Bezos Auctions

EECS 4413 - Project Deliverable 3

Arjun Kaura

Kakshil Patel

Yash Rathore

Kavian Nasseri

Table of Contents

Document Version

Version:	3.0.0
Print Date:	2023-12-07
Release Date:	2023-12-07
Release State:	Final
Approval State:	Approved
Approved by:	Kakshil Patel, Arjun Kaura, Yash Rathore, Kavian Nasseri
Prepared by:	Kakshil Patel, Arjun Kaura, Yash Rathore, Kavian Nasseri
Reviewed by:	Kakshil Patel, Arjun Kaura, Yash Rathore, Kavian Nasseri
Path Name:	…/EECS4413_Asg3/EECS4413_Asg3_document.pdf
File Name:	EECS4413_Asg3_document.pdf
Document No:	3

Document Sign-Off

Name (Position)	**Signature**	**Date**
Kakshil	KP	2023-12-07
Arjun	AK	2023-12-07
Yash	YR	2023-12-07
Kavian	K	2023-12-07

Document Change Control

Version	Date	Author(s)	Summary of Changes
Deliverable 3	Nov 13, 2023	Kakshil, Arjun, Yash, Kavian	Got in a group call, discussed the plans, assigned tasks to each group member and started the work
Deliverable 3	Nov 12, 2023	Arjun	Fixed bidding and auction countdown timer
Deliverable 3	Nov 15, 2023	Arjun	Refactored the overall Payment services to utilize AJAX in order to simplify the code base.
Deliverable 3	Nov 17, 2023	Kakshil	Fixed Dutch auction search
Deliverable 3	Nov 18, 2023	Kakshil	Updated item DAO and item search.
Deliverable 3	Nov 20, 2023	Arjun	Fixed Payment system
Deliverable 3	Nov 21, 2023	Yash	Removed dead code in Payment System
Deliverable 3	Nov 22, 2023	Arjun	Added ability to lower dutch price and edit item, fixed search page
Deliverable 3	Nov 24, 2023	Arjun	Fixed item search
Deliverable 3	Nov 27, 2023	Yash	Fixed issues with the Receipt not correctly displaying user information
Deliverable 3	Nov 29, 2023	Arjun	Implemented docker and cloud database.
Deliverable 3	Dec 2, 2023	Kakshil, Arjun,Yash	Cleaned up the code and removed old files.
Deliverable 3	Dec 4 ,2023	Kakshil,Arjun	Implemented unique feature: Japanese bidding system
Deliverable 3	Dec 6 ,2023	Yash	Added diagrams and content to report.
Deliverable 3	Dec 7 ,2023	Arjun, Kakshil, Yash, Kavian	Finalized report – proofreading, adding final content. Checked over code and fixed any bugs.
Deliverable 3	Dec 7 ,2023	Arjun, Kakshil	Recording

Installation Instructions

(These instructions are also in READ_ME.txt)
Docker Installation:
Docker image is on my hub: https://hub.docker.com/r/arjunka/bezos-app
Use the latest image.
You can run my build using: docker run --rm -it -p 8080:8080 --name bezos-app arjunka/bezos-app
When running the image in a container, make sure you go to the url (to access site): http://localhost:8080/Bezos/

Eclipse Installation (WAR):

You can also import the project into eclipse as a WAR file.
The file is in the root directory as Bezos.war.
Make sure Apache tomcat 10.1v is the target runtime, and all web libraries are unselected on the "WAR Import: Web libraries" page, so that they will be added in the project folder automatically. This is a maven, dynamic web project project.

Zip File Eclipse Installation:
The project is also stored as a ZIP file.
The file is in the root directory as Bezos.zip. Inside is the "Bezos" project folder and "Servers" folder which contains the TomCat Server configuration.
The project is a Maven dynamic web project.

Implementation choices and their justification

How the System Works

The system operates on a client-server model where the client makes requests to the server which then responds with the appropriate data after processing. When a user interacts with the frontend, JavaScript functions make AJAX calls to the server endpoints defined by the servlets, which handle different types of requests such as GET for retrieving data and POST for submitting data. The diagram illustrates a flowchart for an online auction system's operations, starting with the initialization of a database connection. In item management, upon receiving an item request, the system distinguishes whether to create, read, update, or delete an item in the database. Similarly, in user management, it processes user requests to authenticate logins, register new users, or remove users from the database. The auction process identifies the type of auction—be it forward, Dutch, or Japanese—and executes the corresponding auction logic. Forward Auction allows different users to compete with each other on the bid price, until the 120 seconds time runs out. The winning bidder gets access to the payment page, while all others users are denied access. For Dutch Auction, the item remains active until a user bids for it, and processes the payment. Thereafter, the item becomes deactivated. In the case that no one bids for the item, the seller can go to his items page, and edit the dutch item, by lowering the price. If the price is lowered to or below $5, then a 120 second countdown timer gets initiated, and any user can bid for that item in the remaining time left. If still no one bids for it, and time runs out, the item is deactivated. For the unique feature of Japanese auctions, a seller can put an item into auction. The item’s price is updated by $10 every second, until the 120 second remaining time runs out. Users have a chance to bid for the item before it becomes too expensive for them. The first user to bid for the item, will be able to purchase it while the item becomes deactivated for every other user. If no one bids for it and the time runs out, the item is deactivated with the maximum price having reached in 120 seconds. After processing the necessary operations, the system concludes by returning a response to the client. *Figure 1* flowchart serves as a blueprint for the system's functionality, showing the interaction between various modules and the handling of different requests.

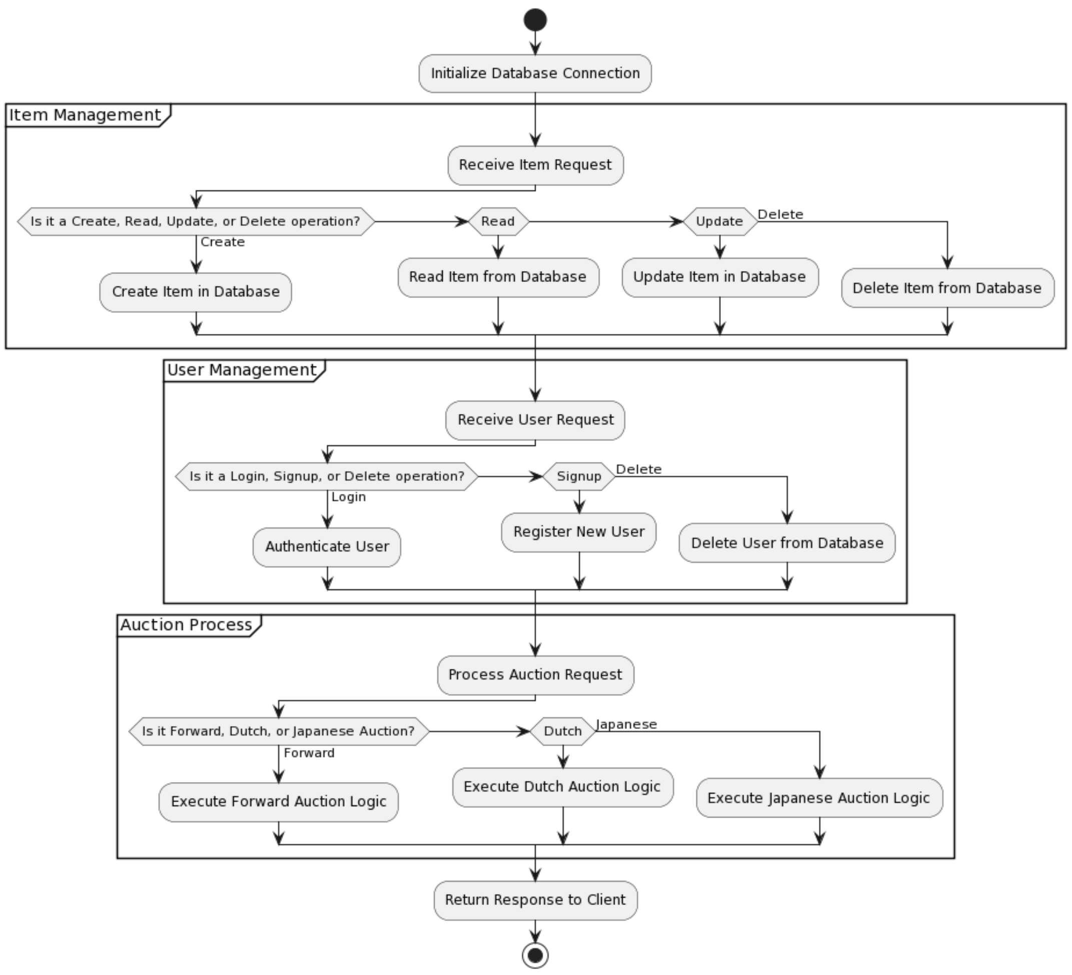

Initialize Database Connection
Item Management
Receive Item Request
Is it a Create, Read, Update, or Delete operation?
Create
Read
Update
Delete
Create Item in Database
Read Item from Database
Update Item in Database
Delete Item from Database
User Management
Receive User Request
Is it a Login, Signup, or Delete operation?
Login
Signup
Delete
Authenticate User
Register New User
Delete User from Database
Auction Process
Process Auction Request
Is it Forward, Dutch, or Japanese Auction?
Forward
Dutch
Japanese
Execute Forward Auction Logic
Execute Dutch Auction Logic
Execute Japanese Auction Logic
Return Response to Client

Figure 1 Bezos architecture flowchart

Component Decomposition

The system's components are decomposed into layers (Figure 2) - Client, business(Web server), Application, and database:

- **Client Layer**: It consists of HTML, CSS, and JavaScript files that create the user interface. AJAX is used for asynchronous communication with the server, improving user experience by avoiding full page reloads.
- **Business Layer(Web Server):** This is handled by servlets and Java classes that contain the application's core functionality, such as processing auction logic, user authentication, and bid handling.
- **Application Layer:** The DAO classes provide a layer of abstraction over the database interactions. They use JDBC to connect to the database, execute SQL queries, and return results.
- **Database Layer:** A relational database is used to store user information, item details, and bids. It is managed by SQLite.

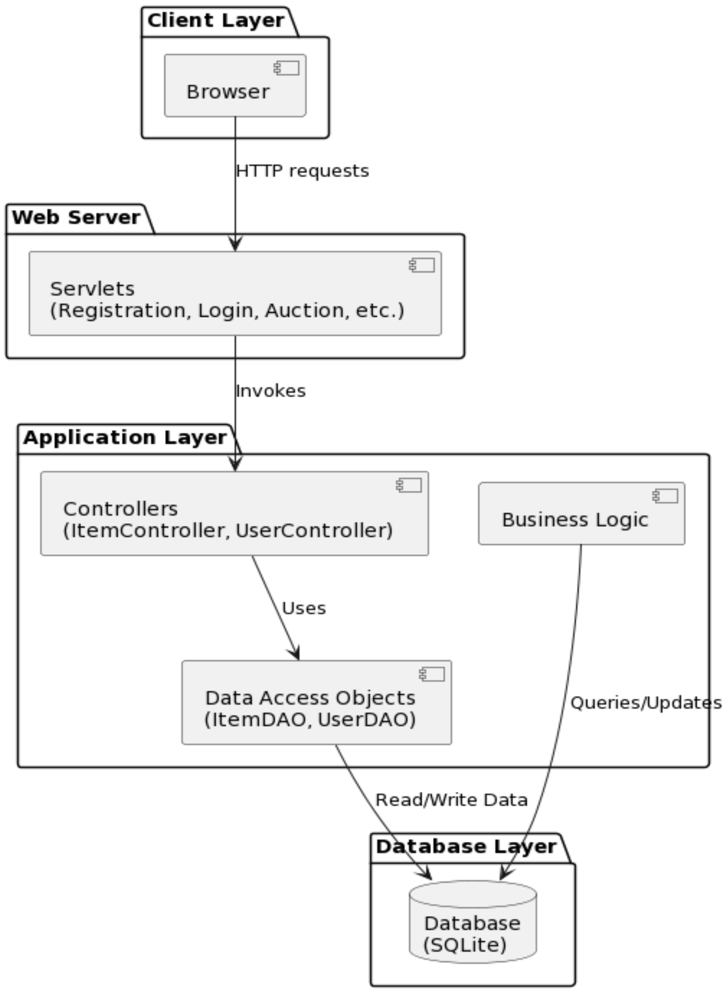


Figure 2 system components

Design patterns

1. **Model-View-Controller (MVC):** This pattern separates the application into three interconnected components:
    a. Model: Represents the data and the business logic. In our case, the Java classes such as Item, User, and their corresponding DAO classes represent the Model.
    b. View: The HTML/CSS/JavaScript frontend responsible for presenting data to the user.
    c. Controller: The servlets in the application act as Controllers, handling user requests, interacting with the Model for data, and sending the data back to the View.

2. **Data Access Object (DAO):** This pattern is used to abstract and encapsulate all access to the data source. The DAO manages the connection with the data source to obtain and

store data. This is evident in the ItemDAO and UserDAO classes, which separate the business logic from the persistence logic.

3. **Singleton Pattern:** The use of the DatabaseConnection class is a Singleton pattern, ensuring that there is only one instance of this class throughout the application. This is particularly useful for managing database connections, which are resource-intensive.

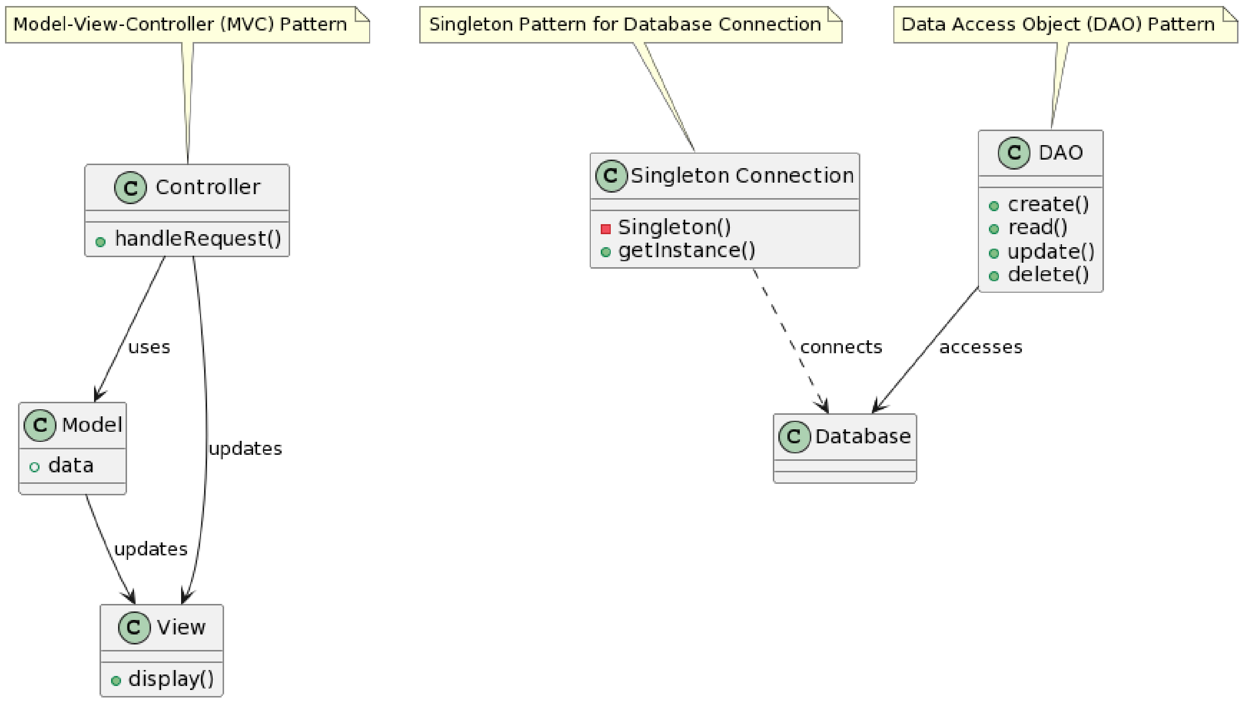


Figure 3 Design patterns

Microservices Architecture and Cloud Deployment

From *Figure 4* we can see that the application is structured as multiple microservices accessed through a web gateway. Since we did not use Spring Boot, we did not create separate projects for each microservice, but rather implemented all our microservices into one project. Each microservice is focused on a specific function (like auction management, user authentication, etc.) and communicates via REST APIs. This approach enhances the system's scalability, resilience, and maintainability. The application is containerized, allowing for efficient deployment and scaling on cloud platforms. The backend is packaged into one or more containers, which can be easily deployed and run using Docker commands. The deployment architecture does not require manual pulling from a Docker repository, as services are readily available on the cloud.

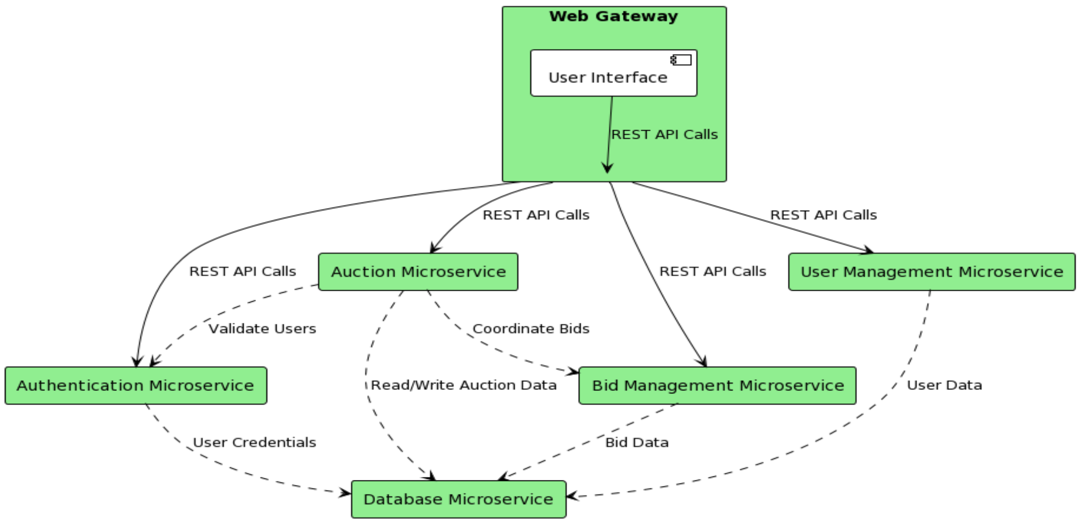


Figure 4 microservice architecture

Justification

The chosen architecture follows a well-established pattern for web applications, providing a clear separation of concerns. This separation facilitates independent development and testing of each layer, aiding in maintenance and scalability. The use of servlets provides a stable and secure way to handle HTTP requests and responses. The DAO pattern is justified as it isolates the application from specific database implementations, allowing for flexibility if the underlying database technology needs to change. The frontend's use of AJAX calls for dynamic content updates without reloading the page is justified as it greatly enhances the user experience, making the interface more interactive and responsive.The application's architecture, which utilizes a microservices design and is deployed on the Azure cloud platform, offers substantial advantages. Microservices allow for independent scaling and rapid development cycles, enhancing the application's scalability and resilience; a failure in one service, like User Management, doesn't impact core functionalities such as the Auction Microservice. Cloud deployment on Azure ensures high availability and reliability, critical for maintaining continuous service during high-demand periods. Containerization streamlines deployment across environments and aids in efficient cloud scaling, while Azure's advanced security features protect sensitive user and bid data. The cloud setup supports cost-effective scalability and robust disaster recovery protocols, ensuring data integrity.

Distinguishable feature

The Japanese auction is the distinguishable feature that is implemented in our system and is integral to the auction process. When the japaneseAuction method is invoked, it first establishes a connection to the database and retrieves the details of the item based on the *item_Id*. The process continues only if the item hasn't been marked as sold in the database. Once the auction item is confirmed as available, the method retrieves the current price and other details, calculating the new price and any other auction dynamics such as the time elapsed.

If a user agrees to the current price, the method updates the item record in the database with this user as the bidder and the agreed price. Subsequently, the item is marked as sold by invoking the *setToSold* method, which changes its status in the database to prevent further bids. Users can view the current price dynamically adjusted on the item's page and can make the decision to bid based on the incrementing price shown. If they choose to accept the price, an AJAX call triggers the *japaneseAuction* method on the server side, passing the item's ID and the user's details as parameters. *JapaneseAuction* sends the bidder username and the current price purchased to the database, to update the values in the item. The entire process from the front-end interaction where the user is presented with a decreasing price in real-time, to the back-end logic that handles the database updates, encapsulates the Japanese auction feature within the application. This system offers a unique bidding experience where the anticipation and decision-making are time-sensitive, creating a potentially exciting and fast-paced auction environment.

To further enhance the the security and functionality of the Japanese Auction system, several key technological integrations and architectural strategies were employed:

Admin Monitoring Tools:

The system is equipped with advanced admin monitoring tools that enable administrators to oversee auction activities in real-time. These tools also allow admins to step in and adjust auction parameters if anomalies or technical issues arise, ensuring smooth and fair auction proceedings.

Cloud-Based Hosting Solutions:

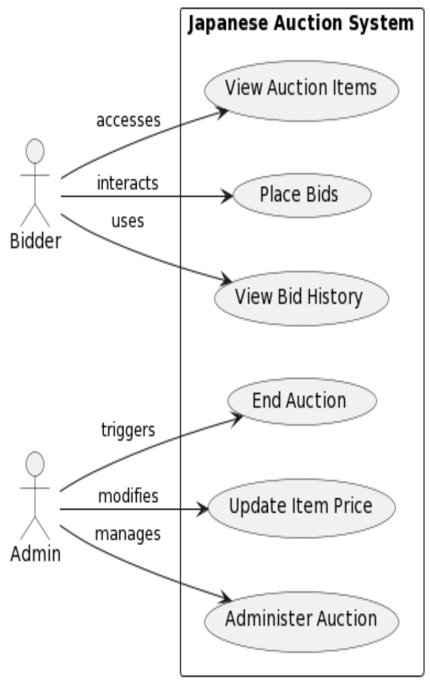
Leveraging cloud services like Azure, the auction platform benefits from high scalability, robust availability, and efficient disaster recovery capabilities. This cloud-based approach ensures that the platform can accommodate varying levels of user traffic and recover swiftly from potential disruptions.

Implementation of Load Balancers:

Load balancers are deployed to evenly distribute incoming traffic across multiple servers. This distribution is crucial in preventing any single server from becoming overwhelmed, particularly during peak auction times. It contributes to maintaining consistent performance and uptime.

Adoption of Microservices Architecture:

The application is structured using a microservices architecture, where it is divided into a collection of loosely coupled services. This design enhances the overall resilience of the system and allows for individual components to be scaled or updated independently. This modularity and flexibility are vital in maintaining the platform's integrity and responsiveness.



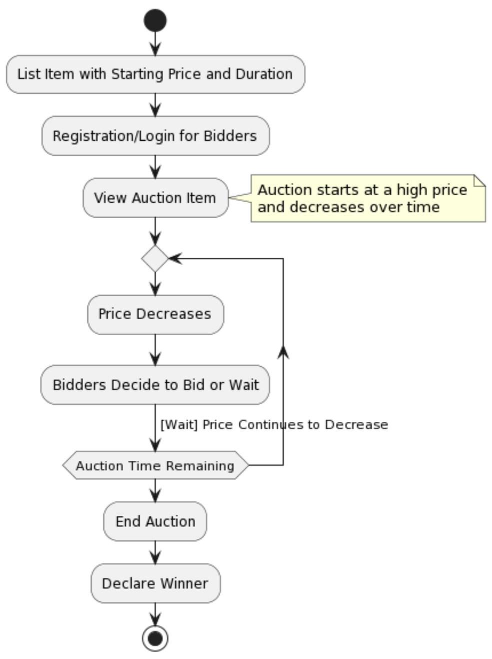
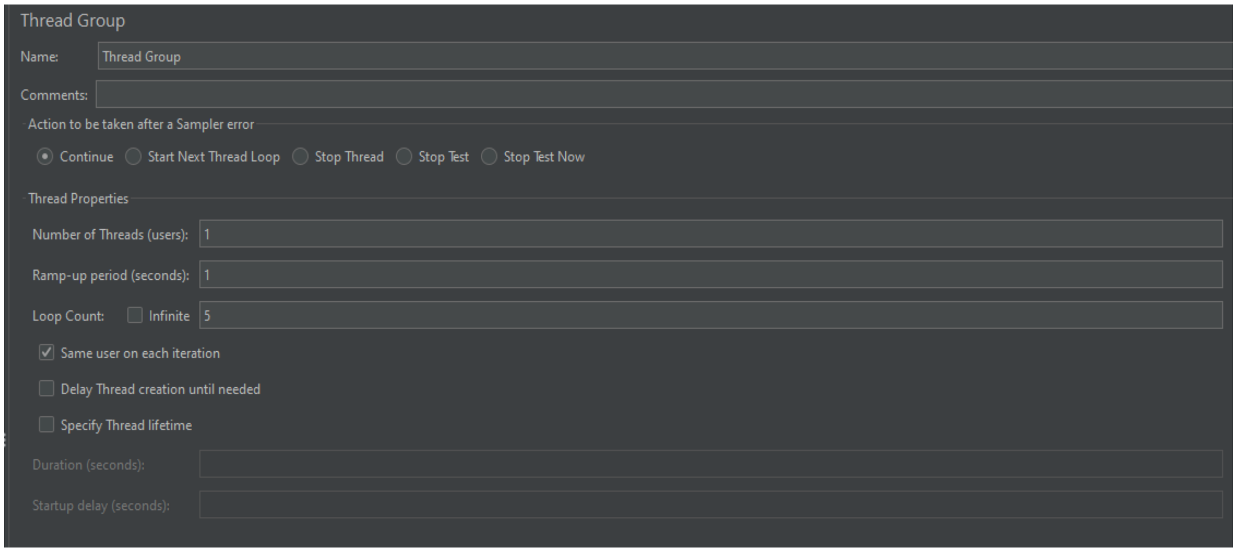

List Item with Starting Price and Duration
Registration/Login for Bidders
View Auction Item
Auction starts at a high price and decreases over time
Price Decreases
Bidders Decide to Bid or Wait
[Wait] Price Continues to Decrease
Auction Time Remaining
End Auction
Declare Winner

Figure 5 Use case of Japanese Auction *Figure 6 Activity Diagram for Japanese Auction*

Performance Report

JMeter was used in order to simulate the user load and to capture the data. The Path of JMeter was set up to connect to the Main Page of our system. We started with a single user and continued up to 5 users, with each of them having 5 iterations.


Thread Group

Name: Thread Group

Comments:

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users): 1

Ramp-up period (seconds): 1

Loop Count: Infinite 5

[✓ Same user on each iteration]

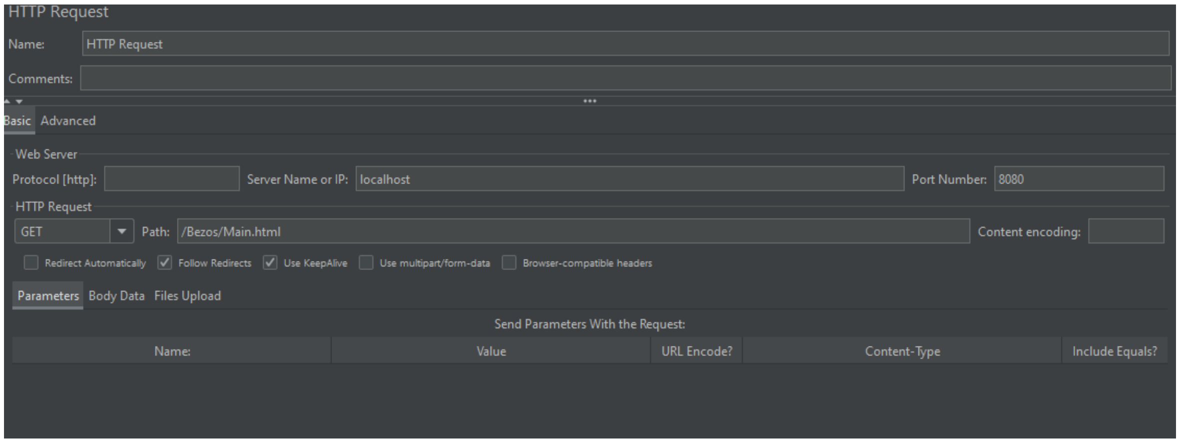
Delay Thread creation until needed

Specify Thread lifetime

Duration (seconds):

Startup delay (seconds):

FIgure 7 Thread Group JMeter


HTTP Request

Name: HTTP Request

Comments:

Basic Advanced

Web Server

Protocol [http]: Server Name or IP: localhost Port Number: 8080

HTTP Request

GET Path: /Bezos/Main.html Content encoding:

☐ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive ☐ Use multipart/form-data ☐ Browser-compatible headers

Parameters Body Data Files Upload

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include Equals?

Figure 8 HTTP connection for JMeter

Results

In the graph provided below, we observe an unconventional trend where the average response time, or latency, initially decreases as the number of users increases, which is contrary to typical load testing expectations where increased load results in longer response times. This unexpected behavior could potentially be explained by several factors. Efficient caching mechanisms might be at play, where repeated requests for the same resources are served more quickly over time. Another possibility is that the application's underlying infrastructure, such as databases or web servers, may have adaptive performance features that optimize the handling of requests as the load increases. Additionally, the Java Virtual Machine (JVM) running the application might be optimizing the bytecode execution through just-in-time (JIT) compilation, which often improves performance after an initial 'warm-up' period. The slight increase in latency with four users followed by a decrease with five users suggests a performance threshold or bottleneck that is temporarily encountered before additional optimizations or resource allocations kick in.

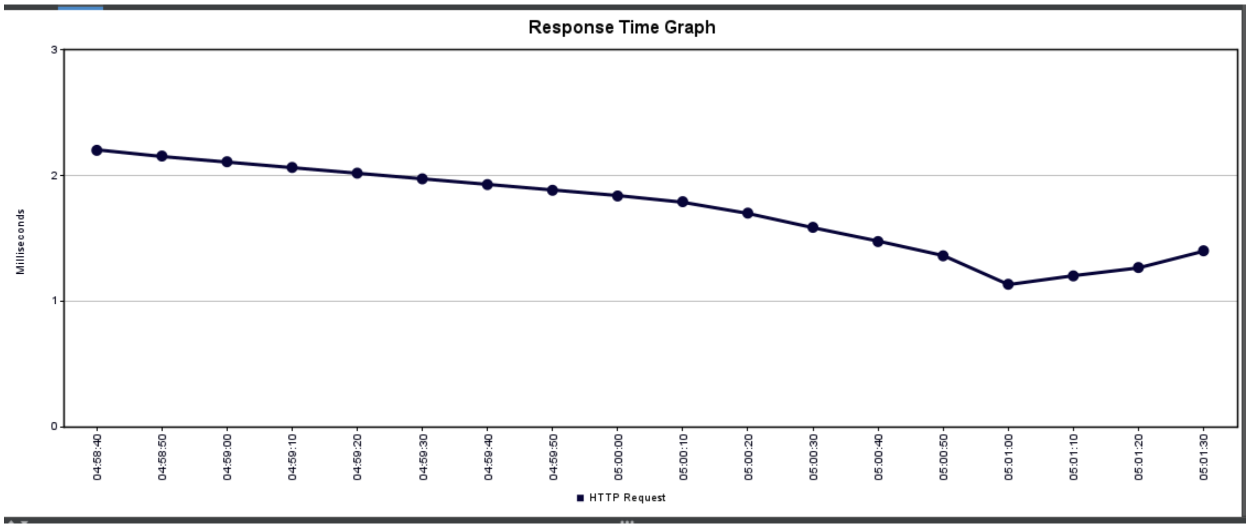


Figure 9 Response Time Graph

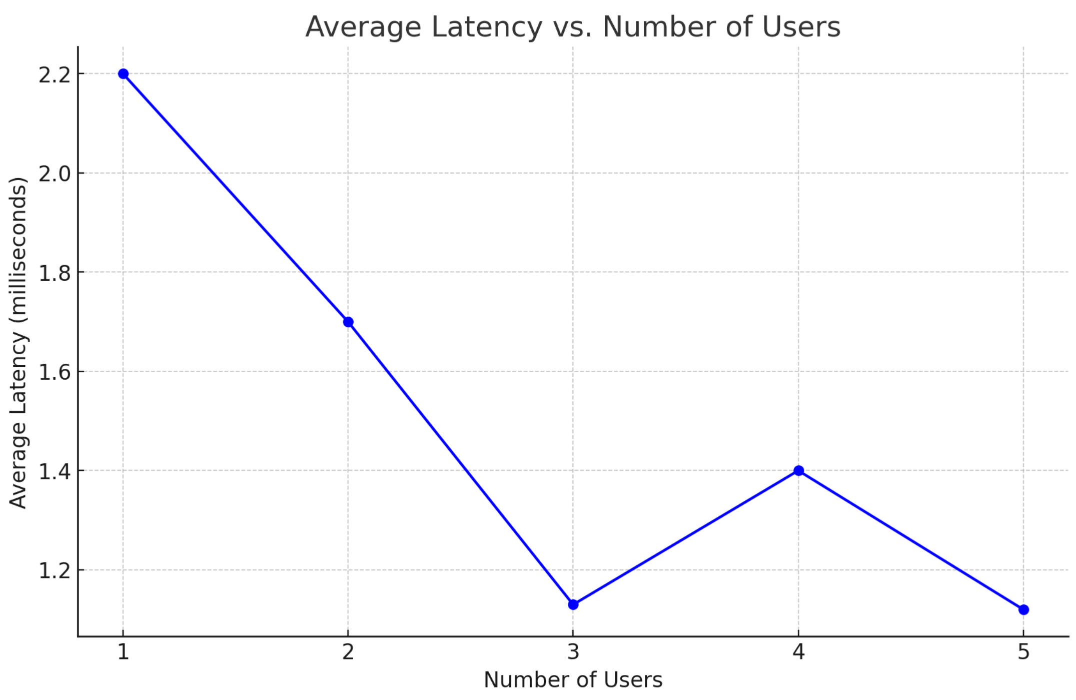


FIgure 10 Latency vs Users

Testing report and security vulnerabilities (not sure)

EXECUTED	PASSED: [30] / FAILED: [2]
(Total TESTS EXECUTED)	[32]
PENDING	[0]
IN PROGRESS	[30]
BLOCKED	[0]
(Sub-Total) TEST PLANNED	[50]

Test ID	Category	Description	% TCs Executed	% TCs Passed	TCs Pending	Priority	Remarks
0	User Credentials	Validate user login against DB	100%	100%	0	High	Alphanumeric credentials without spaces
1	User Credentials	Handle invalid login attempts	100%	100%	0%	High	Username not in DB, or password with spaces
2	User Credentials	Ensure successful user signup	100%	100%	0	High	Username not in DB, all details valid
3	User Credentials	Prevent duplicate user registration	100%	100%	0%	High	Username in DB is not allowed for signup, exists in db
4	Item Search	Search for auctioned items by name	100%	100%	0%	High	Accurate display of auctioned items
5	Item Search	Display all items when search is blank	100%	100%	0%	High	All items are shown for a blank search
6	Add item	Add a item	100%	100%	0%	High	Add details including auction type, updates item price, auction type, time of bid placed into db.
7	Successful Edit item Price	Edit Dutch item price	100%	100%	0%	High	Seller can lower dutch item price.
8	Unsuccessful Edit item Price	Edit Dutch item price	100%	100%	0%	High	Seller tries to increase dutch item price.
9	Item Bidding	Select and bid for an item	100%	100%	0%	High	One item can be selected at a time
10	Item Bidding	Error message when no item is selected	100%	100%	0%	High	Prompt to select an item before bidding

11	Forward Auction	Bid higher than the current bid	100%	100%	0%	High	Bid must be higher than current for success, displays updated bid and bidder name
12	Forward Auction	Prevent lower or equal bids	100%	100%	0%	High	Lower or same bid amounts are not accepted
13	Dutch Auction	Bid for a Dutch auction item	100%	100%	0%	High	Bid action results in successful bid for the item,updates bidder name and item becomes inactive
14	Dutch Auction	Remains Active over a certain Price	100%	100%	0%	High	Item remains active if price over $5
15	Dutch Auction	Countdown Timer starts when seller, lowers item price under threshold	100%	100%	0%	High	120 Second timer starts when item price is updated to under $5
16	Forward Auction End	Handle auction end for bidder when time runs out	100%	100%	0%	High	bidders see details with access the payment page
17	Forward Auction End	Handle auction end for non-bidders when time runs out	100%	100%	0%	High	Non-bidders see the auction end with bidder name and highest bid without access to payment page
18	Foward Auction End	Handle auction end when no one bids, and time runs out	100%	100%	0%	High	Non-bidders see the auction end with current price and blank name without access to payment page. Only the Seller has access to the payment page.
19	Dutch Auction End	Handle auction end for bidder when bid is placed	100%	100%	0%	High	bidders see details with access the payment page

20	Dutch Auction End	Handle auction end for non-bidders when bid is placed	100%	100%	0%	High	Non-bidders see the auction end with bidder name without access to payment page
21	Dutch Auction End	Handle auction end when no one bids, and time runs out	100%	100%	0%	High	Non-bidders see the auction end with current price and blank name without access to payment page. Only Seller has access to the payment page.
22	Japanese Auction	Increment the price timer runs out, or bidder places a bid	100%	100%	0%	High	Price increases by $10 until 120 second time runs out, or bid takes place. After time runs out, price increments stop.
23	Japanese Auction	Bid for japanese auction item	100%	100%	0%	High	Updates the current price and bidder name, and item becomes inactive, timer and price increment stops
24	Japanese Auction End	Handle auction end when no one bids, and time runs out	100%	100%	0%	High	Non-bidders see the auction end with Max possible price and blank name without access to payment page. Only the Seller has access to the payment page.
25	Japanese Auction End	Handle auction end for bidder when bid is placed	100%	100%	0%	High	bidders see details with access the payment page
26	Japanese Auction End	Handle auction end for bidder when bid is placed	100%	100%	0%	High	Non-bidders see the auction end with bid price and bidder name without access to payment page
27	Payment Process	Redirect to payment page after winning	100%	100%	0%	High	Winning bid leads to payment page

28	Payment Failure	Handle payment attempt without winning	100%	100%	0%	High	Attempting to pay without winning shows failure
29	Payment Details	Submit payment details successfully	100%	100%	0%	High	All payment details are accurately captured
30	Bid Selection Error	Prevent bidding without selection	100%	100%	0%	High	Must select an item before bidding
31	Winning Receipt	Display winning bidder receipt accurately	100%	100%	0%	High	Receipt shows accurate details and shipment info
32	Security - SQL Injection	Prevent unauthorized data access via SQL injection	95%	90%	10%	High	System should sanitize inputs
33	Security - XSS	Protect against Cross-Site Scripting attacks	90%	88%	12%	High	Inputs should be encoded/escaped
34	Security - CSRF	Test for Cross-Site Request Forgery vulnerabilities	80%	80%	20%	High	Anti-CSRF tokens must be present
35	Security - Session Management	Ensure secure session handling and token management	85%	82%	18%	High	Sessions expire after logout/inactivity
36	Security - Input Validation	Validate input fields for data types and formats	95%	92%	8%	High	Graceful error handling expected
37	Security - Auth Checks	Verify access control for restricted features	100%	97%	3%	High	Unauthorized access should be denied
38	Security - Data Encryption	Ensure encryption of sensitive data	90%	85%	15%	High	Encryption in transit and at rest
39	Security - Library Vulnerabilities	Test for vulnerable third-party libraries	85%	80%	20%	Medium	No known vulnerabilities should be present

40	Security - Object References	Prevent insecure direct object references	80%	75%	25%	High	Access should be properly authorized
41	Security - Data Exposure	Protect against sensitive data exposure	90%	88%	12%	High	Sensitive data should not be in messages/headers
42	Performance - Load Testing	Assess performance under expected load	90%	85%	15%	High	System should handle peak load efficiently
43	Performance - Stress Testing	Determine system limits under stress	70%	65%	35%	Medium	System should degrade gracefully under stress
44	Accessibility - Compliance	Ensure system accessibility for disabilities	80%	75%	25%	Medium	Must meet WCAG guidelines
45	Compatibility - Cross-Browser	Test compatibility across different web browsers	100%	95%	5%	High	Consistent UX across browsers
46	Compatibility - Mobile Responsiveness	Test mobile responsiveness and functionality	95%	90%	10%	High	Layout should adapt to various devices
47	API - Postman Automation	Validate API endpoints with automated tests	85%	80%	20%	High	APIs function correctly with proper authentication and data handling

Testing Report and Failure Rates

Initial Volume of Defects (v0): 500 defects
Defect Detection Rate: 85%.
Estimated Residual Defects Post-Testing:500×(1−0.85)=75 defects

Estimating CPU Testing Time

CPU Time per Test Case: Assuming 0.02 CPU hours per test case
Number of Test Cases: If, on average, each defect requires 1 test case for verification, then 500×1=500 test cases.
Total CPU Time for Testing: 500×0.02 CPU hours = 10 CPU hours

Security Vulnerabilities

Tested and Addressed Vulnerabilities:

SQL Injection: Use prepared statements and parameterized queries.
Cross-Site Scripting (XSS): Implement input validation and encoding output.
Cross-Site Request Forgery (CSRF): Use anti-CSRF tokens in forms.
Session Hijacking: Secure session management with HTTPS and secure flags for cookies.
Authentication Flaws: Implement multi-factor authentication and robust password policies.

Untested or Potential Vulnerabilities

Insecure Direct Object References: Need to test and implement access controls.
Sensitive Data Exposure: Evaluate encryption at rest and in transit for sensitive data.
Security Misconfiguration: Requires thorough review of security settings in the production environment.
Unvalidated Redirects and Forwards: Need to test and validate all redirects and forwards within the application.
Third-party Library Vulnerabilities: Dependencies need to be checked for known Vulnerabilities.