





*Process MeNtOR 1.0*

EECS 4413 Deliverable 3  
**Software Specification  
Document (SSD)**

## Document Change Control

Version	Date	Authors	Summary of Changes
3.1	Friday March 28th, 2025	William Moran	-Created document -Started explanation of new distinguishable feature
3.2	Monday March 31st, 2025	Hayyaan Paurobally	-Updated architecture section -Added descriptions about infrastructure layer, microservice layer, and frontend architecture
3.3	Tuesday April 1st, 2025	Daniel Santorelli	-Expanded on distinguishable feature explanation, including examples of applicable scenarios -Wrote group meeting logs and did a full editing pass on the document so far
3.4	Tuesday April 1st, 2025	Jay Raut	-Wrote performance report -Wrote Testing report and security vulnerabilities

## Document Sign-Off

Name (Position)	Signature	Date
William Moran		2025/04/01
Hayyaan Paurobally		2025/04/01
Daniel Santorelli		2025/04/01
Jay Raut		2025/04/01

# Contents

<b>1 Introduction</b>	<b>4</b>
1.1 Purpose	4
1.2 Overview	4
1.3 Resources -References	5
<b>2 Major Design Decisions</b>	<b>5</b>
<b>3 Sequence Diagrams</b>	<b>7</b>
Use Case 1: Sign-up and Sign-in	7
Use Case 2: Browse Catalog	8
Use Case 3: Bidding	9
Use Case 5: Payment	10
<b>4 Activity Diagrams</b>	<b>11</b>
Use Case 1: Dutch Auction Update	11
Use Case 2: Bidding	11
Use Case 3: Auction Ended	11
Use Case 4: Payment	12
<b>5 Architecture</b>	<b>12</b>
Infrastructure Layer	13
Microservices Architecture	13
Frontend Architecture	14
<b>6 Activities Plan</b>	<b>16</b>
6.1 Project Backlog and Sprint Backlog	16
6.2 GANTT Chart	18
6.3 Group Meeting Logs	18
<b>7 Test Driven Development</b>	<b>20</b>
Use Case 1: Sign up and Sign-in	20
Use Case 2: Browse Catalogue of Auctioned Items	22
Use Case 3: Bidding	24
Use Case 4: Auction Ended	24
Use Case 5: Payment	25
Use Case 6: Receipt Page and Shipment Details	26
Use Case 7: Sell Item	26
Use Case 8: Dutch Auction Update	27
<b>8 Performance Report</b>	<b>28</b>
<b>9 Testing Report and Security Vulnerabilities</b>	<b>29</b>

# 1 Introduction

This Software Design Document (SSD) outlines the improvements in developing an auction e-commerce application. The document serves as a guide for the system's design, architecture, and implementation, highlighting the changes for each. It contains the processes we followed throughout the development, detailed sequence, activity and architecture diagrams, and important design decisions in line with project requirements. This report provides us developers, a reference for our system architecture and implementation details and allows for flexibility when we create changes to the system. As a result, it ensures that every aspect of the system is aligned with the project's requirements, performance targets, and user expectations.

## 1.1 Purpose

The objective of this project was to develop a robust, auction-based e-commerce platform that enables users to buy and sell items through competitive bidding. The system aims to support two types of auctions: Forward Auctions and Dutch Auctions. Forward Auctions are a form of auction where sellers list items and buyers place increasingly higher bids until the highest bid wins. Dutch Auctions are a form of auction where sellers start with a high price that gradually decreases until a buyer accepts the current price. This application provides an efficient way for buyers to engage in bidding and for sellers to list their items. The system has features like real-time updates, secure payment processing, and user-friendly navigation that allows users to have an enjoyable experience.

## 1.2 Overview

The auction system includes:

- User authentication: Secure user sign-up and sign-in mechanisms
- Auction management: Ability for sellers to create, edit, and end auctions
- Real-time bidding: Live updates on auction status and bidding activity
- Item searching: Users can search for auctioned items using keywords
- Payment processing and receipts system: Secure transaction handling and automated receipt generation
- Auction expiry management: Ensures auctions conclude at the designated time or when a buyer accepts a Dutch Auction price.

This system is built using a multi-tier architecture, including a front-end user interface, a middleware application layer, and a back-end database. The back end manages auction logic, user authentication, and payment processing. Meanwhile, the front end provides an intuitive and responsive user experience.

## 1.3 Resources -References

- [1] Aljaf, B. (2016, December). *Online Auction System*. Theseus. <https://urn.fi/URN:NBN:fi:amk-2016122221416>
- [2] Odoh, K. E. (2012, January 12). *Design and implementation of a web-based auction system*. Theseus. <https://urn.fi/URN:NBN:fi:amk-201201211534>
- [3] Vidal Segura, M. (2019, July 2). *User experience design and front end development of an online auction website*. DiVA. <https://urn.kb.se/resolve?urn=urn%3Anbn%3Ase%3Aakth%3Adiva-254581>

## 2 Major Design Decisions

Application Design Choice **Microservice**:

- Major components such as front-end, middle subsystem and back-end are made up of multiple services
- Front-end service, API gateway, auth service, auction service, payment service
- Application communicates with each other using RESTful APIs
- Use of sockets for listening for events such as time remaining on auction or live bids
- Use of session-based auth for user management such as JWT

Database Design:

- Uses a single relational database to store customer information, current auction listings, and payment information.
- Uses rules inside of database schema to enforce business rules

Architecture Pattern:

- **Chosen Pattern:** Microservices Architecture
  - Aspects considered:
    - Scalability
    - Fault isolation
    - Much more flexibility in language
    - Possible to make development faster since different services can be done by multiple people
  - Possible issues:
    - Increase in complexity
    - Higher latency or increased overhead in communications between services

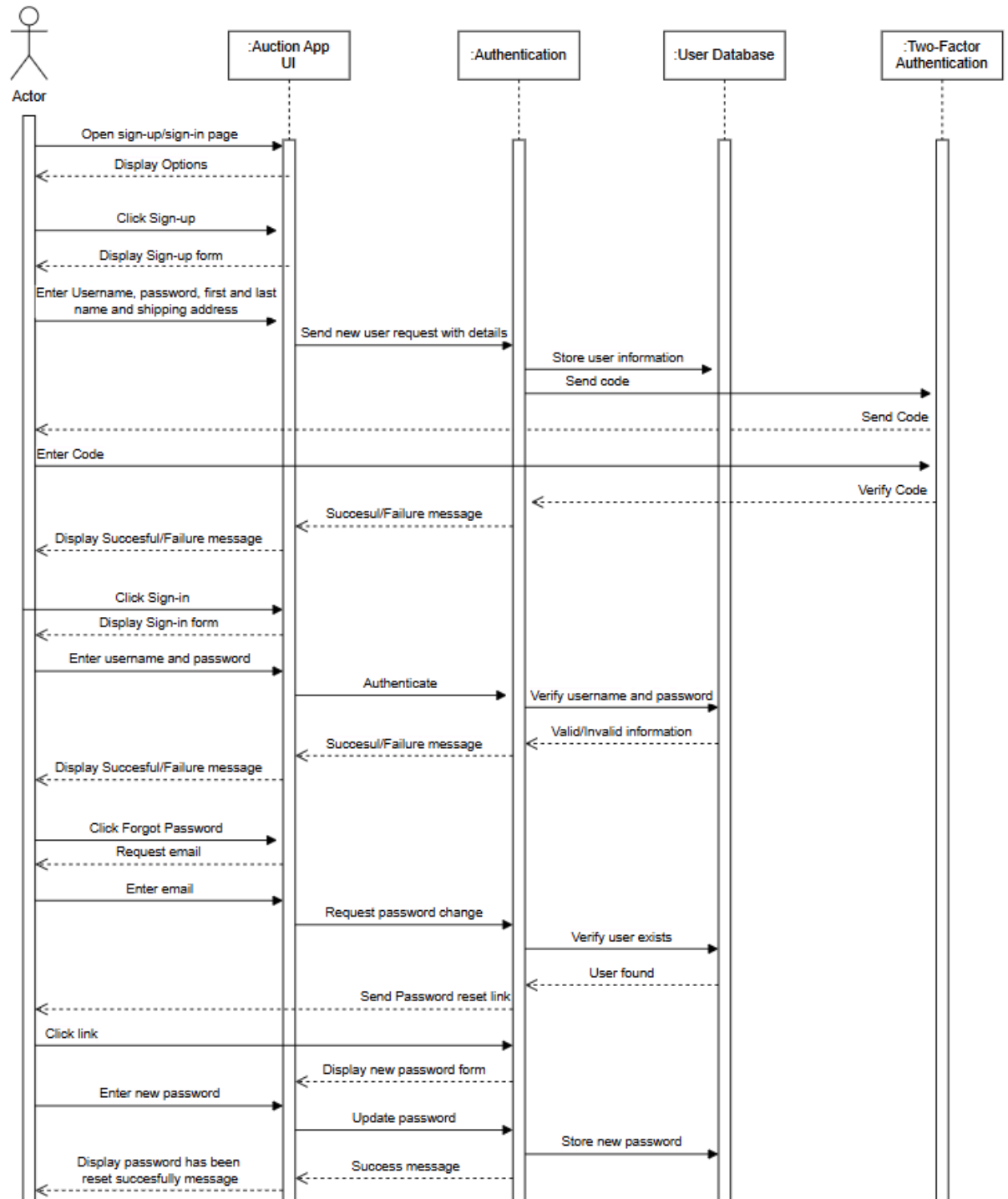
- Loose coupling may cause complexity in data management
- **Considered pattern: Monolith Architecture**
  - Aspects considered:
    - Easier development and debugging, a single development unit
    - Tight coupling between multiple components such as auction to payment
    - Faster communication between components good for real time use
  - Possible issues:
    - Scaling is a challenge
    - Failure in one part may bring down whole system
    - Limited technology choice (forced to use a single language for everything)
- **Considered pattern: Serverless Architecture**
  - Aspects considered:
    - Easier to scale
    - Reduced infrastructure management
    - Built-in fault tolerance
    - Good at handling real-time communication
  - Possible issues:
    - High latency
    - Complex to manage
    - Dependent on cloud providers

### **Distinguishable Feature:**

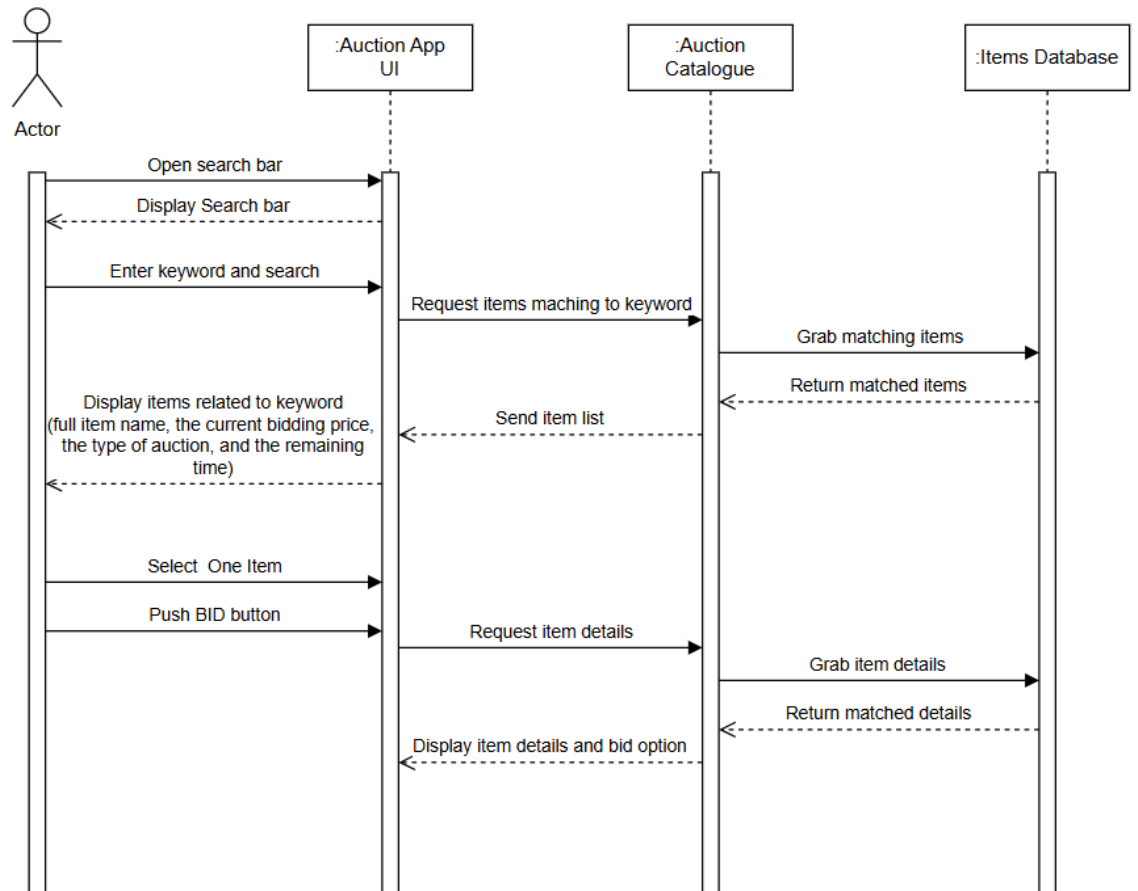
The distinguishing feature that our group decided to implement into our project is giving the auctioneer the option to set a specific date in the future for when they want their items to go live in the auction. This new use case will bring several unique benefits for the user. Firstly, it gives sellers additional flexibility as the website better caters to their schedules (for example, if a seller likes to begin auctions on a specific day of the week for maximum exposure but will not have internet access on that day). Secondly, it may encourage sellers to create more listings – for example, if a seller prefers to begin one new auction every day, they can simply create multiple new auctions in a single sitting, each starting on a different day, rather than being forced to return to the site daily to create a single new one each time. Finally, if a seller has an item to sell that they anticipate will be trending in the near future, they can capitalize on this rush of exposure by choosing to start their auction then rather than right away.

### 3 Sequence Diagrams

#### Use Case 1: Sign-up and Sign-in

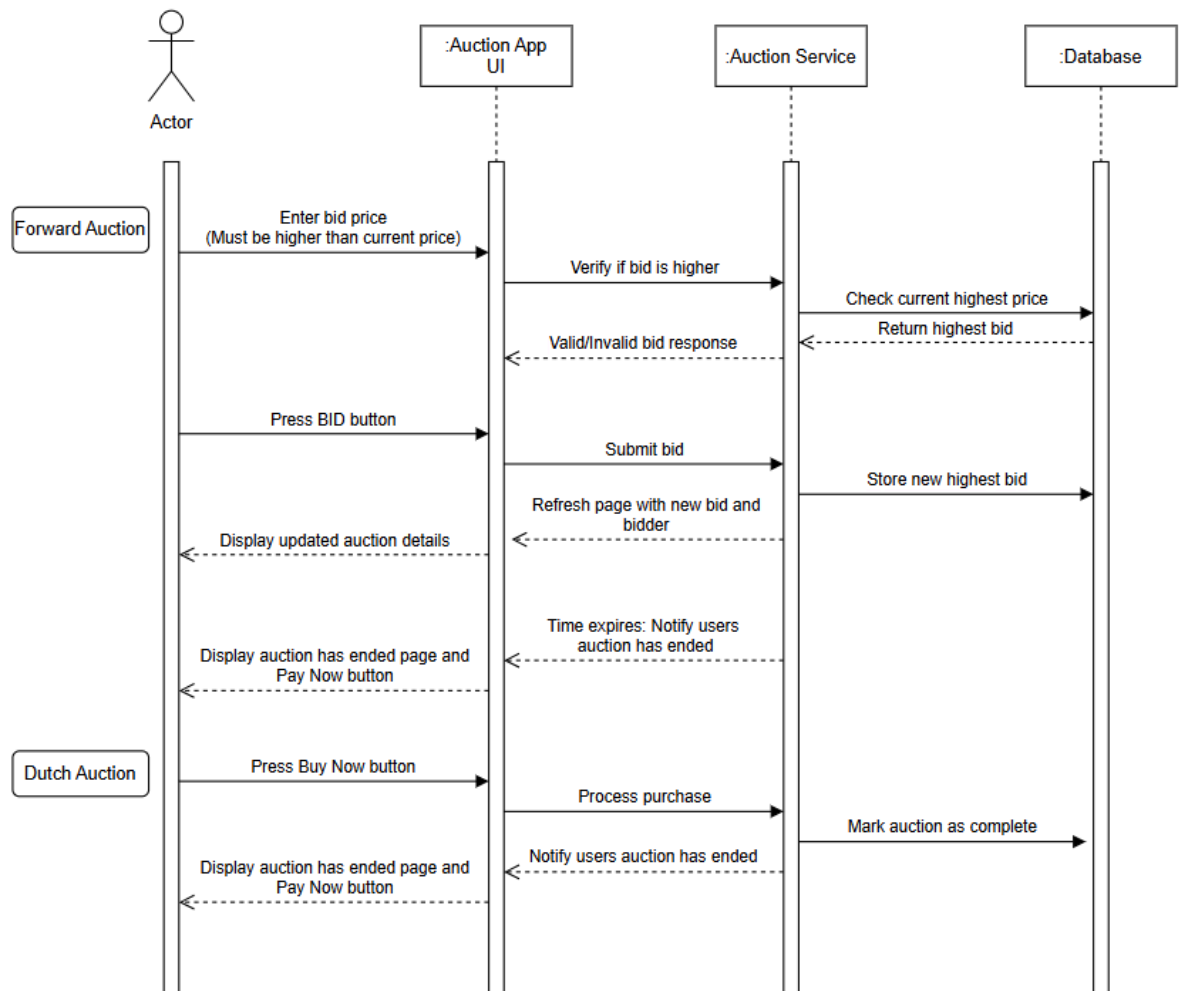


## Use Case 2: Browse Catalog

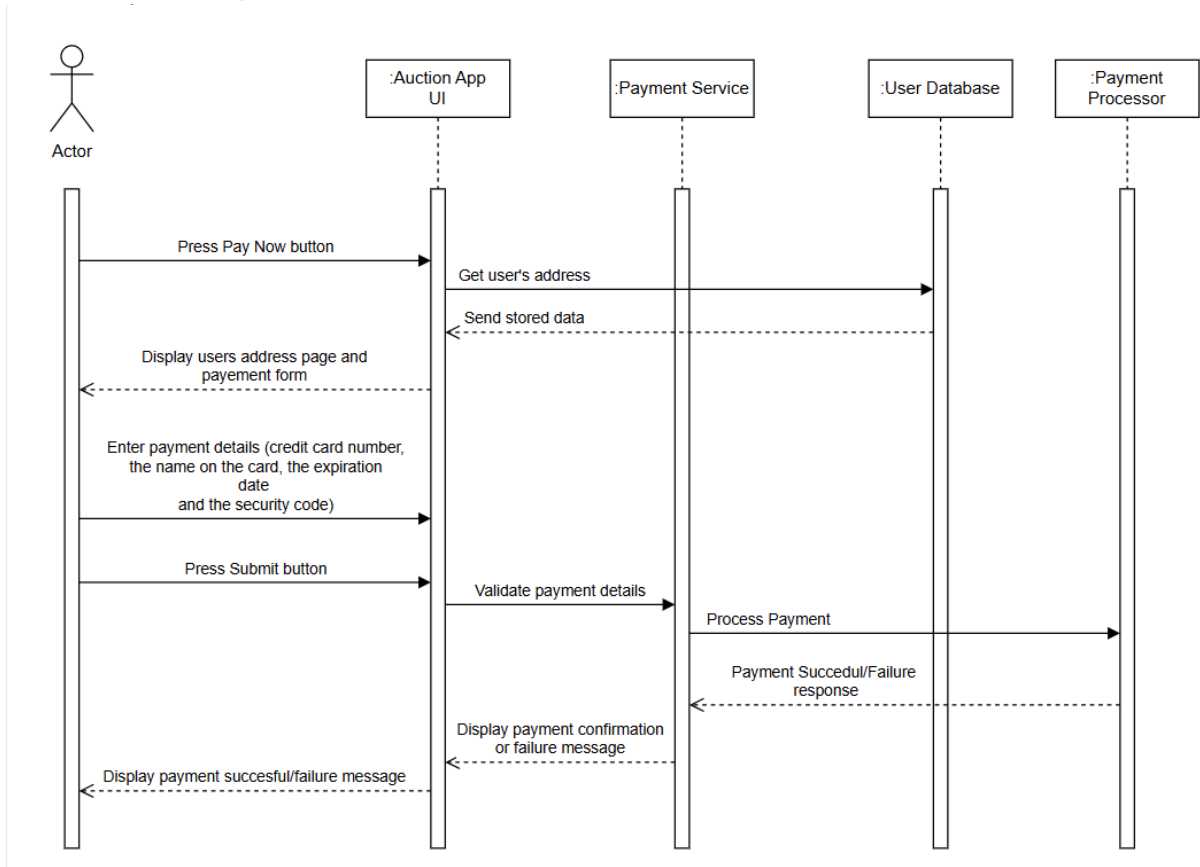




## Use Case 3: Bidding

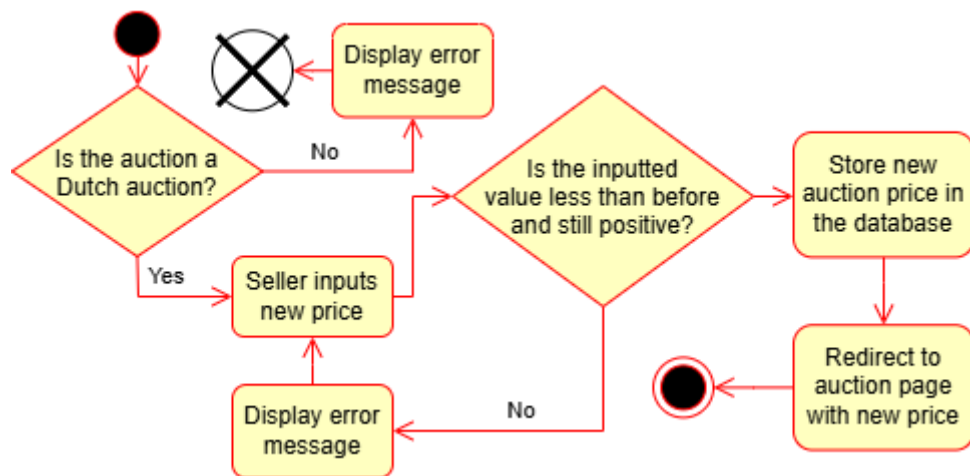


# Use Case 5: Payment

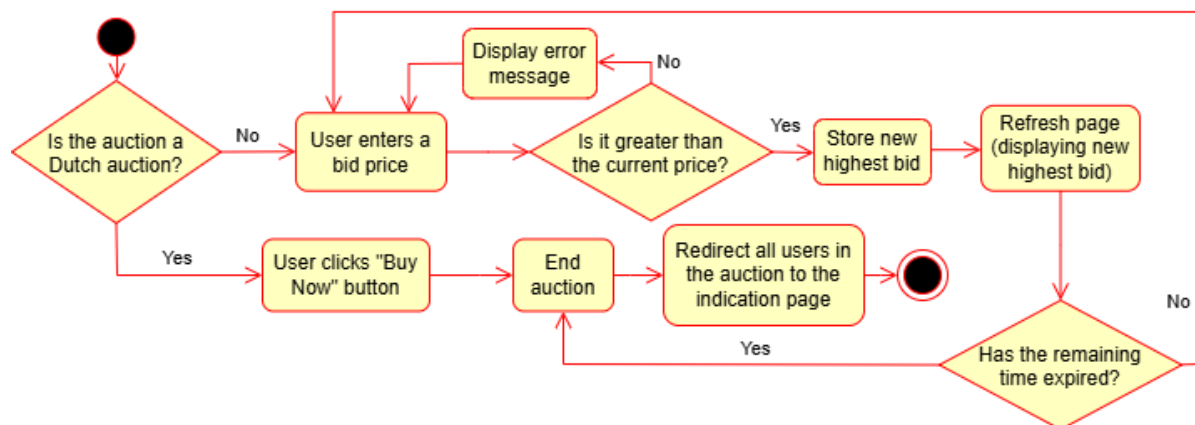


## 4 Activity Diagrams

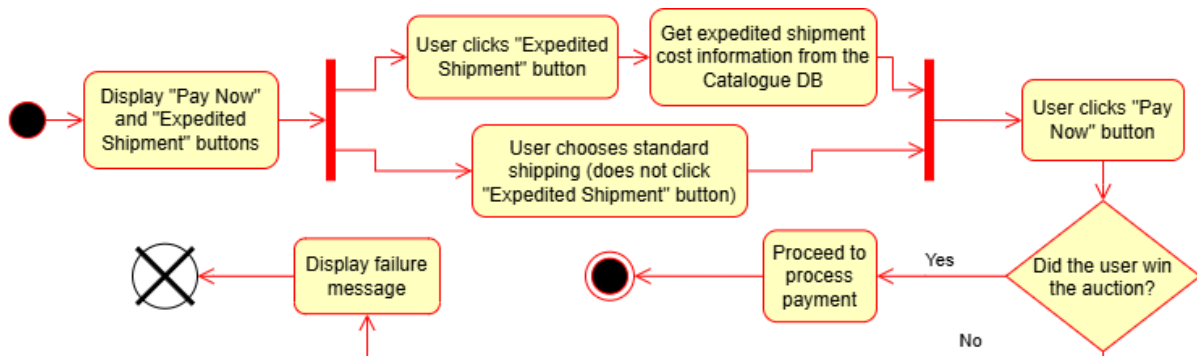
### Use Case 1: Dutch Auction Update



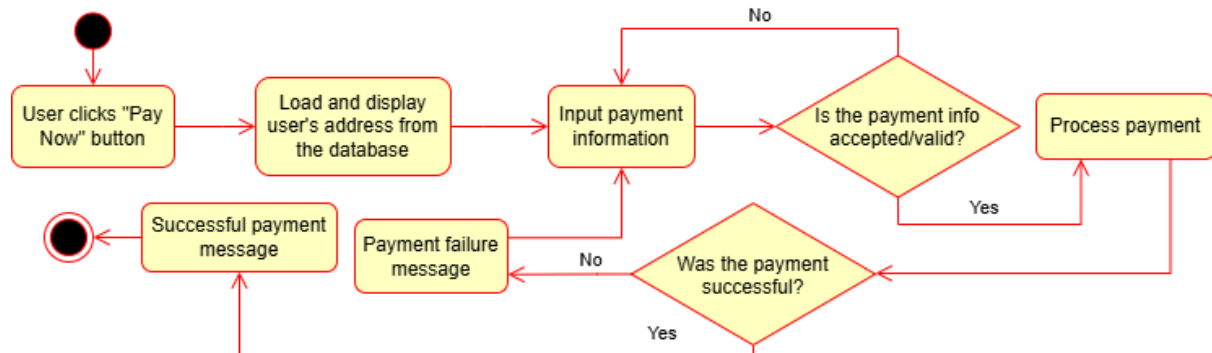
### Use Case 2: Bidding



### Use Case 3: Auction Ended



## Use Case 4: Payment



## 5 Architecture

The project implements an auction system using a monolith microservices architecture. The following key components are:

1. Infrastructure Layer
  - a. Docker and Containerization
  - b. Kafka and Zookeeper
  - c. PostgreSQL
  - d. Redis
2. Microservices Architecture
  - a. API Gateway
  - b. Authentication Service
  - c. Auction Service
  - d. Order Service
  - e. Notification Service
  - f. Auction Event Job Service
3. Frontend Architecture
  - a. Synchronous Communication
    - i. REST API
    - ii. Client to Server
    - iii. Service to Service
  - b. Asynchronous Communication

- i. Kafka Events (start, stop, create, notification event and bid)
- c. Real Time Communication
- d. Horizontal Scaling
  - i. Independent Scaling
  - ii. Event Partitioning

## **Infrastructure Layer**

Docker deploys our application inside containers. Each service runs in its own isolated container. The *docker-compose.yml* manages the entire backend stack. Services are configured with volume mounts for code persistence, environment variables, health checks to manage dependencies, port mappings for external access, and container dependencies. For example, services wait for Kafka and PostgreSQL to be healthy before running.

Kafka handles real-time updates, and it allows microservices to produce and consume events asynchronously. Zookeeper manages our Kafka.

PostgreSQL is our database of choice to store user, auction details, orders, and transaction history.

Redis stores data that are accessed frequently and as a by-product reduces the load of the database. It improves performance of the application by reducing loading time.

## **Microservices Architecture**

The API Gateway routes clients requests from the frontend to the appropriate backend services. For example, it forwards requests of clients trying to sign-in to the Authentication Service, then the Authentication Service formulates a response and sends it back to the API Gateway and updates the frontend.

The Authentication Service verifies user credentials, issues JWT tokens for session management and makes sure that only authenticated users can interact with the auction web app. For example, a user not registered or not signed in cannot view the auction listing.

The Auction Service manages the lifecycle of each auction item. It also handles the creation of auctions, bid validation for both forward and dutch auctions, and manages real-time price updates and each auction item's state (e.g., active, non-active, live, past, and more)

The Order Service manages how an order is processed and handles payment. For example, once an auction ends, this service would be responsible to create an order, process the payment and update the transaction history in the PostgreSQL database.

The Notification Service sends in-app real-time notifications to users (e.g., auction started, bid placed)

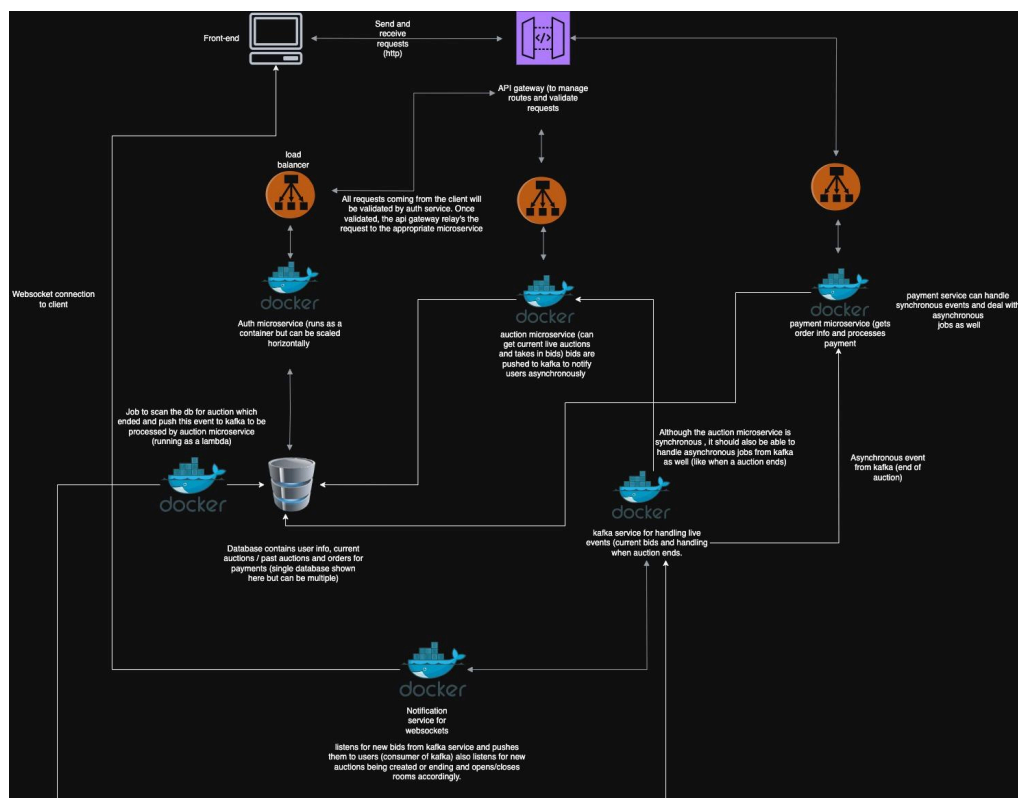
The Auction Event Job Service works in the background and processes jobs related to auctions. It handles the scheduling of each auction item and makes sure that updates happen at the appropriate time. For example, the auction state changes once time has elapsed.

## Frontend Architecture

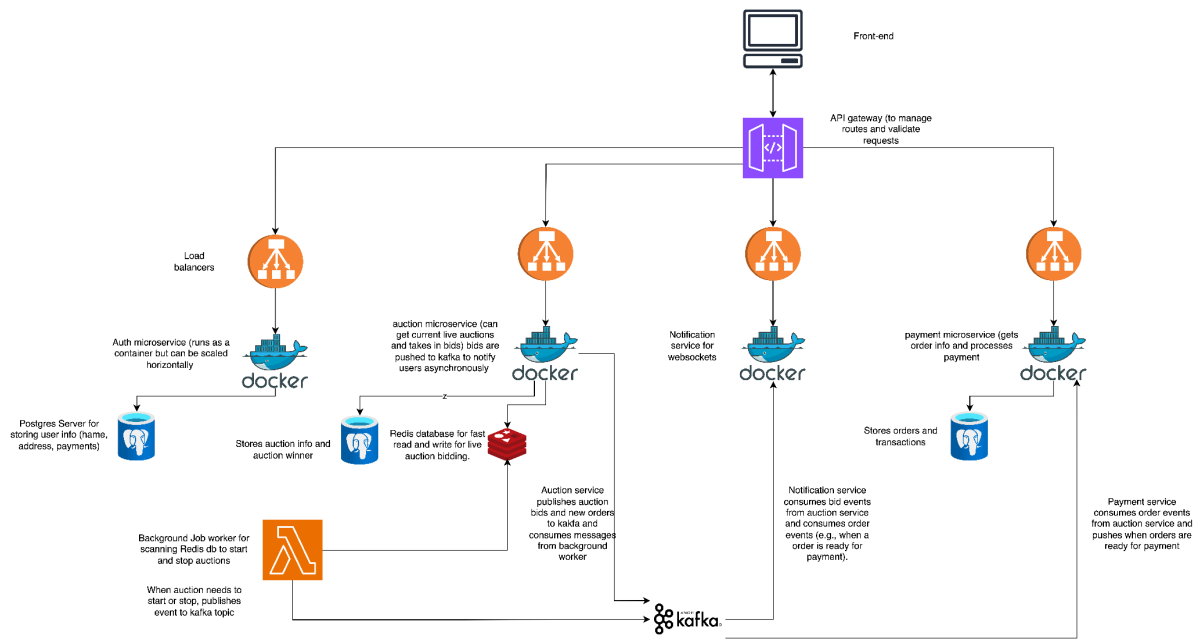
WebSockets gives our client and server the ability to communicate in real-time. We used it to update bidding information and auction state changes so that users can see auction updates as they happen without needing to refresh the browser every time.

Kafka allows for event driven communication. In brief, each event triggers a specific set of actions across the backend. For example, when a bid is placed, Kafka produces an event called 'auction.bid'. The Order and Notification Services then consumes the 'auction.bid' event and reacts by creating orders and notifying users.

Furthermore, services are designed to be stateless, which means it can be scaled horizontally. In other words, microservices can be easily and independently replicated to distribute traffic loads. Kafka also partitions events in a way that allows multiple events to be processed in parallel.



## Updated Diagram:



Modules			
Module Name	Description	Exposed Interface Names	Interface Description
API Gateway	Routes incoming client requests to appropriate services, enforces authentication and authorization policies.	API:RouteRequest API:VerifyRequest	API:RouteRequest:: Routes API requests to internal microservices. Defines how routes shall be structured.  API:VerifyRequest: Verifies a request using auth service
Auction Service	Manages auction creation, bidding, and auction lifecycle.	AS:CreateAuction, AS:PlaceBid, AS:EventListener AS:Search	AS:CreateAuction: Create  AS:PlaceBid: Place bids,  AS:EventListener: Listen for auction events  AS:Search: Search for auction
Payment Service	Processes payments for completed auctions and manages transactions.	PS:CreateOrder PS:VerifyPayment PS:GetOrder	PS:CreateOrder: Creates a order from a auction  PS:VerifyPayment: Verify payments and marks the order as completed  PS:GetOrder: Gets a order details
Notification Service	Sends real-time updates via WebSockets for bid updates and auction status changes.	NS:ManageRoom NS:PublishMessage	NS:ManageRoom: Able to create rooms and delete them  NS:PublishMessage:
Auth Service	Handles user authentication and session management.	AS2:Login AS2:Register AS2:Validate	Login: Login users and provide token  Register: Register users

			Validate: Validate tokens (JWT).
--	--	--	----------------------------------

Interfaces		
Interface Name	Operations	Operation Descriptions
API:RouteRequest	routeRequest(HttpRequest request)	Routes incoming HTTP requests to appropriate microservices.
API:VerifyRequest	verifyRequest(HttpRequest request)	Verifies the request is coming from a user who is logged in and has a valid token
AS:CreateAuction	createAuction(Item item, Date endingDate)	Creates a new auction
AS:PlaceBid	placeBid(String auctionID, double amount, User user)	Places a bid and verifies if the bid price is valid
AS:EventListener	listener()	Allows for communication between microservices
AS:Search	List<Auctions> searchAuction(String query)	Searches for a auction in the DB
PS:CreateOrder	createOrder(Auction auction, User winningUser)	creates a new order for this user using the winning auction
PS:VerifyPayment	verifyPayment(Order order, Payment payment)	Verifies the payment for the order and marks the order as complete
PS:GetOrder	getOrder(String orderID)	gets the Order from ID
NS:ManageRoom	createRoom(String auctionID) deleteRoom(String auctionID)	Creates or deletes a websocket room using the auctionID as the room ID
NS:PublishMessage	publishMessage(String auctionID, AuctionEvent event)	Broadcasts an event to all current participants in the room.
AS2:Login	login(String username, String password)	Gets user credentials, verifies them and provides a session
AS2:Register	register(User user)	Registers this user using info in user object
AS2:Validate	verifyRequest(HttpRequest request)	verifies that the request being sent is a valid request from a user who is logged in with a valid token. (For login and register routes verifying the request is not needed since the user won't be logged in)

## 6 Activities Plan

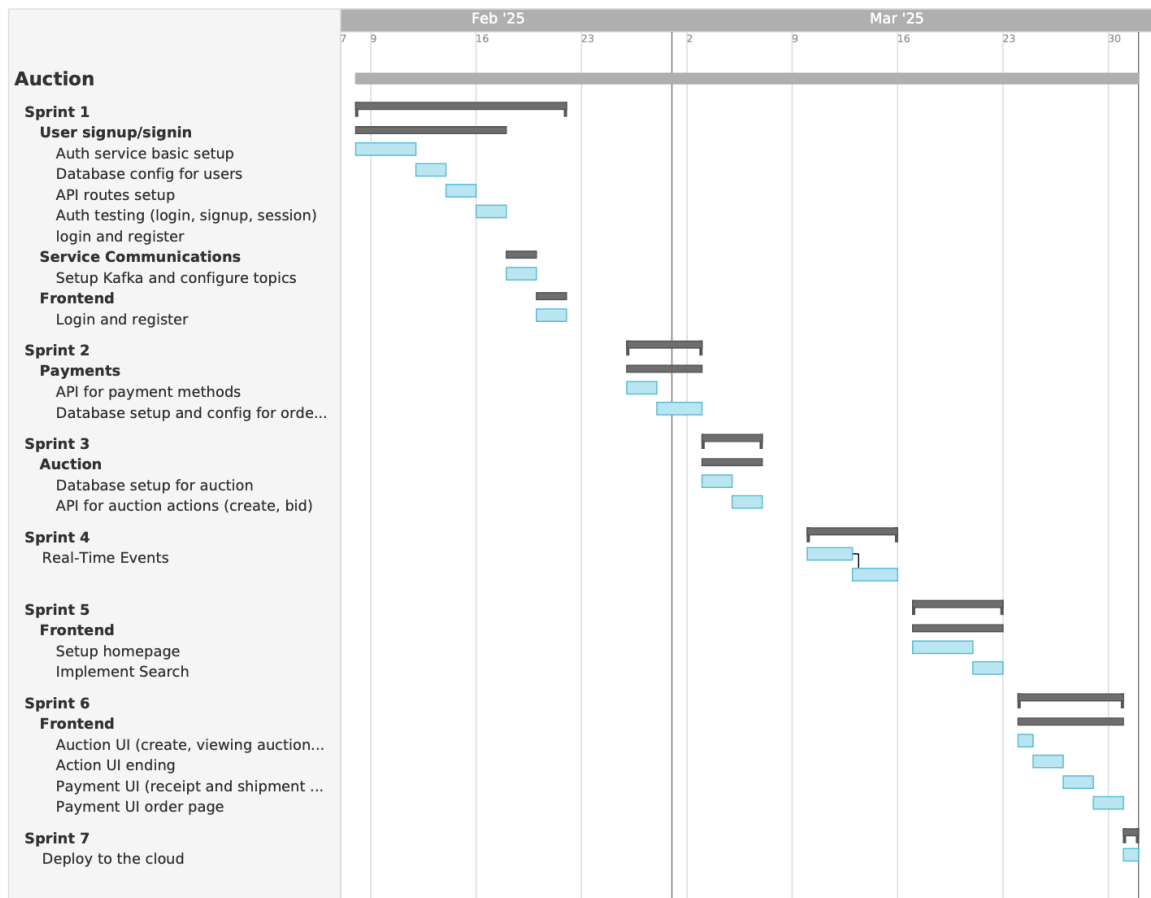
### 6.1 Project Backlog and Sprint Backlog

Task	Sprint	Epic
Auth Service basic setup	Sprint 1	User signup/signin



<i>Database config for users</i>	<i>Sprint 1</i>	<i>User signup/signin</i>
<i>API routes setup</i>	<i>Sprint 1</i>	<i>User signup/signin</i>
<i>Setup Kafka and configure topics</i>	<i>Sprint 1</i>	<i>Service communications</i>
<i>Auth testing (login, signup, session)</i>	<i>Sprint 1</i>	<i>User signup/signin</i>
<i>login and register</i>	<i>Sprint 1</i>	<i>Frontend</i>
<i>Database setup and config for orders</i>	<i>Sprint 2</i>	<i>Payments</i>
<i>API for payment methods</i>	<i>Sprint 2</i>	<i>Payments</i>
<i>Database setup for auction</i>	<i>Sprint 3</i>	<i>Auction</i>
<i>API for auction actions (create, bid)</i>	<i>Sprint 3</i>	<i>Auction</i>
<i>APIs for notification service (listening to kafka events)</i>	<i>Sprint 4</i>	<i>Real-Time events</i>
<i>Setup homepage</i>	<i>Sprint 5</i>	<i>Front-end</i>
<i>Implement search</i>	<i>Sprint 5</i>	<i>Front-end</i>
<i>Auction UI (create, viewing auctions both types dutch and forward)</i>	<i>Sprint 6</i>	<i>Front-end</i>
<i>Auction UI ending</i>	<i>Sprint 6</i>	<i>Front-end</i>
<i>Payment UI (receipt and shipment details)</i>	<i>Sprint 6</i>	<i>Front-end</i>
<i>Payment UI order page</i>	<i>Sprint 6</i>	<i>Front-end</i>
<i>Deploy to the cloud</i>	<i>Sprint 7</i>	<i>Deployment</i>

## 6.2 GANTT Chart



## 6.3 Group Meeting Logs

Present Group Members	Meeting Date	Issues Discussed / Resolved
-William Moran -Hayyaan Paurobally -Jay Raut -Daniel Santorelli	Wednesday, January 29th 60 minutes	<ul style="list-style-type: none"> <li>Decided to use nodejs for backend</li> <li>Split up roles</li> <li>William-Sequence diagrams</li> <li>Daniel-Activity Diagrams</li> <li>Hayyaan- Component diagram</li> <li>Jay- Architecture</li> </ul>
-William Moran -Hayyaan Paurobally -Jay Raut -Daniel Santorelli	Tuesday, March 4th, 2025 60 minutes	<ul style="list-style-type: none"> <li>Discuss how to split up code</li> <li>Set up GitHub repository</li> </ul>

-William Moran -Hayyaan Paurobally -Jay Raut -Daniel Santorelli	Saturday March 22nd, 2025 60 minutes	<ul style="list-style-type: none"> <li>• Discussed finishing touches on the project</li> <li>• Made preparations for the live demo</li> <li>• Discussed the third Software Specification Document</li> </ul>
-William Moran -Hayyaan Paurobally -Jay Raut -Daniel Santorelli	Tuesday April 1st, 2025 480 minutes	<ul style="list-style-type: none"> <li>• Finalized report (Architecture and Performance)</li> <li>• Completed web app with all features and use cases working</li> <li>• Updated Postman scripts</li> </ul>

## 7 Test Driven Development

### Use Case 1: Sign up and Sign-in

<b>Test ID</b>	RG-001
<b>Category</b>	Evaluation of a user creating an account and storing it on the DB
<b>Requirements Coverage</b>	UC1.1 Sign-up
<b>Initial Condition</b>	The system has been initiated and runs
<b>Procedure</b>	<ol style="list-style-type: none"><li>1. The user selects sign-up</li><li>2. The user provides the first name</li><li>3. The user provides the last name</li><li>4. The user provides a street address</li><li>5. The user provides the street number</li><li>6. The user provides city</li><li>7. The user provides country</li><li>8. The user provides a postal code</li><li>9. The user provides a username</li><li>10. The user provides a password</li><li>11. The user presses the submit button and information is stored in the system and is presented with the main UI window</li></ol>
<b>Expected Outcome</b>	The sign-up form closes, and the user is presented with the main UI window
<b>Notes</b>	<ul style="list-style-type: none"><li>• The user should have to confirm the password when creating it</li><li>• All information must be filled out to continue</li></ul>

<b>Test ID</b>	RG-002
<b>Category</b>	Authenticating the Sign-up
<b>Requirements Coverage</b>	UC1.1 Sign-up
<b>Initial Condition</b>	Sign-up details are correct
<b>Procedure</b>	<ul style="list-style-type: none"><li>- Store information to the User-Database</li><li>- Send verification email</li><li>- Create account</li><li>- Display catalogue items</li></ul>
<b>Expected Outcome</b>	A new user is now registered in the system and the user now use the login page for consequent sig-ins
<b>Notes</b>	

<b>Test ID</b>	RG-003
<b>Category</b>	Authenticating the Sign-up
<b>Requirements Coverage</b>	UC1.1 Sign-up
<b>Initial Condition</b>	Sign-up details are incorrect
<b>Procedure</b>	<ul style="list-style-type: none"> <li>- Enter invalid sign-up details to the User-Database</li> <li>- Do not send a verification email</li> <li>- Attempt to create an account with invalid details</li> </ul>
<b>Expected Outcome</b>	The Authentication microservice should verify and reject the invalid sign-up details and not store them i the Use-Database
<b>Notes</b>	

<b>Test ID</b>	RG-004
<b>Category</b>	User Authentication
<b>Requirements Coverage</b>	UC1.2 Sign-In
<b>Initial Condition</b>	<ol style="list-style-type: none"> <li>1. The user has already registered an account.</li> <li>2. The system is online and accessible.</li> </ol>
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. The user selects login</li> <li>2. The user provides a user name</li> <li>3. The user provides a password</li> <li>4. The user logs-in into the system and is presented with the main UI window</li> </ol>
<b>Expected Outcome</b>	the login form closes, and the user is redirected with the main UI window
<b>Notes</b>	

<b>Test ID</b>	RG-005
<b>Category</b>	User Session
<b>Requirements Coverage</b>	UC1 Sign-up and Sign in
<b>Initial Condition</b>	<ol style="list-style-type: none"> <li>1. The user has already registered an account.</li> <li>2. The system is online and accessible.</li> <li>3. The user has already signed in and has a valid session cookie on the browser.</li> </ol>

<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. The user navigates to the login page.</li> <li>2. The system detects an existing session from the valid session cookie.</li> <li>3. If the session is valid, the user is automatically redirected to the homepage without requiring re-authentication.</li> <li>4. If the session is expired or invalid the sign in / up pages will be accessible again</li> </ol>
<b>Expected Outcome</b>	If the session is valid, the user bypasses the login form and is redirected to the homepage
<b>Notes</b>	

<b>Test ID</b>	RG-006
<b>Category</b>	Authenticating the Sign-in
<b>Requirements Coverage</b>	UC1.1 Sign-in
<b>Initial Condition</b>	Sign-in details are incorrect
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. User enters invalid username and password</li> <li>2. Authentication service verifies if user is a registered client</li> <li>3. Find user in the User-Database</li> <li>4. Fail to Sign-in</li> </ol>
<b>Expected Outcome</b>	The authentication microservice should display an error message due to invalid credential details entered by the user.
<b>Notes</b>	

## Use Case 2: Browse Catalogue of Auctioned Items

<b>Test ID</b>	CAT-001
<b>Category</b>	Evaluation of searching for an item up for auction
<b>Requirements Coverage</b>	UC2-Successful-Item-Search
<b>Initial Condition</b>	The system has been initiated, runs and the user has already created an account
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. The user selects the login button</li> <li>2. The user provides a username</li> <li>3. The user provides a password</li> <li>4. The user logs into the system and is presented with the main UI window</li> <li>5. The user provides a keyword</li> </ol>

	6. A list of items matching the keyword appears
<b>Expected Outcome</b>	A table containing all the items up for auction that are related to the keyword is prompted to the user
<b>Notes</b>	<ul style="list-style-type: none"> <li>The user should be able to go back to the search bar if no items appear to their interest</li> </ul>

<b>Test ID</b>	CAT-002
<b>Category</b>	The user wishes to search for auctioned items
<b>Requirements Coverage</b>	UC2.2-Display-Auctioned-Items)
<b>Initial Condition</b>	The user has searched for their desired auctions using keywords, and the corresponding auctions have been found – they now need to be displayed for the user.
<b>Procedure</b>	<ol style="list-style-type: none"> <li>The search results webpage opens to a table of auction listings, with the columns: <ul style="list-style-type: none"> <li>Item Names</li> <li>Current Price</li> <li>Auction Type</li> <li>Remaining Time</li> <li>Select</li> </ul> </li> <li>The table is populated with listings in the database, and their information populates the respective columns (each row is one listing/auction)</li> <li>The user checks off which auctions they wish to bid on using the “Select” column</li> <li>The user pushes the “Bid” button when they are ready to bid</li> </ol>
<b>Expected Outcome</b>	The user has seen all the auctions related to their search and their relevant information. They can choose which of the listed auctions they wish to bid on.

<b>Test ID</b>	CAT-003
<b>Category</b>	Evaluation of user selecting an item to bid on
<b>Requirements Coverage</b>	UC2.3-Succesful-Item-Selection
<b>Initial Condition</b>	The system has been initiated, runs and the user is logged into their account
<b>Procedure</b>	<ol style="list-style-type: none"> <li>The user enters a keyword in the search box</li> <li>The user presses the search button</li> <li>A list of items related to the keyword is displayed, showing the item name, current price, auction type, remaining time, and a radio button</li> </ol>

	<ol style="list-style-type: none"> <li>The user selects only one item by pressing the radio button</li> <li>The user then presses the BID button on that page</li> <li>The user is then presented with an item information page where they will have the option to bid</li> </ol>
<b>Expected Outcome</b>	The table displaying all items related to the keyword closes and is presented with the specific item information page
<b>Notes</b>	<ul style="list-style-type: none"> <li>Users can only select one item at a time</li> <li>Should have a successful message appear when the item has been selected</li> </ul>

### Use Case 3: Bidding

<b>Test ID</b>	BID-001
<b>Category</b>	Evaluation of user participating in a forward auction bidding
<b>Requirements Coverage</b>	UC3.1-Successful-Forward-Auction-Bidding
<b>Initial Condition</b>	The system has been initiated, runs and the user has selected an item to bid on
<b>Procedure</b>	<ol style="list-style-type: none"> <li>The user selects one item to bid on</li> <li>The user is presented with that item information page</li> <li>The user provides a new bidding price in the designated box</li> <li>The user presses the BID button</li> </ol>
<b>Expected Outcome</b>	The page is refreshed, displaying the user's ID and bid amount
<b>Notes</b>	<ul style="list-style-type: none"> <li>The user's bidding price must be higher than the current price</li> <li>The user has to wait until the bidding time has expired to win the auction</li> <li>Once the time ends, all users bidding on the item will receive a notification</li> </ul>

### Use Case 4: Auction Ended

<b>Test ID</b>	AUC-001
<b>Category</b>	Behavior of the website for users as an auction is ending
<b>Requirements Coverage</b>	UC4-Auction-Ended
<b>Initial Condition</b>	The user is signed in and on a web page for an auction. The auction is now ending.
<b>Procedure</b>	<ol style="list-style-type: none"> <li>The auction ends (see Notes section) and the user is on the auction web page</li> </ol>



	<ol style="list-style-type: none"> <li>The user decides whether to click the option “Expedited Shipping” button</li> <li>The user clicks the “Pay Now” button</li> <li>The user who won proceeds to the payment process, and users who did not win are shown an error message</li> </ol>
<b>Expected Outcome</b>	All users who were on the auction web page are redirected to the “Pay Now” indication web page after the auction has ended.
<b>Notes</b>	The auction ends differently depending on what type of auction it is. A Dutch auction ends instantly when a user clicks “Buy Now” and buys the item, and a Forward Auction ends once the timer reaches zero.

## Use Case 5: Payment

<b>Test ID</b>	PYT-001
<b>Category</b>	Payment of auction goods
<b>Requirements Coverage</b>	UC5. Payment
<b>Initial Condition</b>	<p>The auction has ended, and the user has won the auction.</p> <p>The payment service is active.</p> <p>The user has a valid payment method linked to their account or they can enter new payment methods</p> <p>The user has a valid shipping address stored in the database from sign-up.</p> <p>The user is logged in.</p>
<b>Procedure</b>	<ol style="list-style-type: none"> <li>The system displays the auction details, including: <ul style="list-style-type: none"> <li>Winning bid amount</li> <li>Shipping cost</li> <li>Total payable amount</li> <li>User’s shipping address</li> </ul> </li> <li>The user selects the “<b>Pay Now</b>” button.</li> <li>The system redirects the user to the payment page</li> <li>The user enters the payment details, including: <ul style="list-style-type: none"> <li>Credit card number</li> <li>Name on the card</li> <li>Expiration date</li> <li>Security code (CVV)</li> </ul> </li> <li>The user selects the “<b>Submit</b>” button.</li> <li>The system validates that all required fields are filled.</li> <li>The system processes the payment and verifies the transaction.</li> <li>If successful:</li> </ol>

	<ul style="list-style-type: none"> <li>○ The auction status updates to <b>"Paid."</b></li> <li>○ User is given a receipt</li> </ul> <p>9. If payment fails:</p> <ul style="list-style-type: none"> <li>○ The system displays an <b>error message</b> and allows the user to retry.</li> </ul>
<b>Expected Outcome</b>	<p>The payment is successfully processed, and the auction status changes to completed or paid</p> <p>The user receives a payment confirmation (receipt)</p>
<b>Notes</b>	

## Use Case 6: Receipt Page and Shipment Details

<b>Test ID</b>	RPS-001
<b>Category</b>	Payment and Shipment Confirmation
<b>Requirements Coverage</b>	UC6. Receipt Page and Shipment Details
<b>Initial Condition</b>	<p>The user has successfully completed payment.</p> <p>The payment service has confirmed the transaction.</p> <p>The system has retrieved shipping details from the Payment Database</p>
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. After payment confirmation, the system redirects the user to the Receipt Page</li> <li>2. The page displays: <ul style="list-style-type: none"> <li>● Payment Confirmation Message (e.g., "Your payment was successful.")</li> <li>● Total Amount Paid (Item price + Shipping cost)</li> <li>● User's Shipping Address (Pre-filled from database)</li> <li>● Shipment Information with days</li> </ul> </li> </ol>
<b>Expected Outcome</b>	The receipt page is displayed with payment confirmation and shipping details.
<b>Notes</b>	

## Use Case 7: Sell Item

<b>Test ID</b>	SI-001
<b>Category</b>	The user puts up a new item for sale.
<b>Requirements Coverage</b>	UC7-Sell-Item
<b>Initial Condition</b>	Initial conditions required for the test case to run (e.g. the system has been initiated and runs)

<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. The user selects “New Listing”</li> <li>2. The user selects whether the listing will be a Dutch or Forward Auction</li> <li>3. The user inputs the name and description of the item, and uploads images of the item</li> <li>4. The user selects the starting bid price and the duration of the auction.</li> <li>5. The user saves their changes and their new auction is put up on the website for other users to find.</li> </ol>
<b>Expected Outcome</b>	The user has successfully created a new listing/auction, and their new auction is available on the website for other users to find.
<b>Notes</b>	The type of auction the user selects will change certain aspects of the information uploading process, such as selecting a starting bid price and setting the auction’s duration.

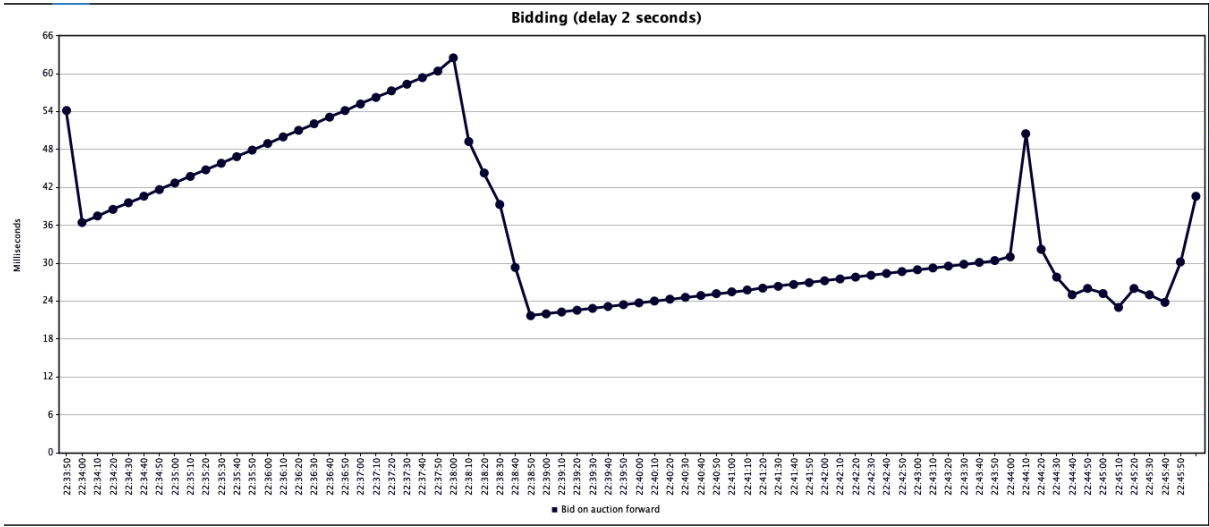
<b>Test ID</b>	SI-002
<b>Category</b>	UC7- Sell-Item
<b>Requirements Coverage</b>	Check if item for sale can be found in the Database
<b>Initial Condition</b>	A Database must already exist for an item to be stored
<b>Procedure</b>	<ul style="list-style-type: none"> <li>- User selects item of choice</li> <li>- User is redirected to a page to enter item details</li> <li>- User clicks sell item button</li> <li>- User is notified “Item uploaded”</li> </ul>
<b>Expected Outcome</b>	Item is uploaded in the Item-Database
<b>Notes</b>	

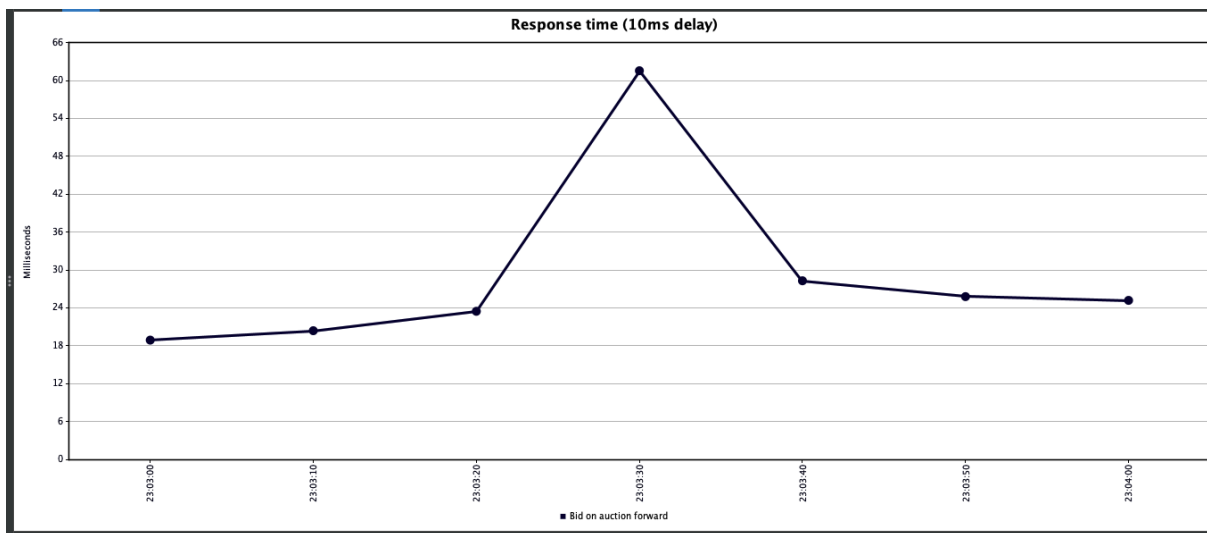
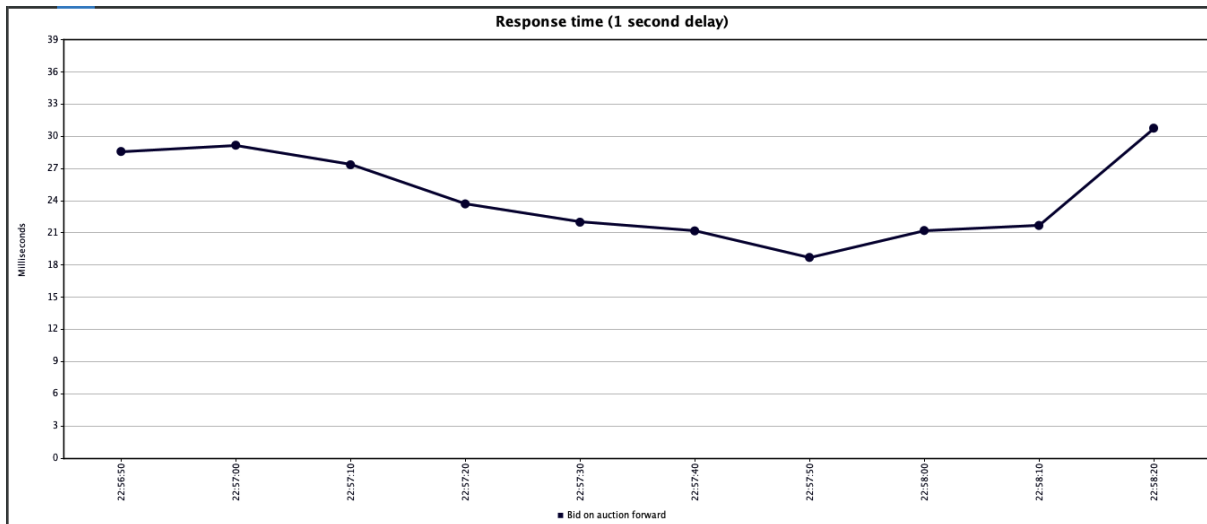
## Use Case 8: Dutch Auction Update

<b>Test ID</b>	DAU-001
<b>Category</b>	A seller updates the price of their existing Dutch auction
<b>Requirements Coverage</b>	UC8-Dutch-Auction-Update
<b>Initial Condition</b>	The user/seller is signed in, and has an existing Dutch auction currently open.
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. The user enters the page for their Dutch auction</li> <li>2. The user clicks the “Edit Auction Price” button</li> <li>3. The user inputs a new auction price</li> <li>4. The price is updated in the database</li> </ol>

	5. The auction web page is refreshed
<b>Expected Outcome</b>	The auction web page is refreshed, showing the newly updated price
<b>Notes</b>	The new price is accepted and considered valid if it is less than the current price but still a positive value.

## 8 Performance Report





## 9 Testing Report and Security Vulnerabilities

### Tested Security Vulnerabilities:

- Testing jwt authenticity to make sure tokens are only signed by the auth service
- User passwords are hashed in database
- Protection from sql injection using prepared statements

### Untested Security Vulnerabilities:

- Untested ssl, tls encryption between client and server
- Allows attackers to view data in flight