

---

## Concurrency and Testing Document

SWEN90007

Marketplace System

Team: Agora

Student Name	Student ID	UniMelb Username	GitHub Username	Email
Daniel Blain	831953	dblain	djblain	<a href="mailto:dblain@student.unimelb.edu.au">dblain@student.unimelb.edu.au</a>
Christopher Byrnes	747295	byrnesc	chrisbyrnes	<a href="mailto:byrnesc@student.unimelb.edu.au">byrnesc@student.unimelb.edu.au</a>
Gonzalo Molina	1085253	gmolina	GonzMol	<a href="mailto:gmolina@student.unimelb.edu.au">gmolina@student.unimelb.edu.au</a>
Mengjiao Wei	1242147	mengjiaow1	mengjiaowei	<a href="mailto:mengjiaow1@student.unimelb.edu.au">mengjiaow1@student.unimelb.edu.au</a>



---

SCHOOL OF  
COMPUTING &  
INFORMATION  
SYSTEMS

---

### Revision History

Date	Version	Description	Author
21/9/2022	03.00-D01	Initial draft	Mengjiao Wei
27/9/2022	03.00-D02	Added draft concurrency issues	Mengjiao Wei
28/9/2022	03.00-D03	Adjusted draft concurrency issues	All
05/10/2022	03.00-D04	Added draft of concurrency patterns	Gonzalo Molina
09/10/2022	03.00-D05	Draft of pattern implementations	Daniel Blain
11/10/2022	03.00-D06	Refined concurrency patterns	Gonzalo Molina
12/10/2022	03.00-D07	Refined issue descriptions	Daniel Blain
17/09/2022	03.00-D08	Added test results	Mengjiao Wei
18/10/2022	03.00-D09	Added class diagram	Christopher Byrnes
19/10/2022	03.00-D10	Added sequence diagrams	Gonzalo Molina
20/10/2022	03.00-D11	Finalised test results	Mengjiao Wei
20/10/2022	03.00-D12	Added appendix	Daniel Blain
20/10/2022	03.00-D13	Finalised document content	All
20/10/2022	03.00-D14	Proofreading	All

# Table of Contents

<b>1 Introduction</b>	<b>5</b>
1.1 Purpose of Document	5
1.2 Target Users	5
1.3 Conventions, terms and abbreviations	5
1.4 Actors	6
<b>2 Class Diagrams</b>	<b>7</b>
<b>3 Implemented concurrency patterns</b>	<b>12</b>
3.1 Optimistic lock	12
3.1.1 Optimistic lock on Listings (when modifying listing details)	12
3.1.2 Optimistic lock on Orders	13
3.2 Read lock	14
3.2.1 Auction Listing	15
3.2.2 User	15
<b>4 Concurrency issues by Use Case</b>	<b>16</b>
4.1 Use Case 5: Manage listings	16
4.1.1 Issue 5.1	17
4.1.2 Issue 5.2	17
4.1.3 Issue 5.3	17
4.1.4 Issue 5.4	17
4.2 Use Case 7: Manage orders	18
4.2.1 Issue 7.1	18
4.2.2 Issue 7.2	18
4.2.3 Issue 7.3	18
4.3 Use Case 8: Bid on auction	19
4.3.1 Issue 8.1	19
4.4 Use Case 9: Purchase a fixed price good	19
4.4.1 Issue 9.1	19
4.5 Use Case 10: Manage all accounts	20
4.5.1 Issue 10.1	20
4.5.2 Issue 10.2	20
4.5.3 Issue 10.3	20
<b>5 Test Strategy</b>	<b>21</b>
5.1 Automation testing tool	21
5.2 Manual testing	21
5.3 Test cases	21
5.3 Testing setups	23
5.4 Test outcomes	23

<b>6 Appendices</b>	<b>35</b>
6.1 Appendix A - Accessing the Heroku Deployment	35
6.2 Appendix B - Test Data	35
6.3 Appendix C - Test Result	35
6.4 Appendix D - Sample User Data	35

# 1 Introduction

## 1.1 Purpose of Document

This document describes the identified concurrency issues present in the Agora marketplace system and approaches to deal with those issues.

## 1.2 Target Users

This document is aimed at the team developing the Agora Marketplace system as well as the teaching staff responsible for assessing the system and work done.

## 1.3 Conventions, terms and abbreviations

This section explains the concept of some important terms that will be used throughout this document. These terms are detailed alphabetically in the following table.

Term	Description
Admin	Special User responsible for overseeing the system's use and managing Listings and Users; short-hand for Administrator
Auction Good	A Good which multiple site users may place a bid on to try and acquire the item; the Good is sold to the user who places the highest bid
Bidding	The act of placing a bid on an Auction good
Buyer	A Marketplace User who purchases Goods
Co-seller	A nominated User which co-manages a Listing created by a Seller
Fixed Price Good	A Good, or Goods, to be sold at a single, non-negotiable price
Good	A product detailed for sale in a Listing
Listing	An entry for a product to be sold as it appears on the Marketplace site
Marketplace	The digital storefront proposed in this document, for buying and selling physical Goods
Order	A purchase, complete or incomplete, of a Good, detailing payee and shipping details
Seller	A Marketplace User who creates Listings for Goods to be sold
User	An individual, or business, registered with the Marketplace system, who can use its services to purchase or sell Goods

## 1.4 Actors

Actor	Description
Administrator	Manages User accounts and Listings; only one Administrator exists for the Marketplace system
All users	Refers to Standard users and Administrators collectively
Standard User	Standard users can create and purchase listings on the system.

## 2 Class Diagrams

Due to the size of system-wide class diagrams, the system has been broken down into multiple different class diagrams organised by package, to show the relationships between classes and layers of the system in a streamlined manner.



Figure 1 - High-Level Class Diagram of the Packages

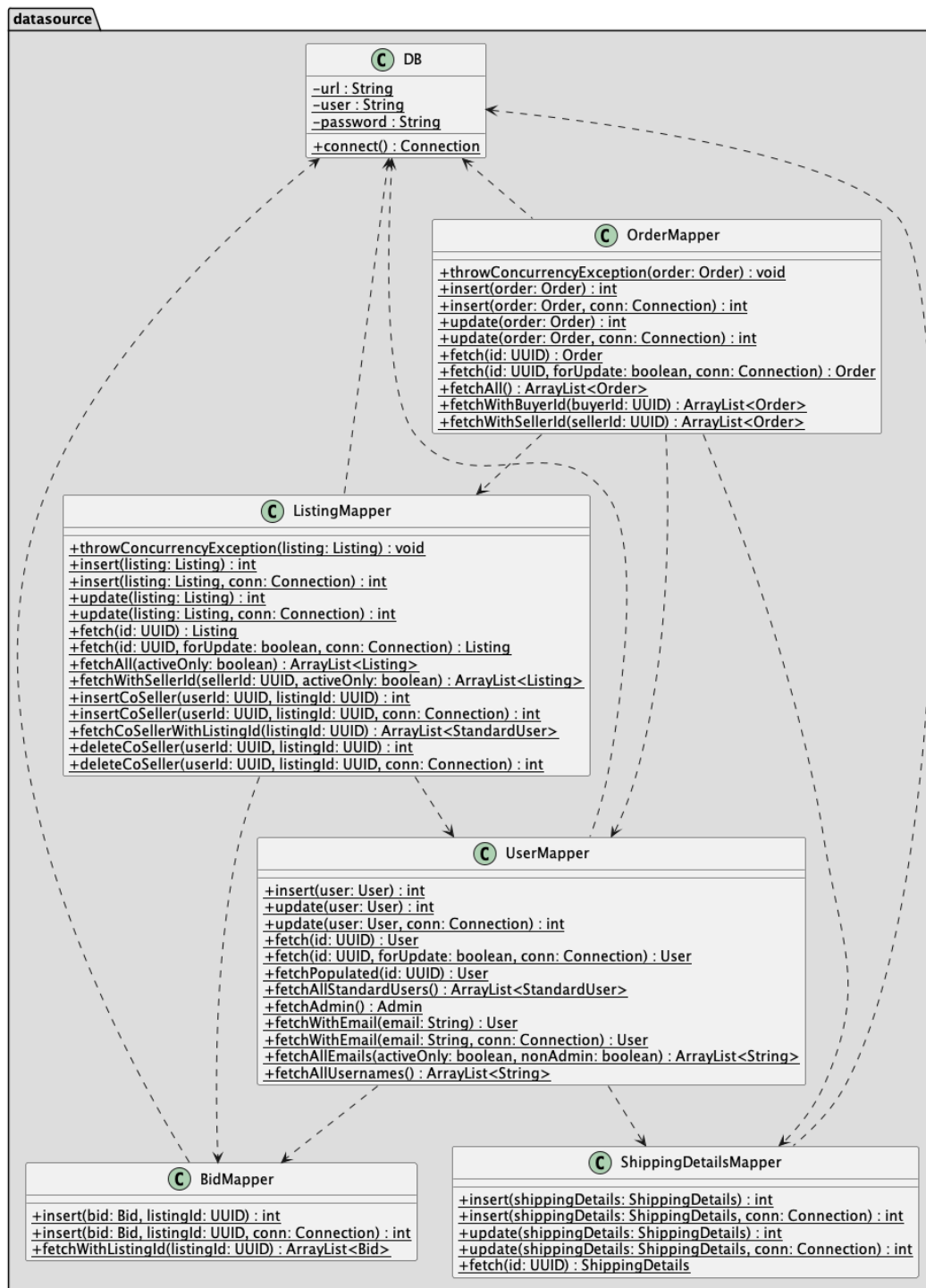


Figure 2 - Class Diagram of the datasource Package



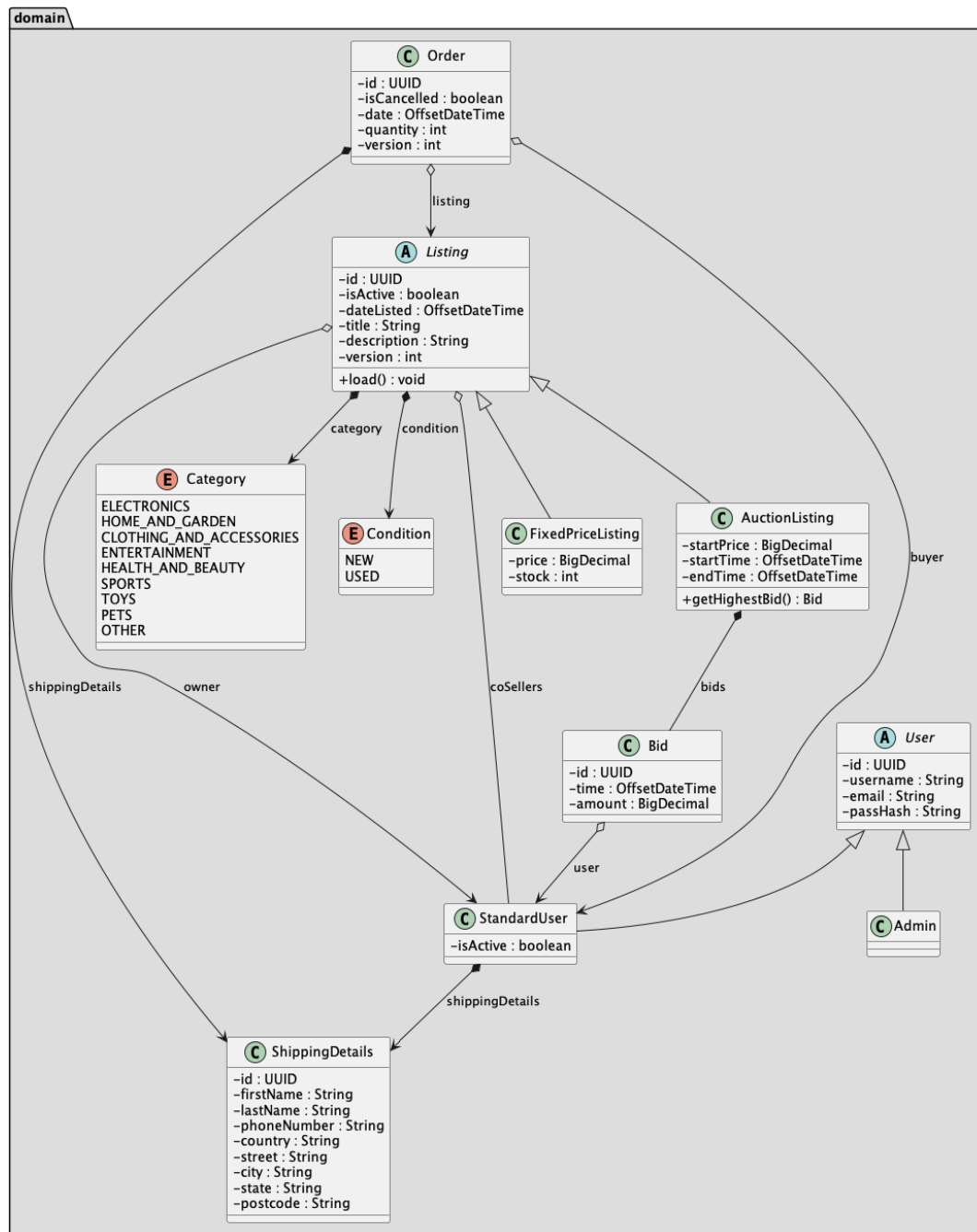


Figure 3 - Class Diagram of the domain Package

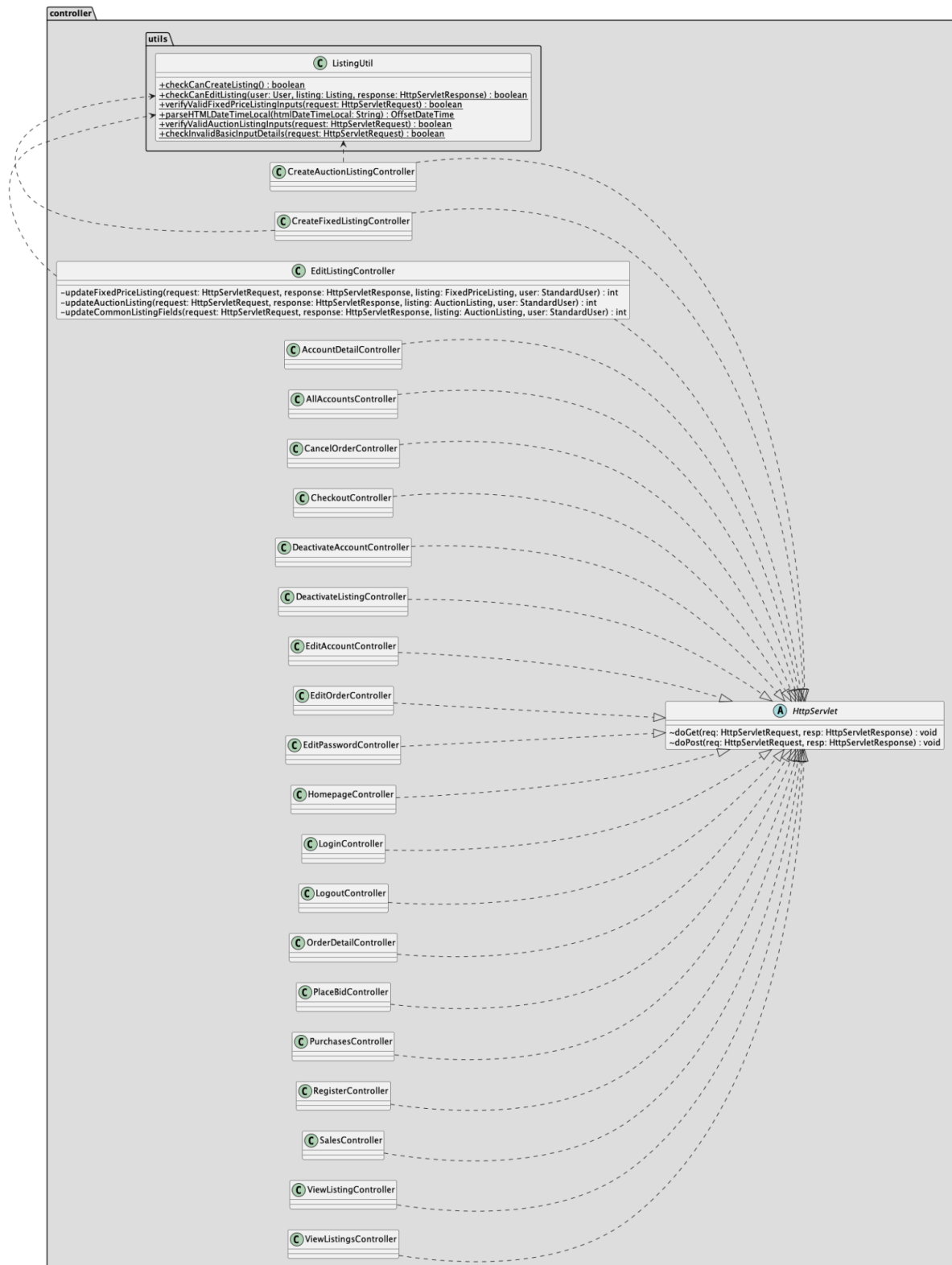


Figure 4 - Class Diagram of the controller Package

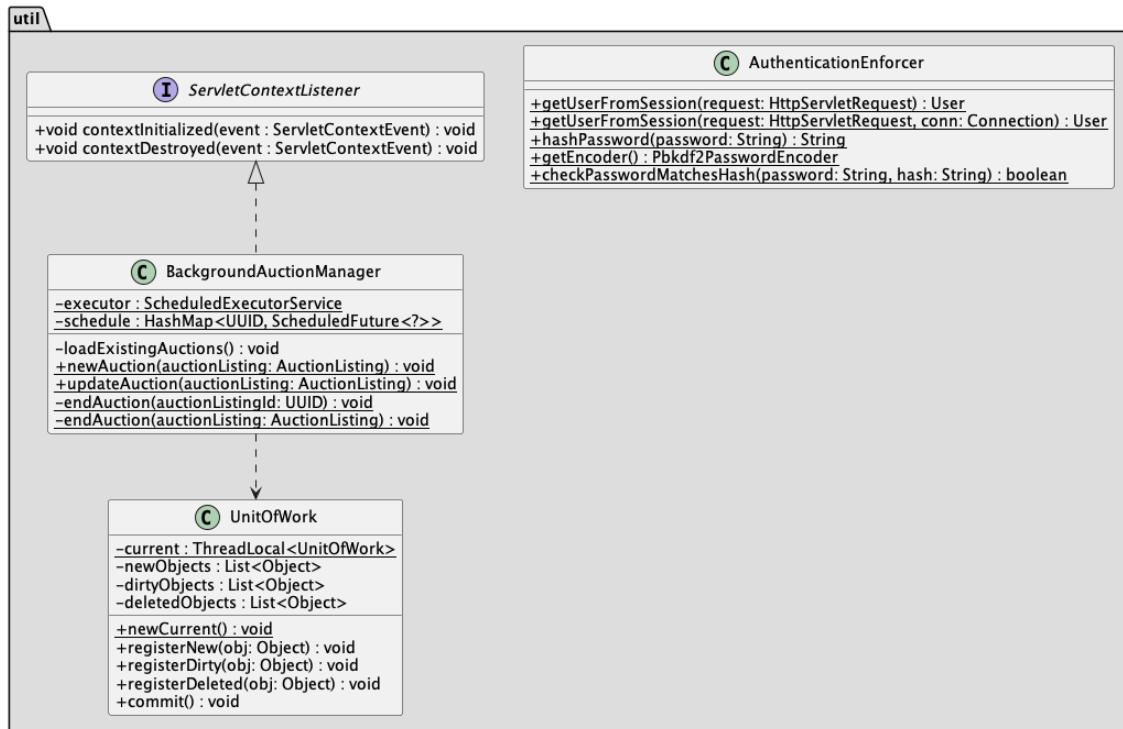


Figure 5 - Class Diagram of the util Package

## 3 Implemented concurrency patterns

### 3.1 Optimistic lock

As the name of this locking strategy suggests, this offline locking strategy is optimistic that the row data which has been read, has not been modified by something else in the system. This is achieved by noting the version of the data which has been read and incrementing the version once it is written back to the database.

Once the row in the database is read, the resulting object is free to be modified without having to connect back to the database and check whether or not the data is up-to-date. This allows multiple transactions to access the same version of row data at the same time.

However, when the data is written back to the database, this locking strategy must check that it is writing back to the same version it came from, if it is found that the version is different to the one when it was fetched, then a conflict is detected and the transaction is rolled back. This results in any work done in modifying the data is lost and the user must start fetch the latest version of the database and start again.

This locking strategy is simple and allows for good liveness of the system. This is because only one additional *version* column needs to be added to each database table where the optimistic lock is implemented. Detecting conflicts is just a matter for checking if the version held offline is the same as the one in the database at commit time. However when conflict does occur, all modifications that were made will be lost and the user must start again. This can be frustrating for the user but does not have such a big impact in a marketplace system. There are only a limited amount of modifications which can be made per listing and order, so while the user might not appreciate writing the same data again, the tradeoff in simplicity and liveness is justified.

#### 3.1.1 Optimistic lock on Listings (when modifying listing details)

The optimistic lock was implemented to combat any conflicts in listing details that can arise when the same listing is being modified by different users. The system allows for multiple users to be assigned as co-sellers for a single listing, and there can be situations where both co-sellers are making changes to the listing at the same time. In order to ensure that the co-seller is modifying the most recent listing details, the optimistic lock was chosen so that when there are inconsistencies between listing versions on commit, the co-seller is forced to fetch the latest version and start again.

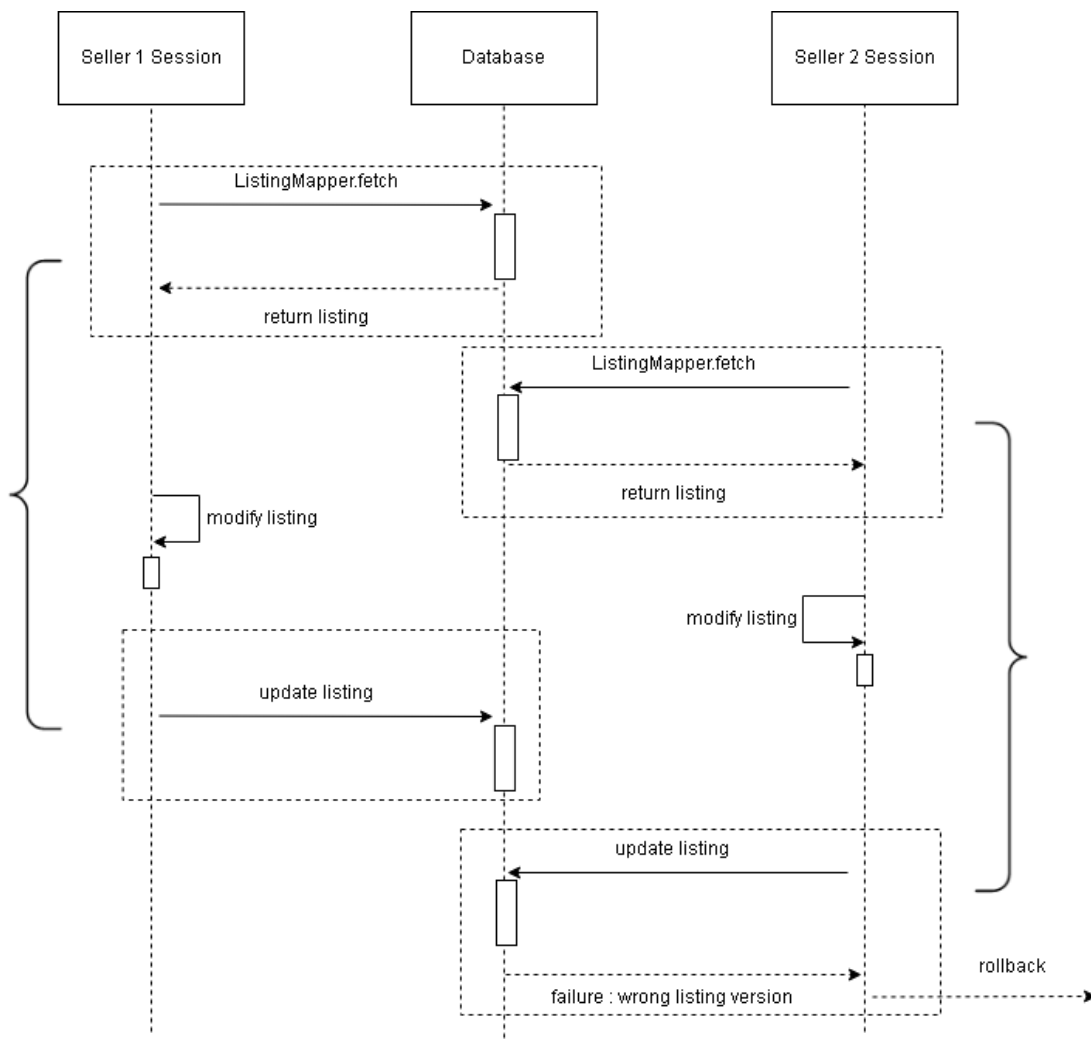


Figure 6 - Sequence diagram demonstrating optimistic locking on listings

### 3.1.2 Optimistic lock on Orders

In similar fashion to listings, the optimistic lock was also chosen for orders. However, for orders, buyers can also make changes to its details. By introducing an additional user that can make modifications on the order row in the database, this heightens the likelihood that the same order is being modified by multiple users. Therefore, the optimistic lock is well suited for the order details as the implementation is very similar to 3.1.1. and maintaining a connection to the database to ensure that an order is always up to date when making changes is impracticable.

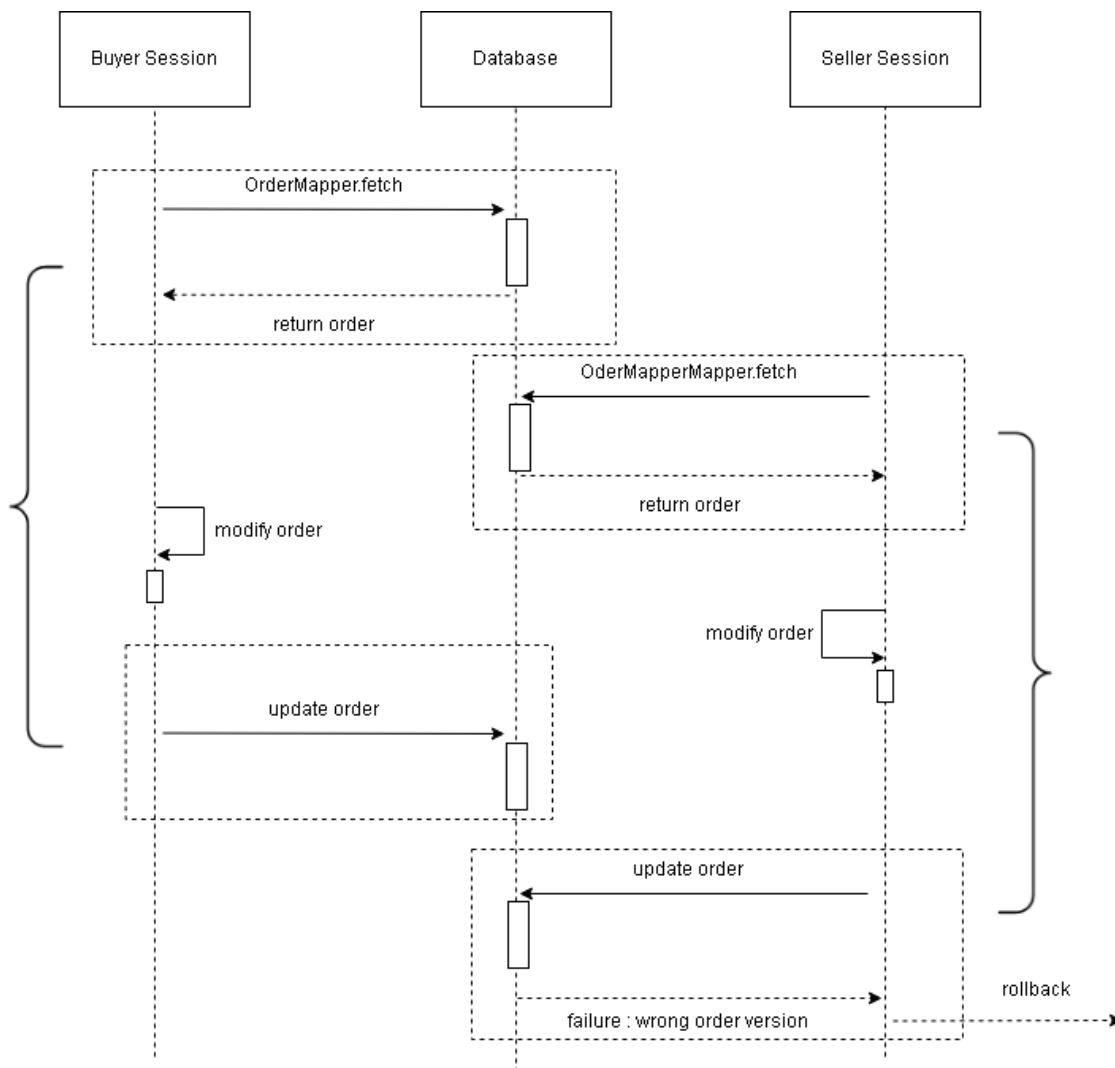


Figure 7 - Sequence diagram demonstrating optimistic locking on orders

### 3.2 Read lock

The read lock is traditionally a type of database locking mechanism that allows multiple threads to acquire a read lock on a row and prevents any other threads from acquiring write lock on the same row. This results in threads having the ability to all read data from the same row but blocks others from updating, or deleting it.

This solution is simple and reasonable for a system that would not expect any or very few updates to be made on the row while it is being read. However this solution sacrifices liveness and is not reasonable for a system such as a marketplace that can hold auctions where multiple users can participate in bidding for the same item or when an admin deactivates a user while that user is editing their details.

### 3.2.1 Auction Listing

For a marketplace system, the auction listing must always update to hold the latest highest bid but should not prevent other users from placing valid bids if they are greater than the current bid. Therefore the auction listing row must expect that the row can be updated by multiple users, and queue any concurrent valid bids. Allowing one to acquire a read lock on the row, updating it to the latest bid and then allowing the next bid to acquire the read lock and load the auction listing with the latest bid.

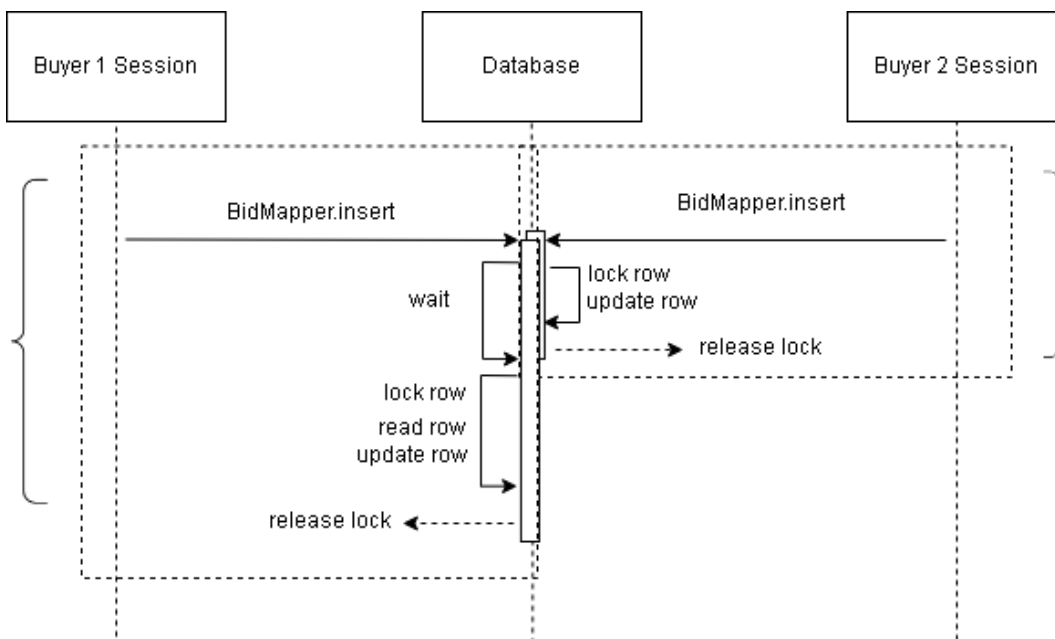


Figure 8 - Sequence Diagram demonstrating read locking on auction listings

This solution solves the problem of preventing other users from participating in live auctions by allowing concurrent valid bids from different users to be made on the same listing, maximising liveness of the system in place for a slightly more complicated solution in locking mechanism.

### 3.2.2 User

When a user updates their details, usually there isn't a need for a sophisticated locking mechanism to be made, as they are the only ones updating their details at a single point time. However, there needs to be in the marketplace system, as an admin can also make changes to user details by deactivating them if they are selling inappropriate items.

This is a result of our implementation of "deleting" data in the system - when an item or user is removed from the system, it doesn't actually delete the corresponding row in the table, instead it updates a flag in the table row as deactivated.

This is however where a locking mechanism needs to be implemented, as an admin can deactivate the user when the user is updating their details. If a read lock isn't acquired by the admin and they complete the action of deactivating a user, and at the same time a user can be

updating their details, once the user is done with updating their details, this could result in a lost update by the admin and the user row will be overwritten with the data acquired by the user, nullifying their deactivation.

To remedy this, the read lock was implemented so that if the situation mentioned above arises, the user will not commit their update until they acquire the lock that is being held by the admin. Once the lock is acquired by the user to commit their update, the row will be read again and they will find their row is now deactivated. This prevents the user from then updating their row and permanently removing their access from the system.

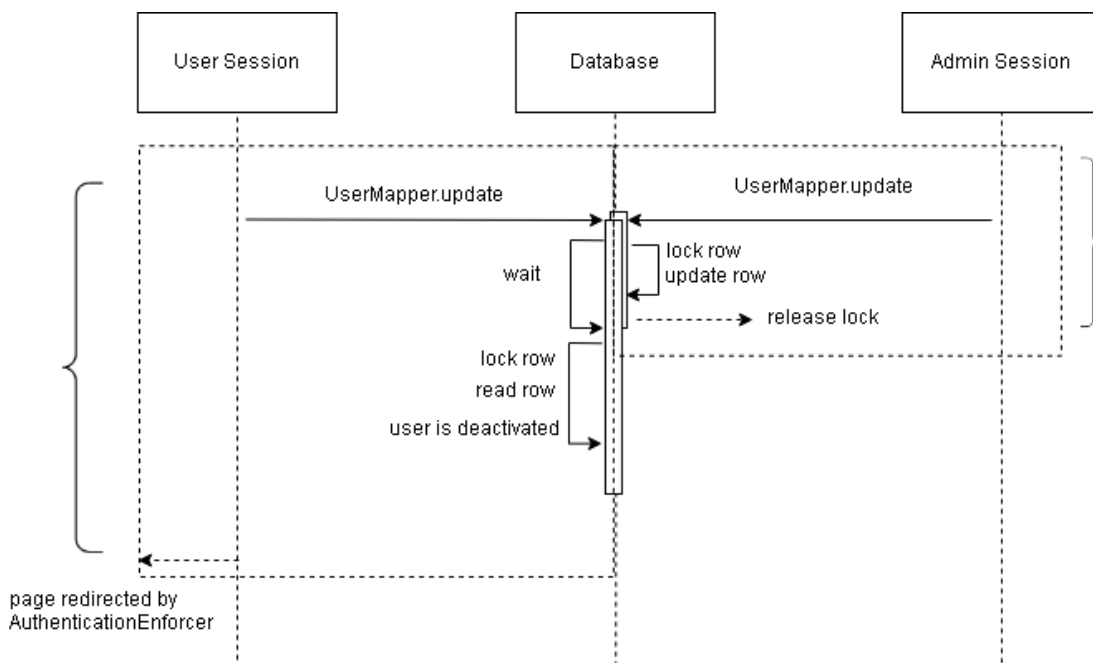


Figure 9 - Sequence Diagram demonstrating read locking on users

## 4 Concurrency issues by Use Case

The team has identified several concurrency issues that might occur within the Agora Marketplace system. The concurrency issues are categorised by their use case for the system. Implemented solutions for solving identified concurrency issues are discussed in the implemented concurrency pattern section (section 3 of this document).

Note that each issue is numbered in the format <use case number>.<issue number>; hence, use case numbers that do not have corresponding concurrency issues will be skipped.

### 4.1 Use Case 5: Manage listings

Based on the system's requirements, one listing can be controlled by the seller, multiple co-sellers and the admin. Therefore, the scenario of multiple users taking any actions on one



listing simultaneously might happen. Four concurrency issues have been identified for this use case.

#### 4.1.1 Issue 5.1

**Description:** A seller can try to modify a listing while another user is purchasing goods on the listing.

**Proposed Solution Pattern:** Optimistic lock on the listing object

**Rationale:** For modifying a listing, the version number on the listing object is expected to be identical to the number fetched during the initial GET request. This is so that details cannot change unexpectedly when a user tries to make a purchase. When purchasing an order, however, the listing is re-fetched at the beginning of the POST request, and the version number from that call is used, rather than the version number from the GET request. This is because it is desirable for multiple users to view an item they want to buy at the same time, and be able to perform that action. As such, due to the limited overlap in time and desire for liveness on listing-related pages, an optimistic lock is used to retain liveness.

#### 4.1.2 Issue 5.2

**Description:** The admin or a seller can deactivate a listing while a customer is trying to purchase goods on the listing.

**Proposed Solution Pattern:** Optimistic lock on the listing object

**Rationale:** A purchase on a listing should not be able to go through for a deactivated listing; hence, an optimistic lock prevents a purchase should the deactivation go through first. Conversely, because the action of purchasing a listing happens almost instantaneously (and the listing is refetched at the beginning of the checkout POST operation), a loss of liveness when deactivating a listing is unlikely to occur. The POST operation on checkout also rejects the purchase if the version loaded at the beginning of the operation is no longer active, making the operation safe.

#### 4.1.3 Issue 5.3

**Description:** Multiple co-sellers can try to modify a listing simultaneously.

**Proposed Solution Pattern:** Optimistic lock on the listing object

**Rationale:** Since updating a listing uses the version number at the time the page is initially loaded to verify no change in the intervening time, if two or more users attempt to modify that same version of the listing at the same time, only one attempt can succeed. This ensures that the second submitting user cannot overwrite the first user's changes without being aware of them first. However, the optimistic lock is still required and preferred over a pessimistic lock to preserve liveness for users attempting to purchase goods on the listing.

#### 4.1.4 Issue 5.4

**Description:** A seller can try to modify the listing at the same time as the admin deactivates the listing.

**Proposed Solution Pattern:** Optimistic lock on the listing object

**Rationale:** The rationale for this issue closely matches the rationale in section 4.1.2, except that instead of a user making a purchase, it is now a user modifying the listing details. The likelihood that the admin will deactivate the listing at the same time as the listing is updated is small, and therefore a potential rejection of the action is tolerable. Also, because deactivation will update the listing version number, it will not be possible for the editing user to complete their change as the locking mechanism will then reject.

## 4.2 Use Case 7: Manage orders

Similar to use case 5 on managing listings, orders can be managed by the seller, co-sellers, admin and buyers. Three concurrency issues have been identified.

### 4.2.1 Issue 7.1

**Description:** A user can try to modify an order on a listing while another user is purchasing the same good(s).

**Proposed Solution Pattern:** Optimistic lock on the listing object

**Rationale:** When modifying an order for a fixed-price listing, the user has the ability to change their desired quantity. As such, the stock remaining on the listing will also need to be updated as a reflection of this. Likewise, a purchase inherently requires changing the stock on the listing by some amount. As discussed previously, due to the near-instantaneous nature of placing or updating an order, optimistic locking is tolerable as it is more important to maintain liveness on the site than to prevent an extremely unlikely event from occurring.

### 4.2.2 Issue 7.2

**Description:** A user can try to modify an order at the same time as another user cancels the order.

**Proposed Solution Pattern:** Optimistic lock on the order object

**Rationale:** The rationale is similar for the situation with listings (refer to section 4.1.4). Modifying an order should not be possible once a listing has been deactivated, which should be prevented by the updated version number. While conflicting operations could cause a rejection of the deactivation attempt, this situation is highly unlikely as the deactivation POST requests loads the object at the beginning of the requests, meaning that it is an extremely small amount of time that the two operations can conflict. Also, since an order can only be modified by the sellers, the buyer, or the admin, there is even less chance of the two operations being performed at the same time than with the operations which conflict with relation to listings.

### 4.2.3 Issue 7.3

**Description:** A user can try to modify the order at the same time as the admin or seller deactivates the listing.

**Proposed Solution Pattern:** Optimistic lock on the listing object

**Rationale:** Deactivating a listing does not cancel orders; however, it does make orders uneditable. As such, if a listing is deactivated, it is tolerable for the update to the order to fail due to optimistic locking. As with previous discussions around deactivation, it is also tolerable to use optimistic locking as the chance of conflict with another operation on the order is likely to be negligible. Using optimistic locking enables liveness of any pages which reference the order, which can be viewed by multiple users, often as part of generalised pages.

## 4.3 Use Case 8: Bid on auction

Based on the requirements of the system, one concurrency issue has been identified.

### 4.3.1 Issue 8.1

**Description:** Multiple users can attempt to place a bid on the same listing simultaneously.

**Proposed Solution Pattern:** Read lock on listing object

**Rationale:** Placing a bid does not update the related listing, hence optimistic locking is ineffective in this instance (the version number will remain the same). Since bids are stored as separate objects which reference listings, there is also the issue that using an optimistic lock allows for two simultaneous attempts to place a bid to consider only the previously placed bids, and be considered the highest, hence allowing a potential lower-amount bid to go through after the higher amount. By using a read lock, bids must wait for previous bids to be placed; in this way, they can be sure that the highest bid in the database will be the highest value to compare to, and hence pass or fail accordingly.

## 4.4 Use Case 9: Purchase a fixed price good

Based on the requirements of the system, one concurrency issue has been identified.

### 4.4.1 Issue 9.1

**Description:** Multiple users can try to purchase some stock of a good which in the combined total quantity selected exceeds the available stock.

**Proposed Solution Pattern:** Optimistic lock on the listing object

**Rationale:** Due to the rapid operation time of placing an order (as previously discussed) and the unlikely event that two of such operations will occur at once, it is permissible to use an optimistic lock in this situation. The danger of two orders being allowed to go through at once, from the same version of the listing, is that their combined total checkout quantity exceeds the available stock. Preventing simultaneous updates on the same version of the listing object naturally prevents this from occurring.

## 4.5 Use Case 10: Manage all accounts

Since the admin can manage all accounts, concurrency issues may occur while a user is trying to perform actions while the admin is attempting to deactivate them. Two concurrency issues have been identified.

### 4.5.1 Issue 10.1

**Description:** The admin tries to deactivate a user while they are interacting with the system.

**Proposed Solution Pattern:** Authentication controller + Read lock on the user row

**Rationale:** From part 2 of the project, an authentication enforcer has been implemented which automatically logs out a deactivated user whenever they try to perform an action that requires authentication. This pattern has been expanded to make use of a read lock, as has deactivation of the user by the admin. It is crucial that deactivation of a user be allowed to succeed, so that they can no longer interact with the system. Hence, a read lock will ensure that the user deactivation always succeeds. Furthermore, it is not likely that there will be liveness issues with this. As the authentication enforcer only performs actions instantaneously at the start of a request, it does not lock a user for any detrimental amount of time, meaning that there should be no delay in their deactivation.

### 4.5.2 Issue 10.2

**Description:** The admin deactivates a user while the user is updating their account details.

**Proposed Solution Pattern:** Read lock on user row

**Rationale:** If a user tries to update their details, it will include the attribute “is\_active”, which for them would be “true” - this is the value that is set to “false” when they are deactivated. By using a read lock, deactivation can be allowed to occur before the user attempts to update their details; then, as described in section 4.5.1, the authentication enforcer will log out the deactivated user.

### 4.5.3 Issue 10.3

**Description:** The admin deactivates a user while another user is updating a listing or order they are tied to (i.e., as the owning seller or a co-seller).

**Proposed Solution Pattern:** Optimistic lock on listing + Optimistic lock on orders + Read lock on user row

**Rationale:** It is imperative that a user who is deactivated can no longer interact with the system in such a way as to make any changes to any data on the site. As discussed previously, locking the user row in the database will prevent the user from gaining authentication to perform an action while they are attempting to make an action. However, optimistic locking on the listings and orders associated with the user is permissible. While the system is designed to cancel orders and deactivate listings for the user when they are deactivated, should these fail due to some other update going through at the same time, this is not so urgent - the user will be unable to edit them once they have been deactivated anyway, leaving the admin free to deactivate those items manually.

## 5 Test Strategy

The overall testing strategy for testing the concurrency problem requires sending multiple requests to the system using Jmeter and verifying the expected outcome from the database. The team decided to use this strategy by following the demonstration from the testing performance of this subject. For some more complex test cases, manual testing was performed due to the limitations of Jmeter and the system's design. More detailed information is provided in section 5 for each test case. Test cases are designed based on boundary testing, which means the concurrency issues on point would be 2 requests happening simultaneously; therefore, all test cases are designed with 2 requests and executed once to catch the potential small concurrency issue. Data for listing and order are not going to affect the performance of the system, therefore in some of the testing listings, users are named as testing just to identify the test result more clearly.

### 5.1 Automation testing tool

The team decided to use *Jmeter* for testing the concurrency problems. The idea for choosing a tool to test concurrency problems is it can send multiple requests to the system simultaneously and simulate real use of the system. Jmeter is an open-sourced testing tool, able to create multiple thread groups, send multiple requests simultaneously, and set the loop for the test request. Therefore, using Jmeter can verify the solution for concurrency problems by sending multiple requests simultaneously and demonstrate the system's robustness by testing the same function for a certain amount of time. Moreover, Jmeter supports performance testing for the product, so using it will benefit part 4 of this project.

### 5.2 Manual testing

Because of the design of the auction system, there is no endpoint available to force an end to an auction, and the sellers cannot edit the listing further. Therefore, a test cannot be written in JMeter where a request to place a bid occurs at the exact same moment an auction ends. Instead, the team has opted to perform manual testing for this particular case.

### 5.3 Test cases

Use case	Identified concurrency issue	Test case
Use Case 5: Manage listings	1. A seller can try to modify a listing while another user is purchasing goods on the listing	TC5.1 Seller tries to update the listing while the seller is purchasing 10 items on the same listing simultaneously.

Use case	Identified concurrency issue	Test case
	2. The admin or a seller can deactivate a listing while a customer is trying to purchase goods on the listing	TC5.2 admin try to deactivate the listing while the buyer is trying to purchase 10 items from the same listing
	3. Multiple co-sellers can try to modify a listing simultaneously	TC5.3 multiple co-sellers try to modify the listing simultaneously
	4. A seller can try to modify the listing at the same time as the admin deactivates the listing	TC5.4 Seller can modify the listing at the same time as the admin deactivate the listing
Use Case 7: Manage orders	1. A user can try to modify an order on a listing while another user is purchasing the same good(s)	TC.7.1.1 seller try to modify the order for a listing while another user is purchasing the same good(s) TC.7.1.2 buyer try to modify the order for a listing while another user is purchasing the same good(s)
	2. A user can try to modify an order at the same time as another user cancels the order	TC7.2 seller try to modify the order at the same time as buyer canceling the order
	3. A user can try to modify the order at the same time as the admin or seller deactivates the listing	TC7.3 seller try to modify the order at the same time as the admin deactivate the listing
Use Case 8: Bid on auction	1. Multiple users can attempt to place a bid on the same listing simultaneously	TC8.1.1 2 users attempt to place the same amount bid on the same listing simultaneously TC8.1.2 2 users attempt to place bid on different amount

Use case	Identified concurrency issue	Test case
		on the same listing simultaneously
Use Case 9: Purchase a fixed price good	1. Multiple users can try to purchase some stock of a good which in the combined total quantity selected exceeds the available stock	TC9.1 2 users try to purchase 50 items on a listing with total stock of 100
Use Case 10: Manage all accounts	1. The admin tries to deactivate a user while they are interacting with the system	TC10.1 Admin tries to deactivate a user when the user is logged in to the system
	2. The admin deactivates a user while the user is updating their account details	TC10.2 Admin tries to deactivate a user when the user is updating the account detail
	3. The admin deactivates a user while another user is updating a listing or order they are tied to (i.e., as the owning seller or a co-seller)	TC10.3 admin deactivate a user while seller is updating the listing the user is one of the co-seller

### 5.3 Testing setups

The instructions to setup the testing environment are as follows:

1. Install Jmeter ([https://jmeter.apache.org/download\\_jmeter.cgi](https://jmeter.apache.org/download_jmeter.cgi))
2. Download the test plans from the SWEN90007-2022-Agora GitHub page; they can be found in docs/testing (<https://github.com/SWEN900072022/SWEN90007-2022-Agora/tags>)

### 5.4 Test outcomes

Test outcomes contain the test type, execution type, objective of the testing, setup for the automation/manual testing, pre-conditions, expected outcome, result, and notes which include the proof of sending simultaneous requests from Jmeter.







TC 5.1 Seller tries to update the listing while the seller is purchasing 10 items on the same listing simultaneously.

<b>Test Type:</b> Functional	<b>Execution Type:</b> Automated testing																														
<b>Objective:</b> <ul style="list-style-type: none"><li>- Ensure data consistent and liveness for the system</li><li>- Seller has no problem with updating listing</li><li>- Buyer can successfully place an order</li></ul>																															
<b>Setup:</b> <ul style="list-style-type: none"><li>- Open TC5.1.jmx</li><li>- Run the test plan</li></ul>																															
<b>Pre-Conditions:</b> <ul style="list-style-type: none"><li>- Seller is active in the system</li><li>- Buyer is active in the system</li><li>- One fixed price listing has stock and active in the system</li></ul>																															
<b>Expected outcome:</b> <ul style="list-style-type: none"><li>- Listing is updated by seller</li><li>- New order with 10 items is created for the same listing</li><li>- Stock of listing reduce by 10</li><li>- Version of listing +2</li></ul>																															
<b>Result: PASS</b>																															
<b>Note:</b>																															
<table><tr><th></th><th>Start Time</th><th>Thread Name</th><th>Label</th><th>Sample Time(ms)</th><th>Status</th></tr><tr><td>1</td><td>19:46:50.350</td><td>Thread Group 1-1</td><td>/login</td><td>4863</td><td></td></tr><tr><td>2</td><td>19:46:55.225</td><td>Thread Group 1-1</td><td>get listing</td><td>570</td><td></td></tr><tr><td>3</td><td>19:46:55.797</td><td>Thread Group 1-1</td><td>/checkout</td><td>1680</td><td></td></tr><tr><td>4</td><td>19:46:50.262</td><td>Thread Group 1-1</td><td>buyer purchase listing</td><td>7113</td><td></td></tr></table>			Start Time	Thread Name	Label	Sample Time(ms)	Status	1	19:46:50.350	Thread Group 1-1	/login	4863		2	19:46:55.225	Thread Group 1-1	get listing	570		3	19:46:55.797	Thread Group 1-1	/checkout	1680		4	19:46:50.262	Thread Group 1-1	buyer purchase listing	7113	
	Start Time	Thread Name	Label	Sample Time(ms)	Status																										
1	19:46:50.350	Thread Group 1-1	/login	4863																											
2	19:46:55.225	Thread Group 1-1	get listing	570																											
3	19:46:55.797	Thread Group 1-1	/checkout	1680																											
4	19:46:50.262	Thread Group 1-1	buyer purchase listing	7113																											
<table><tr><th>File #</th><th>Start Time</th><th>Thread Name</th><th>Label</th><th>Sample Time(ms)</th><th>Status</th></tr><tr><td>1</td><td>19:46:50.350</td><td>Thread1 2-1</td><td>/login</td><td>2829</td><td></td></tr><tr><td>2</td><td>19:46:53.227</td><td>Thread1 2-1</td><td>https://swen90007-a...</td><td>2176</td><td></td></tr><tr><td>3</td><td>19:46:55.414</td><td>Thread1 2-1</td><td>/edit-listing</td><td>1091</td><td></td></tr><tr><td>4</td><td>19:46:50.262</td><td>Thread1 2-1</td><td>seller modify listing</td><td>6246</td><td></td></tr></table>		File #	Start Time	Thread Name	Label	Sample Time(ms)	Status	1	19:46:50.350	Thread1 2-1	/login	2829		2	19:46:53.227	Thread1 2-1	https://swen90007-a...	2176		3	19:46:55.414	Thread1 2-1	/edit-listing	1091		4	19:46:50.262	Thread1 2-1	seller modify listing	6246	
File #	Start Time	Thread Name	Label	Sample Time(ms)	Status																										
1	19:46:50.350	Thread1 2-1	/login	2829																											
2	19:46:53.227	Thread1 2-1	https://swen90007-a...	2176																											
3	19:46:55.414	Thread1 2-1	/edit-listing	1091																											
4	19:46:50.262	Thread1 2-1	seller modify listing	6246																											

TC5.2 admin try to deactivate the listing while the buyer is trying to purchase 10 items from the same listing

<b>Test Type:</b> Functional	<b>Execution Type:</b> Automated testing
<b>Objective:</b> <ul style="list-style-type: none"> <li>- Remain liveness for the system</li> <li>- Admin have more control over than regular user</li> </ul>	
<b>Setup:</b>	



<ul style="list-style-type: none"> <li>- Open TC5.2.jmx</li> <li>- Run the test plan</li> </ul>					
<b>Pre-Conditions:</b> <ul style="list-style-type: none"> <li>- Admin account exists in the system</li> <li>- Seller is active in the system</li> <li>- Buyer is active in the system</li> <li>- One fixed price listing has stock and active in the system</li> </ul>					
<b>Expected outcome:</b> <ol style="list-style-type: none"> <li>1) An order is created</li> </ol>					
<b>Result</b>					
<b>Note:</b>					
	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	20:02:58.626	Thread Group 2-1	/login	2804	
2	20:03:01.449	Thread Group 2-1	/deactivate-listing	1669	
3	20:02:58.557	Thread Group 2-1	admin deactivate a list...	4473	
	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	20:02:58.626	Thread Group 1-1	/login	2844	
2	20:03:01.482	Thread Group 1-1	/checkout	745	
3	20:02:58.558	Thread Group 1-1	buyer purchase same ...	3589	

TC5.3 multiple co-sellers try to modify the listing simultaneously

<b>Test Type:</b> Functional	<b>Execution Type:</b> Automated testing
<b>Objective:</b> <ul style="list-style-type: none"> <li>- Stay data consistent</li> <li>- Listing can be update only once</li> </ul>	
<b>Setup:</b> <ul style="list-style-type: none"> <li>- Open TC5.3.jmx</li> <li>- Run the test plan</li> </ul>	
<b>Pre-Conditions:</b> <ul style="list-style-type: none"> <li>- Co-sellers are active in the system</li> <li>- Seller is active in the system</li> <li>- One fixed price listing has stock and active in the system</li> </ul>	
<b>Expected outcome:</b> <ul style="list-style-type: none"> <li>- Only one request pass through to the database</li> <li>- Listing version number +1</li> </ul>	
<b>Result:</b> Pass	

**Note:**

	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	21:40:14.437	Thread Group 1-1	/login	3084	✓
2	21:40:17.525	Thread Group 1-1	edit-listing	2063	✓
3	21:40:19.598	Thread Group 1-1	/edit-listing	1220	✓
4	21:40:14.383	Thread Group 1-1	Co-seller1 modify li...	6367	✓
-					
	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	21:40:14.437	Thread Group 2-1	/login	3006	✓
2	21:40:17.465	Thread Group 2-1	edit-listing	2128	✓
3	21:40:19.598	Thread Group 2-1	/edit-listing	1248	✓
4	21:40:14.383	Thread Group 2-1	Co-seller2 modify li...	6382	✓
-					

TC5.4 Seller can modify the listing at the same time as the admin deactivate the listing

Test Type:

Functional

Execution Type:

Automated testing

Objective:

- Ensure the liveness for the system

- Admin can deactivate the listing anytime

Setup:

- Open TC5.4.jmx

- Run the test plan

Pre-Conditions:

- Admin account exists in the system

- Co-sellers are active in the system

- Seller is active in the system

- One fixed price listing has stock and active in the system

Expected outcome:

- No error message found

- Listing get deactivate

- Listing detail get updated

- Version of listing +2

Result:

PASS

Note:

	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	22:09:02.906	Thread Group 2-1	/login	2692	
2	22:09:05.629	Thread Group 2-1	get listing	3141	
3	22:09:08.781	Thread Group 2-1	/deactivate-listing	2357	
4	22:09:02.798	Thread Group 2-1	admin deactivate list...	8190	

	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	22:09:02.906	Thread Group 1-1	/login	2917	
2	22:09:05.838	Thread Group 1-1	get listing	2001	
3	22:09:07.846	Thread Group 1-1	/edit-listing	913	
4	22:09:02.798	Thread Group 1-1	seller modify listing	5831	

- Step 4 for both thread in Jmeter indicate the request happened simultaneously, step 1-3 are the set-up for the request due to the design of the system.

TC.7.1.1 seller tries to modify the order for a listing while another user is purchasing the same good(s)

<b>Test Type:</b> Functional	<b>Execution Type:</b> Automated testing																																																												
<b>Objective:</b> <ul style="list-style-type: none"><li>- Maintain the liveness of the system</li><li>- Buyer can place a new order</li><li>- Seller can update existed order</li></ul>																																																													
<b>Setup:</b> <ul style="list-style-type: none"><li>- Open TC7.1.1.jmx</li><li>- Run the test plan</li></ul>																																																													
<b>Pre-Conditions:</b> <ul style="list-style-type: none"><li>- Co-sellers are active in the system</li><li>- Seller is active in the system</li><li>- One fixed price listing has stock and active in the system</li><li>- One order exists for the testing listing</li></ul>																																																													
<b>Expected outcome:</b> <ul style="list-style-type: none"><li>- When new buyer try to purchase 30 listing stock -30, new order created, listing version +1</li><li>- When seller try to increase order from 10 to 20, listing stock -10, order version +1, listing version +1</li><li>- Both requests should go through and version of listing +2</li></ul>																																																													
<b>Result: PASS</b>																																																													
<b>Note:</b>																																																													
<table><tr><th></th><th>Start Time</th><th>Thread Name</th><th>Label</th><th>Sample Time(ms)</th><th>Status</th></tr><tr><td>1</td><td>17:04:34.158</td><td>Thread Group 2-1</td><td>/login</td><td>3191</td><td></td></tr><tr><td>2</td><td>17:04:37.356</td><td>Thread Group 2-1</td><td>get listing</td><td>590</td><td></td></tr><tr><td>3</td><td>17:04:37.948</td><td>Thread Group 2-1</td><td>/checkout</td><td>1512</td><td></td></tr><tr><td>4</td><td>17:04:34.116</td><td>Thread Group 2-1</td><td>buyer purchase listing</td><td>5293</td><td></td></tr></table> <table><tr><th></th><th>Start Time</th><th>Thread Name</th><th>Label</th><th>Sample Time(ms)</th><th>Status</th></tr><tr><td>1</td><td>17:04:34.158</td><td>Thread Group 1-1</td><td>/login</td><td>2966</td><td></td></tr><tr><td>2</td><td>17:04:37.152</td><td>Thread Group 1-1</td><td>edit-listing</td><td>2535</td><td></td></tr><tr><td>3</td><td>17:04:39.716</td><td>Thread Group 1-1</td><td>/edit-order</td><td>1895</td><td></td></tr><tr><td>4</td><td>17:04:34.116</td><td>Thread Group 1-1</td><td>seller modify order</td><td>7396</td><td></td></tr></table>			Start Time	Thread Name	Label	Sample Time(ms)	Status	1	17:04:34.158	Thread Group 2-1	/login	3191		2	17:04:37.356	Thread Group 2-1	get listing	590		3	17:04:37.948	Thread Group 2-1	/checkout	1512		4	17:04:34.116	Thread Group 2-1	buyer purchase listing	5293			Start Time	Thread Name	Label	Sample Time(ms)	Status	1	17:04:34.158	Thread Group 1-1	/login	2966		2	17:04:37.152	Thread Group 1-1	edit-listing	2535		3	17:04:39.716	Thread Group 1-1	/edit-order	1895		4	17:04:34.116	Thread Group 1-1	seller modify order	7396	
	Start Time	Thread Name	Label	Sample Time(ms)	Status																																																								
1	17:04:34.158	Thread Group 2-1	/login	3191																																																									
2	17:04:37.356	Thread Group 2-1	get listing	590																																																									
3	17:04:37.948	Thread Group 2-1	/checkout	1512																																																									
4	17:04:34.116	Thread Group 2-1	buyer purchase listing	5293																																																									
	Start Time	Thread Name	Label	Sample Time(ms)	Status																																																								
1	17:04:34.158	Thread Group 1-1	/login	2966																																																									
2	17:04:37.152	Thread Group 1-1	edit-listing	2535																																																									
3	17:04:39.716	Thread Group 1-1	/edit-order	1895																																																									
4	17:04:34.116	Thread Group 1-1	seller modify order	7396																																																									

TC.7.1.2 buyer try to modify the order for a listing while another user is purchasing the same good(s)

<b>Test Type:</b> Functional	<b>Execution Type:</b> Automated testing
<b>Objective:</b> <ul style="list-style-type: none"> <li>- No error message found for each request</li> </ul>	

<ul style="list-style-type: none"> <li>- Buyer can place a new order</li> <li>- Another buyer can update existed order</li> </ul>						
<b>Setup:</b>						
<ul style="list-style-type: none"> <li>- Open TC7.1.2.jmx</li> <li>- Run the test plan</li> </ul>						
<b>Pre-Conditions:</b>						
<ul style="list-style-type: none"> <li>- Both buyers are active in the system</li> <li>- Seller is active in the system</li> <li>- One fixed price listing has stock and active in the system</li> <li>- One order exists for the testing listing</li> </ul>						
<b>Expected outcome:</b>						
<ul style="list-style-type: none"> <li>- When new buyer try to purchase 15 items, listing stock -15, new order created, listing version +1</li> <li>- When buyer try to increase order from 20 to 40, listing stock -20, order version +1, listing version +1</li> <li>- If both request should go through and version of listing +2</li> </ul>						
<b>Result: PASS</b>						
<ul style="list-style-type: none"> <li>- Both thread went through without any errors</li> <li>- Buyer modify order got rolled back, existed order maintained the same</li> <li>- New order is created, and listing version got update once.</li> </ul>						
<b>Note:</b>						
	Start Time	Thread Name	Label	Sample Time(ms)	Status	
1	17:26:32.796	Thread Group 1-1	/login	3104	✓	
2	17:26:35.927	Thread Group 1-1	get listing	594	✓	
3	17:26:36.524	Thread Group 1-1	/checkout	1723	✓	
4	17:26:32.755	Thread Group 1-1	buyer purchase listing	5421	✓	
-						
	Start Time	Thread Name	Label	Sample Time(ms)	Status	
1	17:26:32.797	Thread Group 2-1	/login	3127	✓	
2	17:26:35.930	Thread Group 2-1	edit-order	3365	✓	
3	17:26:39.300	Thread Group 2-1	/edit-order	1460	✓	
4	17:26:32.755	Thread Group 2-1	another buyer modify ...	7952	✓	

TC7.2 seller tries to modify the order at the same time as buyer cancelling the order

<b>Test Type:</b>	<b>Execution Type:</b>
Functional	Automated testing
<b>Objective:</b>	
<ul style="list-style-type: none"> <li>- No error message found for each request</li> <li>- Version number for listing change based on the time of the request pass through</li> <li>- Listing stock changes according to the request pass through</li> </ul>	
<b>Setup:</b>	
<ul style="list-style-type: none"> <li>- Open TC7.2.jmx</li> <li>- Run the test plan</li> </ul>	

**Pre-Conditions:**

- Both buyers are active in the system
- Seller is active in the system
- One fixed price listing has stock and is active in the system
- One order exists for the testing listing

**Expected outcome:**

- When seller successfully increase the existing order from 15 to 20, stock of relevant listing reduce 5, version of listing +1
- When buyer successfully cancelled the order for 15 items, stock of relevant listing increase 15, version of listing +1, version of order +1.

**Result: PASS**

- Cancel order passed through
- Order and listing changed according to the expected outcome
- No error message

**Note:**

	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	17:48:27.164	Thread Group 2-1	/login	2780	✓
2	17:48:29.968	Thread Group 2-1	purchases	1265	✓
3	17:48:31.236	Thread Group 2-1	/cancel-order	1659	✓
4	17:48:27.121	Thread Group 2-1	buyer cancel the same...	5704	✓

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	17:48:27.164	Thread Group 1-1	/login	2780	✓
2	17:48:29.970	Thread Group 1-1	edit-order	2134	✓
3	17:48:32.111	Thread Group 1-1	/edit-order	1814	✓
4	17:48:27.122	Thread Group 1-1	seller modify the order	6728	✓

TC7.3 seller tries to modify the order at the same time as the admin deactivates the listing

<b>Test Type:</b> Functional	<b>Execution Type:</b> Automated testing
<b>Objective:</b> <ul style="list-style-type: none"> <li>- No error message found for each request</li> <li>- Version number for listing change based on the time of the request pass through</li> <li>- Listing stock changes according to the request pass through</li> </ul>	
<b>Setup:</b> <ul style="list-style-type: none"> <li>- Open TC7.3.jmx</li> <li>- Run the test plan</li> </ul>	
<b>Pre-Conditions:</b> <ul style="list-style-type: none"> <li>- Both buyers are active in the system</li> <li>- Seller is active in the system</li> <li>- One fixed price listing has stock and is active in the system</li> <li>- One order exists for the testing listing</li> </ul>	

**Expected outcome:**

- When seller successfully increase the existing order from 1 to 11, stock of listing -10, version of listing +1
- When admin successfully deactivate the listing, listing status become inactive, listing version +1
- Listing will be deactivate eventually but no guarantee on the modify listing

**Result: PASS**

- Cancel order passed through
- Order and listing changed according to expected outcome
- Listing version +2 and became inactive
- No error message

**Note:**

	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	18:20:54.096	Thread Group 2-1	/login	4628	✓
2	18:20:58.734	Thread Group 2-1	all listing	1699	✓
3	18:21:00.436	Thread Group 2-1	/deactivate-listing	1714	✓
4	18:20:54.016	Thread Group 2-1	admin deactivate the s...	8041	✓
-					
	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	18:20:54.095	Thread Group 1-1	/login	2767	✓
2	18:20:56.914	Thread Group 1-1	edit-order	2227	✓
3	18:20:59.158	Thread Group 1-1	/edit-order	1769	✓
4	18:20:54.016	Thread Group 1-1	seller modify the order	6763	✓
-					

TC8.1.1 2 users attempt to place the same amount bid on the same listing simultaneously

<b>Test Type:</b> Functional	<b>Execution Type:</b> Automated testing
<b>Objective:</b> <ul style="list-style-type: none"> <li>- No error message found for each request</li> <li>- Ensure the auction listing don't create the same bids</li> </ul>	
<b>Setup:</b> <ul style="list-style-type: none"> <li>- Open TC8.1.1.jmx</li> <li>- Run the test plan</li> </ul>	
<b>Pre-Conditions:</b> <ul style="list-style-type: none"> <li>- Both buyers are active in the system</li> <li>- Seller is active in the system</li> <li>- An auction listing is active in the system</li> </ul>	
<b>Expected outcome:</b> <ul style="list-style-type: none"> <li>- Only one bid go through the system</li> <li>- Unsuccessful bid get response with 400 as bad request</li> <li>- Only one new bid created in the database</li> <li>- Auction listing detail maintain the same</li> </ul>	
<b>Result: PASS</b>	

**Note:**

	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	18:53:35.353	Thread Group 1-1	/login	3613	✓
2	18:53:39.024	Thread Group 1-1	/place-bid	446	✗
3	18:53:35.285	Thread Group 1-1	buyer1 place a bid on...	4059	✗
-					
	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	18:53:35.353	Thread Group 2-1	/login	3613	✓
2	18:53:39.024	Thread Group 2-1	/place-bid	991	✓
3	18:53:35.285	Thread Group 2-1	buyer2 place a bid on...	4604	✓
-					

TC8.1.2 2 users attempt to place bid on different amount on the same listing simultaneously

<b>Test Type:</b> Functional	<b>Execution Type:</b> Automated testing
<b>Objective:</b> <ul style="list-style-type: none"> <li>- No error message found for each request</li> <li>- Ensure the auction listing don't create the same bids</li> </ul>	
<b>Setup:</b> <ul style="list-style-type: none"> <li>- Open TC8.1.2.jmx</li> <li>- Run the test plan</li> </ul>	
<b>Pre-Conditions:</b> <ul style="list-style-type: none"> <li>- Both buyers are active in the system</li> <li>- Seller is active in the system</li> <li>- An auction listing is active in the system</li> </ul>	
<b>Expected outcome:</b> 2 different outcomes are expected: <ol style="list-style-type: none"> <li>1) 2 bids are created in the system               <ul style="list-style-type: none"> <li>- Lower amount bid got created first and higher amount bib got create after lower one</li> </ul> </li> <li>2) Only one bid is created in the system               <ul style="list-style-type: none"> <li>- Higher amount bid got created first and lower bid get response with 400 as bad request</li> </ul> </li> </ol>	
<b>Result: PASS</b> <ul style="list-style-type: none"> <li>- Only one bid is created in the system</li> <li>- Higher amount bid got created first and lower bid get response with 400 as bad request</li> </ul>	

**Note:**

- Sent two requests with \$300 and another one \$400

	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	19:10:15.084	Thread Group 1-1	/login	2767	
2	19:10:17.900	Thread Group 1-1	/place-bid	975	
3	19:10:15.021	Thread Group 1-1	buyer1 place a bid on...	3742	

	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	19:10:15.084	Thread Group 2-1	/login	2768	
2	19:10:17.899	Thread Group 2-1	/place-bid	462	
3	19:10:15.021	Thread Group 2-1	buyer2 place a bid on...	3230	

TC9.1 2 users try to purchase 50 items each on a listing with total stock 100

<b>Test Type:</b> Functional	<b>Execution Type:</b> Automated testing																																																
<b>Objective:</b> <ul style="list-style-type: none"><li>- Ensure system livenss</li><li>- Data consisten</li></ul>																																																	
<b>Setup:</b> <ul style="list-style-type: none"><li>- Open TC9.1.jmx</li><li>- Run the test plan</li></ul>																																																	
<b>Pre-Conditions:</b> <ul style="list-style-type: none"><li>- Both buyers are active in the system</li><li>- Seller is active in the system</li><li>- A fixed price listing is active in the system</li></ul>																																																	
<b>Expected outcome:</b> <ul style="list-style-type: none"><li>- Only one purchase request pass through the system</li><li>- One order with 50 items is created in the database</li><li>- Listing version number +1</li><li>- Listing stock -50</li></ul>																																																	
<b>Result:</b> <b>PASS</b> <ul style="list-style-type: none"><li>- Only one order is created in the system</li><li>- Another order request get response with 404 as designed</li></ul>																																																	
<b>Note:</b> <table><tr><th></th><th>Start Time</th><th>Thread Name</th><th>Label</th><th>Sample Time(ms)</th><th>Status</th></tr><tr><td>1</td><td>21:49:31.182</td><td>Thread Group 2-1</td><td>/login</td><td>2874</td><td></td></tr><tr><td>2</td><td>21:49:34.065</td><td>Thread Group 2-1</td><td>/checkout</td><td>714</td><td></td></tr><tr><td>3</td><td>21:49:31.129</td><td>Thread Group 2-1</td><td>buyer 2 purchase lis...</td><td>3588</td><td></td></tr></table> <table><tr><th></th><th>Start Time</th><th>Thread Name</th><th>Label</th><th>Sample Time(ms)</th><th>Status</th></tr><tr><td>1</td><td>21:49:31.182</td><td>Thread Group 1-1</td><td>/login</td><td>2869</td><td></td></tr><tr><td>2</td><td>21:49:34.065</td><td>Thread Group 1-1</td><td>/checkout</td><td>1044</td><td></td></tr><tr><td>3</td><td>21:49:31.129</td><td>Thread Group 1-1</td><td>buyer 1 purchase lis...</td><td>3913</td><td></td></tr></table>			Start Time	Thread Name	Label	Sample Time(ms)	Status	1	21:49:31.182	Thread Group 2-1	/login	2874		2	21:49:34.065	Thread Group 2-1	/checkout	714		3	21:49:31.129	Thread Group 2-1	buyer 2 purchase lis...	3588			Start Time	Thread Name	Label	Sample Time(ms)	Status	1	21:49:31.182	Thread Group 1-1	/login	2869		2	21:49:34.065	Thread Group 1-1	/checkout	1044		3	21:49:31.129	Thread Group 1-1	buyer 1 purchase lis...	3913	
	Start Time	Thread Name	Label	Sample Time(ms)	Status																																												
1	21:49:31.182	Thread Group 2-1	/login	2874																																													
2	21:49:34.065	Thread Group 2-1	/checkout	714																																													
3	21:49:31.129	Thread Group 2-1	buyer 2 purchase lis...	3588																																													
	Start Time	Thread Name	Label	Sample Time(ms)	Status																																												
1	21:49:31.182	Thread Group 1-1	/login	2869																																													
2	21:49:34.065	Thread Group 1-1	/checkout	1044																																													
3	21:49:31.129	Thread Group 1-1	buyer 1 purchase lis...	3913																																													



## TC10.1 Admin tries to deactivate a user when the user is logged in to the system

<b>Test Type:</b> Functional	<b>Execution Type:</b> Manual testing
<b>Objective:</b> <ul style="list-style-type: none"> <li>- No error message found for each request</li> <li>- Ensure admin can deactivate any users even when user is interacting with the system</li> </ul>	
<b>Setup:</b> <ul style="list-style-type: none"> <li>- Login to the system with admin credential with normal tab</li> <li>- Login to the system with normal user credential with iginito tab</li> </ul>	
<b>Pre-Conditions:</b> <ul style="list-style-type: none"> <li>- User is active in the system</li> <li>- Admin is active in the system</li> </ul>	
<b>Expected outcome:</b> <ul style="list-style-type: none"> <li>- User will be deactivated</li> <li>- Get kicked out by the system</li> </ul>	
<b>Result:</b> PASS	
<b>Notes:</b> <ul style="list-style-type: none"> <li>- Admin deactivate the logged in user</li> <li>- User tries to make any interaction with the system</li> </ul>	

## TC10.2 Admin tries to deactivate a user when the user is updating the account detail

<b>Test Type:</b> Functional	<b>Execution Type:</b> Automated testing
<b>Objective:</b> <ul style="list-style-type: none"> <li>- No error message found for each request</li> <li>- Ensure admin can deactivate any users to avoid when user update the account detail then set is_active to true</li> </ul>	
<b>Setup:</b> <ul style="list-style-type: none"> <li>- Open TC10.2.jmx</li> <li>- Run the test plan</li> </ul>	
<b>Pre-Conditions:</b> <ul style="list-style-type: none"> <li>- User is active in the system</li> <li>- Admin is active in the system</li> </ul>	
<b>Expected outcome:</b> <ul style="list-style-type: none"> <li>- User is deactivated in the system</li> </ul>	
<b>Result:</b> PASS	

**Note:**

	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	23:21:57.289	Thread Group 1-1	/login	4847	✓
2	23:22:02.142	Thread Group 1-1	/deactivate-account	2475	✓
3	23:21:57.223	Thread Group 1-1	admin deactivate a ...	7322	✓

-

	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	23:21:57.289	Thread Group 2-1	/login	3033	✓
2	23:22:00.347	Thread Group 2-1	/edit-account	1285	✓
3	23:21:57.223	Thread Group 2-1	user update account...	4318	✓

-

TC10.3 admin deactivate a user while coseller is updating the listing that user is a owner of the listing

<b>Test Type:</b> Functional	<b>Execution Type:</b> Automated testing
---------------------------------	---

**Objective:**

- No error message found for each request
- Ensure admin can deactivate any users to avoid when user update the account detail then set is\_active to true

**Setup:**

- Open TC10.3.jmx
- Run the test plan

**Pre-Conditions:**

- User is active in the system
- Admin is active in the system

**Expected outcome:**

- User is deactivated in the system
- User detail can be updated

**Result: PASS****Note:**

	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	23:29:39.938	Thread Group 1-1	/login	2891	✓
2	23:29:42.831	Thread Group 1-1	get accounts	497	✓
3	23:29:43.330	Thread Group 1-1	/deactivate-account	2049	✓
4	23:29:39.902	Thread Group 1-1	Admin deactivate a user	5437	✓

-

	Start Time	Thread Name	Label	Sample Time(ms)	Status
1	23:29:39.938	Thread Group 2-1	/login	2672	✓
2	23:29:42.633	Thread Group 2-1	edit-listing	2161	✓
3	23:29:44.805	Thread Group 2-1	/edit-listing	986	✓
4	23:29:39.902	Thread Group 2-1	User update a listing...	5819	✓

## 6 Appendices

### 6.1 Appendix A - Accessing the Heroku Deployment

The Heroku deployment is found at <https://swen90007-agora-marketplace.herokuapp.com/>.

### 6.2 Appendix B - Test Data

The data used for performing the tests is stored within the deployment database itself. The tests provide credentials for accessing the database to reset the data to their previous values afterwards.

### 6.3 Appendix C - Test Result

The test result for the concurrency issue can be found in docs/test\_result, which includes data before the tests were executed and data after the tests were executed.

(<https://github.com/SWEN900072022/SWEN90007-2022-Agora/tags>)

### 6.4 Appendix D - Sample User Data

**NOTE:** the data in the following table is NOT data used in testing. It is supplied for academic staff and the team to be able to provide realistic user data for user testing. The first row, highlighted in yellow, represents the system

Username	Email	Password
admin	admin@agora.com	Admin_123
jphillips	jphillips@mail.net	JP-999
mcwilliams	williams.melissa@mail.com	Furball47
pw_superman	williams.paul@mail.com	Super_Man38
rogerdodger	rbooth@mail.com	Dodger971
newtons_low	sinclair.n@mail.com	Game!01