchase table after the test:  Account table before the test:  Account table after the test: 	Pass

Purchase table before the test:

	id [PK] character varying (36)	title character varying (50)	quantity integer	price real	buyer character varying (36)	purchase_status character varying (50)	product character varying (36)	seller_group character varying (36)	fixed_price_listing character varying (36)
1	074f91ff-2e36-4b0a-b5b6-...	Batman Batmobile Tu...	1	159	4b0a488e-93c1-421e-...	fulfilled	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...
2	2cdfd967-04d9-4ea5-91d9...	Gloss Bomb Cream	2	29	a56220e6-cea4-4ac1-...	placed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
3	965b0df7-6a53-437e-bb2a...	Batman Batmobile Tu...	4	318	4b0a488e-93c1-421e-...	placed	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...
4	c55b8e70-ee9b-423c-a39...	Gloss Bomb Cream	3	43.5	4b0a488e-93c1-421e-...	processed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
5	d0eef00b-2ba9-43d9-b7fa...	Gloss Bomb Cream	4	29	4b0a488e-93c1-421e-...	placed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
6	d60db374-3d77-46f3-818...	iPhone 14 Pro Max 25...	2	2099	4b0a488e-93c1-421e-...	placed	7fbd1bdb-5573-4975-...	593aab5d-96f6-459a-...	0e923f33-13cd-4189-...

Purchase table after the test:

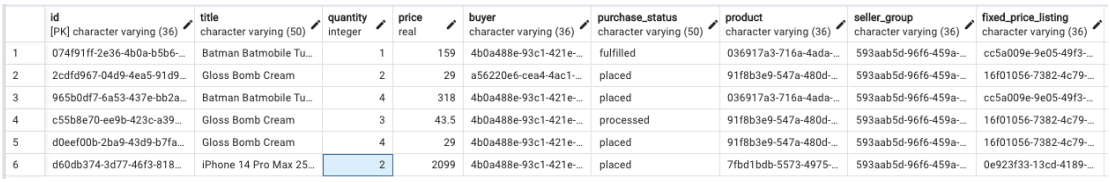
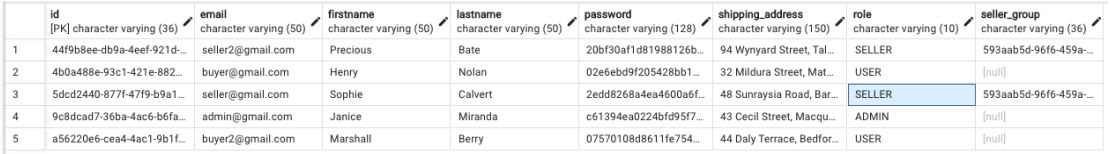
	id [PK] character varying (36)	title character varying (50)	quantity integer	price real	buyer character varying (36)	purchase_status character varying (50)	product character varying (36)	seller_group character varying (36)	fixed_price_listing character varying (36)
1	074f91ff-2e36-4b0a-b5b6-...	Batman Batmobile Tu...	1	159	4b0a488e-93c1-421e-...	fulfilled	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...
2	2cdfd967-04d9-4ea5-91d9...	Gloss Bomb Cream	2	29	a56220e6-cea4-4ac1-...	placed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
3	965b0df7-6a53-437e-bb2a...	Batman Batmobile Tu...	4	318	4b0a488e-93c1-421e-...	placed	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...
4	c55b8e70-ee9b-423c-a39...	Gloss Bomb Cream	3	43.5	4b0a488e-93c1-421e-...	processed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
5	d0eef00b-2ba9-43d9-b7fa...	Gloss Bomb Cream	4	29	4b0a488e-93c1-421e-...	placed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
6	d60db374-3d77-46f3-818...	iPhone 14 Pro Max 25...	2	2099	4b0a488e-93c1-421e-...	placed	7fbd1bdb-5573-4975-...	593aab5d-96f6-459a-...	0e923f33-13cd-4189-...

Account table before the test:

	id [PK] character varying (36)	email character varying (50)	firstname character varying (50)	lastname character varying (50)	password character varying (128)	shipping_address character varying (150)	role character varying (10)	seller_group character varying (36)
1	44f9b8ee-db9a-4eef-921d-...	seller2@gmail.com	Precious	Bate	20bf30af1d81988126b...	94 Wynyard Street, Tal...	SELLER	593aab5d-96f6-459a-...
2	4b0a488e-93c1-421e-882...	buyer@gmail.com	Henry	Nolan	02e6ebd9f205428bb1...	32 Mildura Street, Mat...	USER	[null]
3	5dcd2440-877f-47f9-b9a1...	seller@gmail.com	Sophie	Calvert	2edd8268a4ea4600a6f...	48 Sunraysia Road, Bar...	SELLER	593aab5d-96f6-459a-...
4	9c8dcad7-36ba-4ac6-b6fa...	admin@gmail.com	Janice	Miranda	c61394ea0224bfd95f7...	43 Cecil Street, Macqu...	ADMIN	[null]
5	a56220e6-cea4-4ac1-9b1f...	buyer2@gmail.com	Marshall	Berry	07570108d8611fe754...	44 Daly Terrace, Bedfor...	USER	[null]

Account table after the test:

	id [PK] character varying (36)	email character varying (50)	firstname character varying (50)	lastname character varying (50)	password character varying (128)	shipping_address character varying (150)	role character varying (10)	seller_group character varying (36)
1	44f9b8ee-db9a-4eef-921d-...	seller2@gmail.com	Precious	Bate	20bf30af1d81988126b...	94 Wynyard Street, Tal...	SELLER	593aab5d-96f6-459a-...
2	4b0a488e-93c1-421e-882...	buyer@gmail.com	Henry	Nolan	02e6ebd9f205428bb1...	32 Mildura Street, Mat...	USER	[null]
3	5dcd2440-877f-47f9-b9a1...	seller@gmail.com	Sophie	Calvert	2edd8268a4ea4600a6f...	48 Sunraysia Road, Bar...	USER	[null]
4	9c8dcad7-36ba-4ac6-b6fa...	admin@gmail.com	Janice	Miranda	c61394ea0224bfd95f7...	43 Cecil Street, Macqu...	ADMIN	[null]
5	a56220e6-cea4-4ac1-9b1f...	buyer2@gmail.com	Marshall	Berry	07570108d8611fe754...	44 Daly Terrace, Bedfor...	USER	[null]

2	Only the admin's request got accepted. The admin removed the seller from the seller group, and the seller failed to cancel the order. Purchase table before the test:  Purchase table after the test:  Account table before the test:  Account table after the test: 	Pass

id [PK] character varying (36)	title character varying (50)	quantity integer	price real	buyer character varying (36)	purchase_status character varying (50)	product character varying (36)	seller_group character varying (36)	fixed_price_listing character varying (36)
1	074f91ff-2e36-4b0a-b5b6-...	Batman Batmobile Tu...	1	159	4b0a488e-93c1-421e-...	fulfilled	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...
2	2cdfd967-04d9-4ea5-91d9...	Gloss Bomb Cream	2	29	a56220e6-cea4-4ac1-...	placed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
3	965b0df7-6a53-437e-bb2a...	Batman Batmobile Tu...	4	318	4b0a488e-93c1-421e-...	placed	036917a3-716a-4ada-...	593aab5d-96f6-459a-...	cc5a009e-9e05-49f3-...
4	c55b8e70-ee9b-423c-a39...	Gloss Bomb Cream	3	43.5	4b0a488e-93c1-421e-...	processed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
5	d0eef00b-2ba9-43d9-b7fa...	Gloss Bomb Cream	4	29	4b0a488e-93c1-421e-...	placed	91f8b3e9-547a-480d-...	593aab5d-96f6-459a-...	16f01056-7382-4c79-...
6	d60db374-3d77-46f3-818...	iPhone 14 Pro Max 25...	2	2099	4b0a488e-93c1-421e-...	placed	7fbd1bdb-5573-4975-...	593aab5d-96f6-459a-...	0e923f33-13cd-4189-...

id [PK] character varying (36)	email character varying (50)	firstname character varying (50)	lastname character varying (50)	password character varying (128)	shipping_address character varying (150)	role character varying (10)	seller_group character varying (36)
1	44f9b8ee-db9a-4eef-921d-...	seller2@gmail.com	Precious	Bate	20bf30af1d81988126b...	94 Wynyard Street, Tal...	SELLER	593aab5d-96f6-459a-...
2	4b0a488e-93c1-421e-882...	buyer@gmail.com	Henry	Nolan	02e6ebd9f205428bb1...	32 Mildura Street, Mat...	USER	[null]
3	5dcd2440-877f-47f9-b9a1...	seller@gmail.com	Sophie	Calvert	2edd8268a4ea4600a6f...	48 Sunraysia Road, Bar...	SELLER	593aab5d-96f6-459a-...
4	9c8dcad7-36ba-4ac6-b6fa...	admin@gmail.com	Janice	Miranda	c61394ea0224bfd95f7...	43 Cecil Street, Macqu...	ADMIN	[null]
5	a56220e6-cea4-4ac1-9b1f...	buyer2@gmail.com	Marshall	Berry	07570108d8611fe754...	44 Daly Terrace, Bedfor...	USER	[null]

Concurrency issue 5

Two buyers trying to place an order on the same fixed price listing at the same time.

Testing Approach

Let two buyers send their order requests to the system simultaneously. There will be 2 possible sub test cases for this concurrency issue.	
1	The item quantity is enough to fulfil both orders.
2	The item quantity is not enough to fulfil both orders.

Expected Result

Sub test cases	Expected Result
1	Both orders were accepted by the system and created the records in the purchase table. The fixed-price listing quantity should be correctly deducted.
2	Either buyer 1 or buyer 2's order gets accepted by the system and creates the purchase record in the database, while the other buyer's request will be rejected. The fixed-price listing quantity should be correctly deducted.

Testing configuration

Sub test cases	Configuration
1	FixPrice listing ● Listing id = cc5a009e-9e05-49f3-af6e-90cfc0f8ee09 ● Quantity = 6 Buyer1 ● Id = 4b0a488e-93c1-421e-8823-1459ac7acac8 ● Purchase quantity = 2 Buyer2 ● Id = a56220e6-cea4-4ac1-9b1f-3dd586f822b0 ● Purchase quantity = 2
2	FixPrice listing ● Listing id = cc5a009e-9e05-49f3-af6e-90cfc0f8ee09 ● Quantity = 2 Buyer1 ● Id = 4b0a488e-93c1-421e-8823-1459ac7acac8 ● Purchase quantity = 2 Buyer2 ● Id = a56220e6-cea4-4ac1-9b1f-3dd586f822b0 ● Purchase quantity = 2

Actual Result

Sub-test cases	Test Result	Status
1	**Before Testing:** Purchase table: id [PK] character varying (36) / title character varying (50) / quantity integer / price real / buyer character varying (36) / purchase_status character varying (50) / product character varying (36)	Pass

Fix Price Listing:

	id [PK] character varying (36)	listing_title character va	quantity integer
1	0e923f33-13cd-4189-afa8-db65f386fef9	iPhone 1...	5
2	16f01056-7382-4c79-9e02-a0ea91402ed8	Gloss Bo...	7
3	cc5a009e-9e05-49f3-af6e-90cfc0f8ee09	Batman B...	6

After Testing:
Both users' purchases got accepted.

Purchase table:

	id [PK] character varying (36)	title character varying (50)	quantity integer	price real	buyer character varying (36)	purchase_status character varying (50)	product character varying (36)
1	20a49bb8-432e-41e1-91aa-caa80cbe6ddf	Batman Batmobile Tumbler	2	318	a56220e6-cea4-4ac1-9b1f-3dd586f822b0	placed	036917a3-716a-4ada-9af3-46fb442406a9
2	a8c00a28-3123-4b0e-a117-e9e9d3b81420	Batman Batmobile Tumbler	2	318	4b0a488e-93c1-421e-8823-1459ac7acac8	placed	036917a3-716a-4ada-9af3-46fb442406a9

Fix Price Listing:

The quantity of the listing has been deducted correctly, from 6 to 2.

	id [PK] character varying (36)	listing_title character varying (50)	quantity integer
1	0e923f33-13cd-4189-afa8-db65f386fef9	iPhone 14 Pro Max 256GB	5
2	16f01056-7382-4c79-9e02-a0ea91402ed8	Gloss Bomb Cream	7
3	cc5a009e-9e05-49f3-af6e-90cfc0f8ee09	Batman Batmobile Tumbler	2

2

Before Testing:

Pass

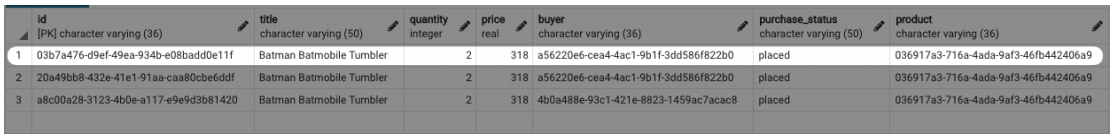
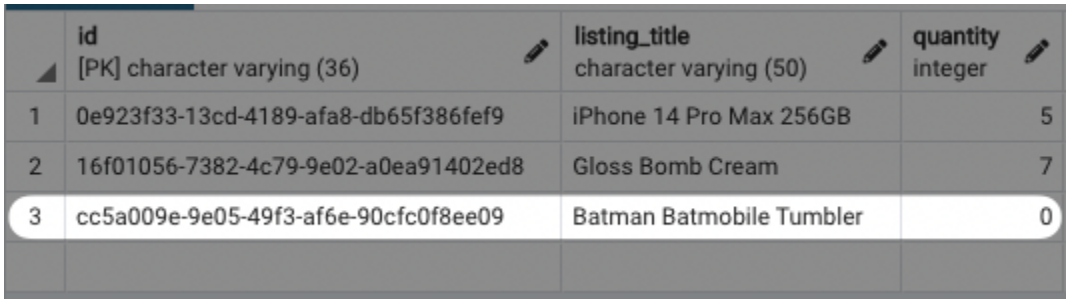
Purchase table:

	id [PK] character varying (36)	title character varying (50)	quantity integer	price real	buyer character varying (36)	purchase_status character varying (50)	product character varying (36)
1	20a49bb8-432e-41e1-91aa-caa80cbe6ddf	Batman Batmobile Tumbler	2	318	a56220e6-cea4-4ac1-9b1f-3dd586f822b0	placed	036917a3-716a-4ada-9af3-46fb442406a9
2	a8c00a28-3123-4b0e-a117-e9e9d3b81420	Batman Batmobile Tumbler	2	318	4b0a488e-93c1-421e-8823-1459ac7acac8	placed	036917a3-716a-4ada-9af3-46fb442406a9

Fix Price Listing:

	id [PK] character varying (36)	listing_title character varying (50)	quantity integer
1	0e923f33-13cd-4189-afa8-db65f386fef9	iPhone 14 Pro Max 256GB	5
2	16f01056-7382-4c79-9e02-a0ea91402ed8	Gloss Bomb Cream	7
3	cc5a009e-9e05-49f3-af6e-90cfc0f8ee09	Batman Batmobile Tumbler	2

After Testing:

	Only Buyer2 got the order and created the purchase record in the database. Purchase table:  Fix Price Listing: The quantity of the listing has been deducted correctly, from 2 to 0. 	

	id [PK] character varying (36)	title character varying (50)	quantity integer	price real	buyer character varying (36)	purchase_status character varying (50)	product character varying (36)
1	03b7a476-d9ef-49ea-934b-e08badd0e11f	Batman Batmobile Tumbler	2	318	a56220e6-cea4-4ac1-9b1f-3dd586f822b0	placed	036917a3-716a-4ada-9af3-46fb442406a9
2	20a49bb8-432e-41e1-91aa-caa80cbe6ddf	Batman Batmobile Tumbler	2	318	a56220e6-cea4-4ac1-9b1f-3dd586f822b0	placed	036917a3-716a-4ada-9af3-46fb442406a9
3	a8c00a28-3123-4b0e-a117-e9e9d3b81420	Batman Batmobile Tumbler	2	318	4b0a488e-93c1-421e-8823-1459ac7acac8	placed	036917a3-716a-4ada-9af3-46fb442406a9

	id [PK] character varying (36)	listing_title character varying (50)	quantity integer
1	0e923f33-13cd-4189-afa8-db65f386fef9	iPhone 14 Pro Max 256GB	5
2	16f01056-7382-4c79-9e02-a0ea91402ed8	Gloss Bomb Cream	7
3	cc5a009e-9e05-49f3-af6e-90cfc0f8ee09	Batman Batmobile Tumbler	0

Concurrency Issue 6

The concurrency issue refers to a scenario where an admin tries to remove a fixed-price listing, while, at the same time, the buyer tries to purchase an item from the same fixed price listing.

Testing Approach

Let the admin send a request to remove a fixed price listing, while at the same time, the buyer sends a request to purchase the item from the same fixed price listing. There will only be one possible test case for this concurrency issue. Note: The quantity of the buyer's purchase request must be equal to or under the available quantity of the fixed-price listing. Otherwise, the purchase request will not be accepted by the system, thus deviating from the purpose of the testing.	
1	The item quantity is enough to fulfil the buyer's order while the admin tries to remove the listing from the system.

Expected Result

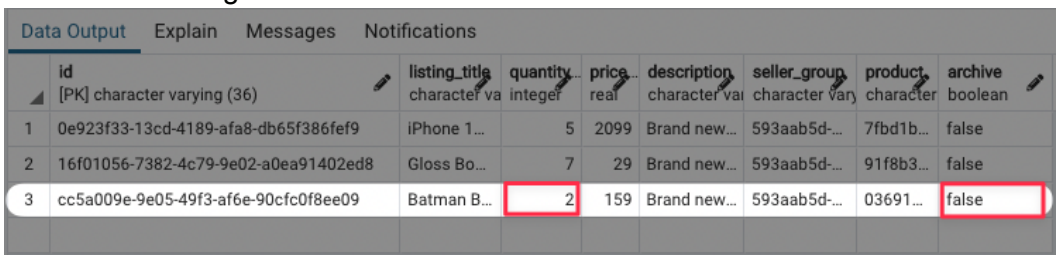
Sub-test	Expected Result

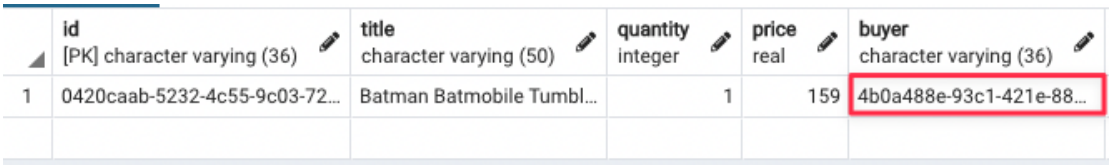
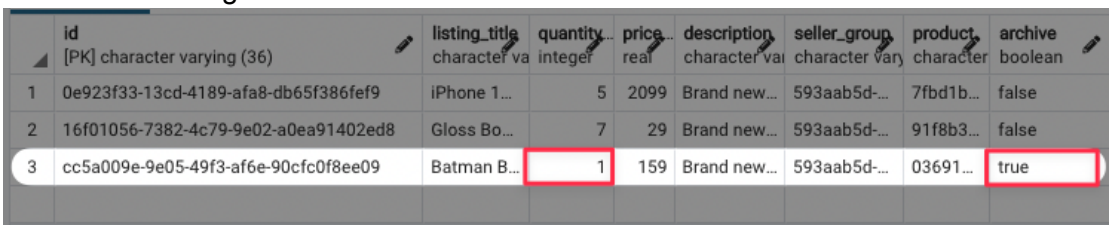
cases	
1	Whether the buyer ultimately buys the item or not depends on which request the system handles first. The focal point of this testing is that the status of the fixed price listing, archive = true, should not be overwritten by the buyer's request, regardless of the processing order. Please note, the processing order of the request is out of the control of the test case and JMeter, hence the team will only be showing one test run and its result.

Testing configuration

Sub test cases	Configuration
1	FixPrice listing ● Listing id = cc5a009e-9e05-49f3-af6e-90cfc0f8ee09 ● Quantity = 2 Admin ● Id = 9c8dcad7-36ba-4ac6-b6fa-630e1883f4c9 Buyer1 ● Id = 4b0a488e-93c1-421e-8823-1459ac7acac8 ● Purchase quantity = 1

Actual Result

Sub test cases	Test Result	Status
1	**Before Testing:** Purphase table: empty purphase table  Fixed Price Listing table: 	Pass

	After Testing: Buyer's purchase order get processed first before the system archives the fixed price listing. The system has created a purchase record accordingly in the purphase table. Purphase table:  As shown in the image below, the listing has been archived successfully while the quantity of the listing is also deducted correctly, from 2 to 1. Fixed Price Listing table: 	

	id [PK] character varying (36)	title character varying (50)	quantity integer	price real	buyer character varying (36)
1	0420caab-5232-4c55-9c03-72...	Batman Batmobile Tumbl...	1	159	4b0a488e-93c1-421e-88...

	id [PK] character varying (36)	listing_title character va	quantity integer	price real	description character var	seller_group character vary	product character	archive boolean
1	0e923f33-13cd-4189-afa8-db65f386fef9	iPhone 1...	5	2099	Brand new...	593aab5d-...	7fbd1b...	false
2	16f01056-7382-4c79-9e02-a0ea91402ed8	Gloss Bo...	7	29	Brand new...	593aab5d-...	91f8b3...	false
3	cc5a009e-9e05-49f3-af6e-90cfc0f8ee09	Batman B...	1	159	Brand new...	593aab5d-...	03691...	true

Concurrency Issue 7

Two buyers trying to place a bid on the same auction listing.

Testing Approach

Let two buyers send their bid requests to the system simultaneously. As shown in the list down below, there will be 3 possible configurations of the bid requests for this concurrency testing. Note: All the bidding prices placed by the buyer in this test case are equal to or higher than the current auction listing price +1, as the system will not accept the bid request that does not meet the above condition.	
1	Buyer1's bidding price = Buyer2's bidding price +1
2	Buyer1's bidding price > Buyer2's bidding price +1
3	Buyer1's bidding price = Buyer2's bidding price

Expected Result

Sub test cases	Expected Result
1	Buyer1 wins the auction while buyer two's request could either be rejected (not creating any record in the bid table) or outbidded (created a record in the bid table, but immediately gets outbid by the buyer's one bid request).
2	The expected result for this test case is similar to test case 1.
3	Either buyer1 or buyer2 wins the auction, and only the winner gets to create a bid record in the bid table while the other buyer's request will be rejected.

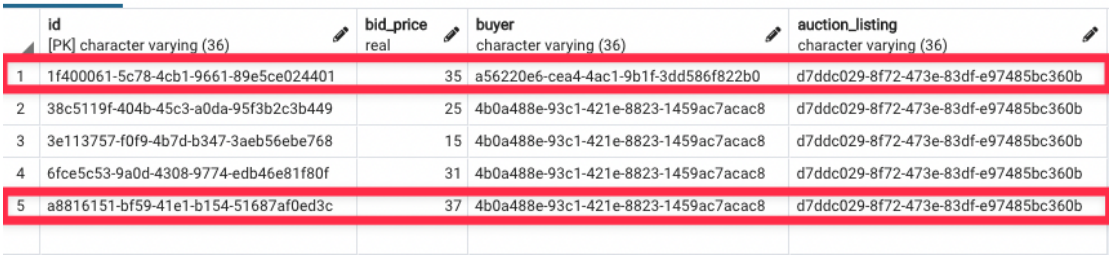
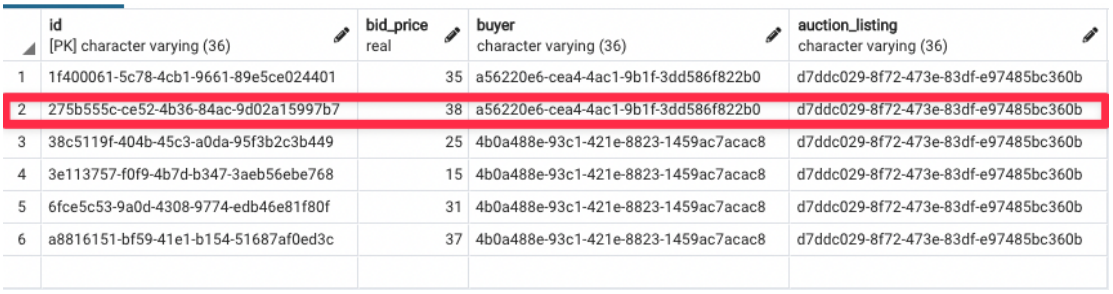
Testing configuration

Sub test cases	Configuration
1	Auction listing ● Listing id = d7ddc029-8f72-473e-83df-e97485bc360b ● Current price = 25 Buyer1 ● Id = 4b0a488e-93c1-421e-8823-1459ac7acac8 ● Bid price = 31 Buyer2 ● Id = a56220e6-cea4-4ac1-9b1f-3dd586f822b0 ● Bid price = 30
2	Auction listing ● Listing id = d7ddc029-8f72-473e-83df-e97485bc360b ● Current price = 31 Buyer1 ● Id = 4b0a488e-93c1-421e-8823-1459ac7acac8 ● Bid price = 37 Buyer2 ● Id = a56220e6-cea4-4ac1-9b1f-3dd586f822b0 ● Bid price = 35
3	Auction listing ● Listing id = d7ddc029-8f72-473e-83df-e97485bc360b ● Current price = 37 Buyer1 ● Id = 4b0a488e-93c1-421e-8823-1459ac7acac8 ● Bid price = 38 Buyer2 ● Id = a56220e6-cea4-4ac1-9b1f-3dd586f822b0

	• Bid price = 38

Actual Result

Sub test cases	Test Result	Status
1	**Before Testing:** Bidding table: id [PK] character varying (36) / bid_price real / buyer character varying (36) / auction_listing character varying (36) 1 38c5119f-404b-45c3-a0da-95f3b2c3b449 25 4b0a488e-93c1-421e-8823-1459ac7acac8 d7ddc029-8f72-473e-83df-e97485bc360b 2 3e113757-f0f9-4b7d-b347-3aeb56ebe768 15 4b0a488e-93c1-421e-8823-1459ac7acac8 d7ddc029-8f72-473e-83df-e97485bc360b Auction Listing table id [PK] character varying (36) / listing_title character varying (50) / quantity integer / start_time timestamp without time zone / end_time timestamp without time zone / price real 1 d7ddc029-8f72-473e-83df-e97485bc360b Nerf Elite 2.0 1 2022-10-15 20:55:08.076317 2022-10-20 02:55:08.076317 25 **After Testing:** Buyer1 wins the action while buyer2's request gets rejected (not able to create a record in the bid table). Bidding table id [PK] character varying (36) / bid_price real / buyer character varying (36) / auction_listing character varying (36) 1 38c5119f-404b-45c3-a0da-95f3b2c3b449 25 4b0a488e-93c1-421e-8823-1459ac7acac8 d7ddc029-8f72-473e-83df-e97485bc360b 2 3e113757-f0f9-4b7d-b347-3aeb56ebe768 15 4b0a488e-93c1-421e-8823-1459ac7acac8 d7ddc029-8f72-473e-83df-e97485bc360b 3 6fce5c53-9a0d-4308-9774-edb46e81f80f 31 4b0a488e-93c1-421e-8823-1459ac7acac8 d7ddc029-8f72-473e-83df-e97485bc360b Auction Listing table id [PK] character varying (36) / listing_title character varying (50) / quantity integer / start_time timestamp without time zone / end_time timestamp without time zone / price real 1 d7ddc029-8f72-473e-83df-e97485bc360b Nerf Elite 2.0 1 2022-10-15 20:55:08.076317 2022-10-20 02:55:08.076317 31	Pass
2	**Before Testing:** Bidding table id [PK] character varying (36) / bid_price real / buyer character varying (36) / auction_listing character varying (36) 1 38c5119f-404b-45c3-a0da-95f3b2c3b449 25 4b0a488e-93c1-421e-8823-1459ac7acac8 d7ddc029-8f72-473e-83df-e97485bc360b 2 3e113757-f0f9-4b7d-b347-3aeb56ebe768 15 4b0a488e-93c1-421e-8823-1459ac7acac8 d7ddc029-8f72-473e-83df-e97485bc360b 3 6fce5c53-9a0d-4308-9774-edb46e81f80f 31 4b0a488e-93c1-421e-8823-1459ac7acac8 d7ddc029-8f72-473e-83df-e97485bc360b Auction Listing table	Pass

	 **After Testing:** Buyer1 wins the action, both buyer1 and buyer2 created records in the bid table. Bidding table  Auction Listing table 	
3	**Before Testing:** Bidding table  Auction Listing table  **After Testing:** Buyer2 wins the bid and only one bidding record has been added to the bidding table. Bidding table  Auction Listing table	Pass

	id [PK] character varying (36)	listing_title character varying (50)	quantity integer	start_time timestamp without time zone	end_time timestamp without time zone	price real
1	d7ddc029-8f72-473e-83df-e97485bc360b	Nerf Elite 2.0	1	2022-10-15 20:55:08.076317	2022-10-20 02:55:08.076317	31

	id [PK] character varying (36)	bid_price real	buyer character varying (36)	auction_listing character varying (36)
1	1f400061-5c78-4cb1-9661-89e5ce024401	35	a56220e6-cea4-4ac1-9b1f-3dd586f822b0	d7ddc029-8f72-473e-83df-e97485bc360b
2	38c5119f-404b-45c3-a0da-95f3b2c3b449	25	4b0a488e-93c1-421e-8823-1459ac7acac8	d7ddc029-8f72-473e-83df-e97485bc360b
3	3e113757-f0f9-4b7d-b347-3aeb56ebe768	15	4b0a488e-93c1-421e-8823-1459ac7acac8	d7ddc029-8f72-473e-83df-e97485bc360b
4	6fce5c53-9a0d-4308-9774-edb46e81f80f	31	4b0a488e-93c1-421e-8823-1459ac7acac8	d7ddc029-8f72-473e-83df-e97485bc360b
5	a8816151-bf59-41e1-b154-51687af0ed3c	37	4b0a488e-93c1-421e-8823-1459ac7acac8	d7ddc029-8f72-473e-83df-e97485bc360b

	id [PK] character varying (36)	listing_title character varying (50)	quantity integer	start_time timestamp without time zone	end_time timestamp without time zone	price real
1	d7ddc029-8f72-473e-83df-e97485bc360b	Nerf Elite 2.0	1	2022-10-15 20:55:08.076317	2022-10-20 02:55:08.076317	37

	id [PK] character varying (36)	bid_price real	buyer character varying (36)	auction_listing character varying (36)
1	1f400061-5c78-4cb1-9661-89e5ce024401	35	a56220e6-cea4-4ac1-9b1f-3dd586f822b0	d7ddc029-8f72-473e-83df-e97485bc360b
2	38c5119f-404b-45c3-a0da-95f3b2c3b449	25	4b0a488e-93c1-421e-8823-1459ac7acac8	d7ddc029-8f72-473e-83df-e97485bc360b
3	3e113757-f0f9-4b7d-b347-3aeb56ebe768	15	4b0a488e-93c1-421e-8823-1459ac7acac8	d7ddc029-8f72-473e-83df-e97485bc360b
4	6fce5c53-9a0d-4308-9774-edb46e81f80f	31	4b0a488e-93c1-421e-8823-1459ac7acac8	d7ddc029-8f72-473e-83df-e97485bc360b
5	a8816151-bf59-41e1-b154-51687af0ed3c	37	4b0a488e-93c1-421e-8823-1459ac7acac8	d7ddc029-8f72-473e-83df-e97485bc360b

	id [PK] character varying (36)	listing_title character varying (50)	quantity integer	start_time timestamp without time zone	end_time timestamp without time zone	price real
1	d7ddc029-8f72-473e-83df-e97485bc360b	Nerf Elite 2.0	1	2022-10-15 20:55:08.076317	2022-10-20 02:55:08.076317	37

	id [PK] character varying (36)	bid_price real	buyer character varying (36)	auction_listing character varying (36)
1	1f400061-5c78-4cb1-9661-89e5ce024401	35	a56220e6-cea4-4ac1-9b1f-3dd586f822b0	d7ddc029-8f72-473e-83df-e97485bc360b
2	275b555c-ce52-4b36-84ac-9d02a15997b7	38	a56220e6-cea4-4ac1-9b1f-3dd586f822b0	d7ddc029-8f72-473e-83df-e97485bc360b
3	38c5119f-404b-45c3-a0da-95f3b2c3b449	25	4b0a488e-93c1-421e-8823-1459ac7acac8	d7ddc029-8f72-473e-83df-e97485bc360b
4	3e113757-f0f9-4b7d-b347-3aeb56ebe768	15	4b0a488e-93c1-421e-8823-1459ac7acac8	d7ddc029-8f72-473e-83df-e97485bc360b
5	6fce5c53-9a0d-4308-9774-edb46e81f80f	31	4b0a488e-93c1-421e-8823-1459ac7acac8	d7ddc029-8f72-473e-83df-e97485bc360b
6	a8816151-bf59-41e1-b154-51687af0ed3c	37	4b0a488e-93c1-421e-8823-1459ac7acac8	d7ddc029-8f72-473e-83df-e97485bc360b

	id [PK] character varying (36)	listing_title character varying (50)	quantity integer	start_time timestamp without time zone	end_time timestamp without time zone	price real
1	d7ddc029-8f72-473e-83df-e97485bc360b	Nerf Elite 2.0	1	2022-10-15 20:55:08.076317	2022-10-20 02:55:08.076317	38

Concurrency Issue 8

The concurrency issue is referring to a scenario where an admin tries to remove an auction listing, while at the same time the buyer tries to place a bid on the same auction listing.

Testing Approach

Let the admin send a request to remove an auction listing, while at the same time, the buyer sends a request to place a bid on the same auction listing. There will only be one possible test case for this concurrency issue. Note: The bidding price of the buyer's bid request must be equal to or higher than the current price of the fixed-price listing + 1. Otherwise, the bid request will not be accepted by the system, thus deviating from the purpose of the testing.	
1	Buyer1 places a bid that is higher than the current price of the auction listing, while the admin tries to remove the listing from the system.

Expected Result

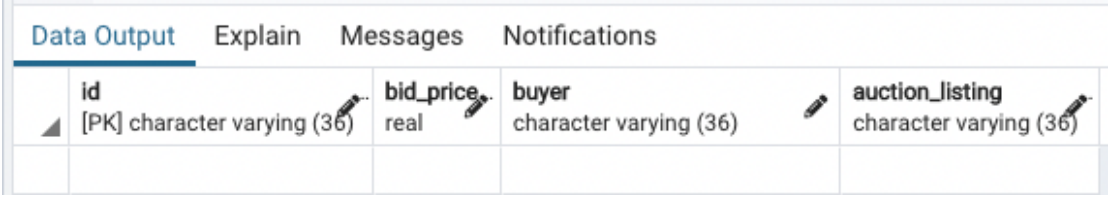
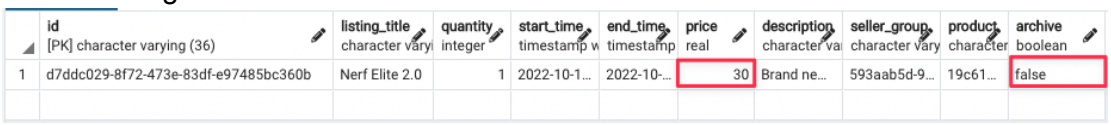
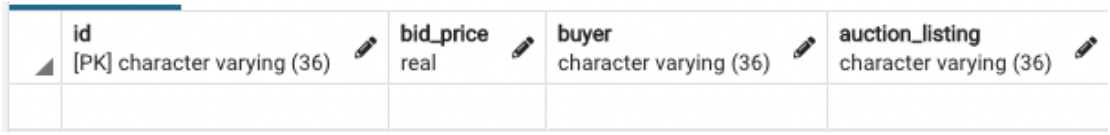
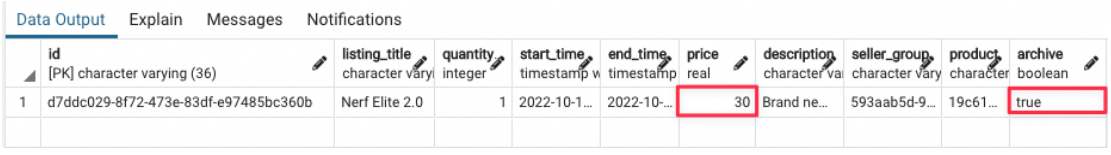
Sub-test cases	Expected Result
1	The expected result of this test case is similar to concurrency Issue 6, where the focal point of this testing is that the status of the auction listing, archive = true, should not be overwritten by the buyer's request, regardless of the processing order. Please note, the processing order of the request is out of the control of the test case and JMeter, hence the team will only be executing one test run and documenting its result.

Testing configuration

Sub test cases	Configuration
1	Auction listing

	• Listing id = d7ddc029-8f72-473e-83df-e97485bc360b • Current price = 30 Admin • Id = 9c8dcad7-36ba-4ac6-b6fa-630e1883f4c9 Buyer1 • Id = 4b0a488e-93c1-421e-8823-1459ac7acac8 • Current price = 40

Actual Result

Sub test cases	Test Result	Status
1	**Before Testing:** Bid table: empty bid table  Data Output Explain Messages Notifications id [PK] character varying (36) · bid_price real · buyer character varying (36) · auction_listing character varying (36) Auction listing table:  id [PK] character varying (36) · listing_title character varyi · quantity integer · start_time timestamp w · end_time timestamp · price real · description character va · seller_group character vary · product character · archive boolean 1 · d7ddc029-8f72-473e-83df-e97485bc360b · Nerf Elite 2.0 · 1 · 2022-10-1... · 2022-10-... · 30 · Brand ne... · 593aab5d-9... · 19c61... · false **After Testing:** The system processes the archive request before processing the buyer's bid request. Bid table: Bid table remains empty.  id [PK] character varying (36) · bid_price real · buyer character varying (36) · auction_listing character varying (36) As shown in the image below, the listing has been archived successfully while the other fields remain unchanged, which inline with the expected result as buyer's bid request gets rejected. Auction Listing table:  Data Output Explain Messages Notifications id [PK] character varying (36) · listing_title character varyi · quantity integer · start_time timestamp w · end_time timestamp · price real · description character va · seller_group character vary · product character · archive boolean 1 · d7ddc029-8f72-473e-83df-e97485bc360b · Nerf Elite 2.0 · 1 · 2022-10-1... · 2022-10-... · 30 · Brand ne... · 593aab5d-9... · 19c61... · true	Pass

Concurrency Issue 9

The concurrency issue is referring to a scenario where a seller tries to update its user profile, while at the same time the admin tries to remove this seller from a seller group.

Testing Approach

Let the admin remove the buyer from the seller group, while the buyer updates its profile. There will only be one possible test case for this concurrency issue.	
1	As described above

Expected Result

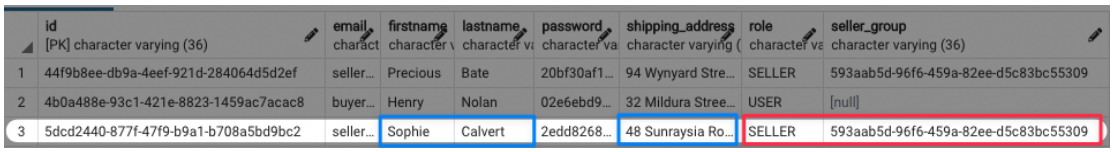
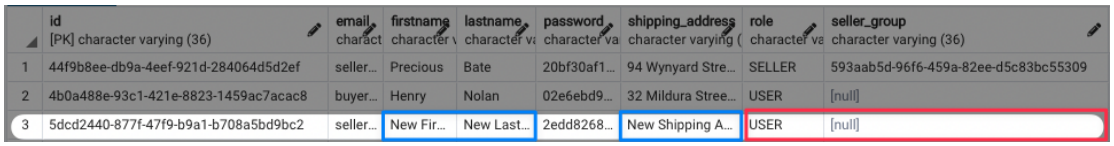
Sub-test cases	Expected Result
1	Regardless of the processing order, both requests should be executed and update the corresponding database table correctly. No request should make changes to the database that overwrite the other request.

Testing configuration

Sub test cases	Configuration
1	Admin ● Id = 9c8dcad7-36ba-4ac6-b6fa-630e1883f4c9 ● Remove seller's id = 5dcd2440-877f-47f9-b9a1-b708a5bd9bc2 ● From seller group's id = 593aab5d-96f6-459a-82ee-d5c83bc55309 Seller ● Id = 5dcd2440-877f-47f9-b9a1-b708a5bd9bc2 ● ChangeFirstname = New Firstname ● ChangeLastname = New Lastname ● ChangeShippingAddress = New Shipping Address

Actual Result

Sub test cases	Test Result	Status
1	**Before Testing:** Account	Pass

	 **After Testing:** Both requests have been executed and updated the corresponding database table correctly. Account 	

id [PK] character varying (36)	email charact	firstname character v	lastname character va	password character va	shipping_address character varying (	role character va	seller_group character varying (36)
1 44f9b8ee-db9a-4eef-921d-284064d5d2ef	seller...	Precious	Bate	20bf30af1...	94 Wynyard Stre...	SELLER	593aab5d-96f6-459a-82ee-d5c83bc55309
2 4b0a488e-93c1-421e-8823-1459ac7acac8	buyer...	Henry	Nolan	02e6ebd9...	32 Mildura Stree...	USER	[null]
3 5dcd2440-877f-47f9-b9a1-b708a5bd9bc2	seller...	Sophie	Calvert	2edd8268...	48 Sunraysia Ro...	SELLER	593aab5d-96f6-459a-82ee-d5c83bc55309

id [PK] character varying (36)	email charact	firstname character v	lastname character va	password character va	shipping_address character varying (	role character va	seller_group character varying (36)
1 44f9b8ee-db9a-4eef-921d-284064d5d2ef	seller...	Precious	Bate	20bf30af1...	94 Wynyard Stre...	SELLER	593aab5d-96f6-459a-82ee-d5c83bc55309
2 4b0a488e-93c1-421e-8823-1459ac7acac8	buyer...	Henry	Nolan	02e6ebd9...	32 Mildura Stree...	USER	[null]
3 5dcd2440-877f-47f9-b9a1-b708a5bd9bc2	seller...	New Fir...	New Last...	2edd8268...	New Shipping A...	USER	[null]

Concurrency Issue 10

The concurrency issue is referring to a scenario where a buyer tries to update its user profile, while at the same time the admin tries to onboard this buyer to a seller group.

Testing Approach

Let the admin onboard the buyer to the seller group, while the buyer updates its profile. There will only be one possible test case for this concurrency issue.	
1	As described above

Expected Result

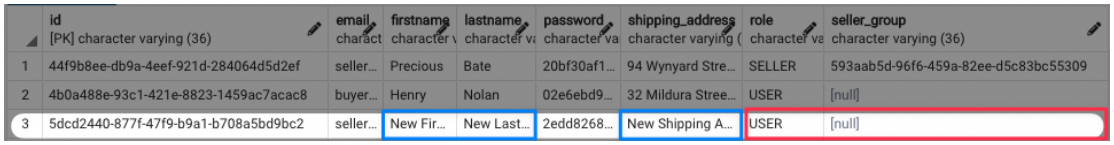
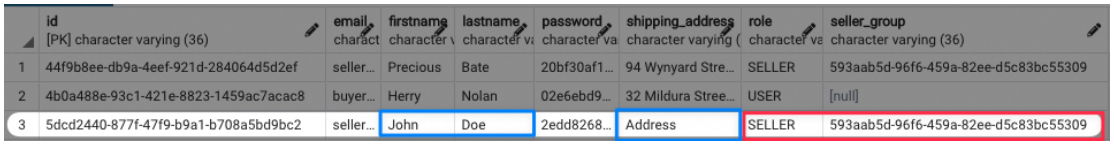
Sub-test cases	Expected Result
1	Regardless of the processing order, both requests should be executed and update the corresponding database table correctly. No request should make changes to the database that overwrite the other request.

Testing configuration

Sub test cases	Configuration
1	Admin

	• Id = 9c8dcad7-36ba-4ac6-b6fa-630e1883f4c9 • Onboard seller's email = seller@gmail.com • Onboard seller's id = 5dcd2440-877f-47f9-b9a1-b708a5bd9bc2 • To seller group's id = 593aab5d-96f6-459a-82ee-d5c83bc55309 Seller • Id = 5dcd2440-877f-47f9-b9a1-b708a5bd9bc2 • ChangeFirstname = John • ChangeLastname = Doe • ChangeShippingAddress = Address

Actual Result

Sub test cases	Test Result	Status
1	**Before Testing:** Account  **After Testing:** Both requests have been executed and updated the corresponding database table correctly. Account 	Pass